Proceedings of the ASME 2014 Dynamic Systems and Control Conference
DSCC2014
October 22-24, 2014, San Antonio, USA

DSCC2014-6201

ALGORITHMIC OPTIMIZATION OF INVERSE KINEMATICS TABLES FOR HIGH DEGREE-OF-FREEDOM LIMBS

Katie Byl[*]
Marten Byl
Brian Satzinger
Robotics Laboratory
ECE / ME Depts.
University of California
Santa Barbara, California 93106
Email: {katiebyl, marten.byl} @gmail.com

ABSTRACT

This work addresses the problem of resolving kinematic redundancy in legged robots, with the dual goals of maintaining a large reachable workspace and of achieving fast end effector motions in task space. In particular, for robots with four or more legs, gait planning allows for considerable flexibility in the orientation of a stance limb with respect to both body orientation and the ground. By appropriately commanding pitch, roll and yaw of the end effector as it moves relative to the body coordinate frame, one can increase the volume of space the feet can reach and thus allow the robot to negotiate larger terrain obstacles. At the same time, motions of the foot in task space should be done rapidly, given the joint velocities of the limbs. In this paper, we focus on RoboSimian, a robot with four identical limbs designed for dual use in manipulation and locomotion tasks, which was designed at Jet Propulsion Labs (JPL) for the DARPA Robotics Challenge (DRC). We present both heuristic guidelines and a novel, gradient-based algorithm for developing rules to set the inverse kinematics (IK) solution for the seven joint angles of a limb, allowing us to prescribe joint solutions rapidly through the use of an IK look-up table.

NOMENCLATURE

ϕ_{ee}	Roll (about x axis) of the end effector.
ψ_{ee}	Yaw (about z axis) of the end effector.
θ_{ee}	Pitch (about y axis) of the end effector.
q_i	The i^{th} joint angle, starting proximal to the body and going outward in a serial, kinematic chain.
C	Cost function for end effector motion, to be minimize.
$w_{\phi,i}$	Scaling parameter (weighting) on roll, for i^{th} joint.
$w_{\psi,i}$	Scaling parameter (weighting) on pitch, for i^{th} joint.
$w_{\theta,i}$	Scaling parameter (weighting) on yaw, for i^{th} joint.
x_{ee}	Cartesian end effector x coordinate; body coordinate frame.
y_{ee}	Cartesian end effector y coordinate; body coordinate frame.
z_{ee}	Cartesian end effector z coordinate; body coordinate frame.
R_{ee}	Polar coordinate for end effector radius, relative to limb coordinate frame.
Θ_{ee}	Polar coordinate for end effector angle, relative to limb coordinate frame.
Z_{ee}	Polar coordinate for end effector height, relative to limb coordinate frame.

Cartesian end effector (e.e.) coordinates and Euler angles are defined *relative* to the coordinate system of the chassis of the robot in space, referred to as the *body coordinate system*. Euler angles in this work use x-y-z ordering, i.e., with roll (ϕ), pitch (θ), and yaw (ψ) rotations performed sequentially about local axes. Polar coordinate definitions of the end effector are relative to limb coordinate frame, based at the root (proximal end) of the serial kinematic chain of a limb. The right, front limb is used in

[*]Address all correspondence to this author.

all analyses in this work; with solutions for other limbs simply being related in an obvious manner, through symmetry.

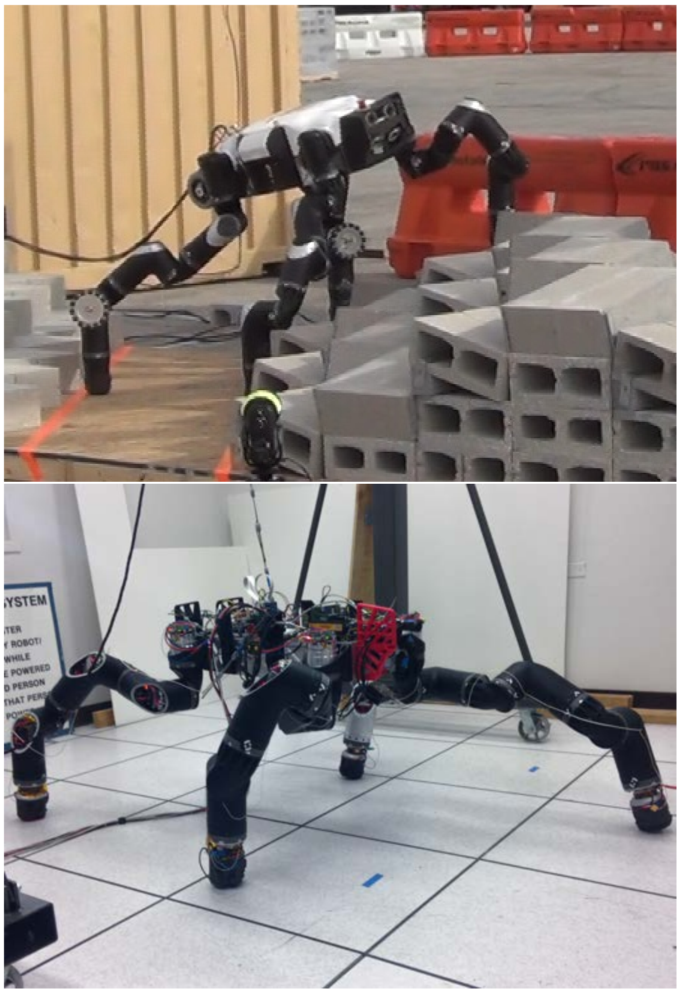

FIGURE 1. RoboSimian during the DARPA Robotics Challenge. Dexterity of the limbs allows the robot the reach a large workspace on rough terrain (top), and adding roll and pitch to the end effector enables even longer footsteps (bottom).

1 INTRODUCTION

Planning smooth, efficient, and collision-free kinematic joint trajectories for high degree-of-freedom (DOF) limbs remains an open challenge within both robotics and computer-generated imagery (CGI). In both fields, there is a long history of generating motion plans based upon inverse kinematics (IK) tables developed using both physical and heuristic guidelines [1,2]. In this paper, we present approaches for generating and optimizing inverse kinematics (IK) tables for the high-DOF limbs of Robosimian to walk in a quadruped gait, as depicted on both rough terrain and in the lab in Fig.1.

In recent years, there have been a number of quadruped robots developed for legged locomotion including (but certainly not limited to) Boston Dynamics' LittleDog and BigDog, IIT's HyQ, and ETH-Zurich's Starleth. Multiple teams have developed both motion planning and control algorithms that allow these robots to successfully negotiate rough terrain [3–7]. Many quadrupeds (e.g., LittleDog) have only 3 actuated joints per leg, so that there is usually at most one kinematic solution for a desired 3 degree-of-freedom (DOF) foot position (x,y,z), with a closed form analytic expression available to calculate joint angles. End effector trajectories can then be calculated both to avoid obstacles and to maintain stability while respecting joint displacement and velocity limits. For walking robots with higher DOF legs, however, additional constraints need to be added to the kinematic solver to resolve kinematic redundancies. The problem of solving redundant kinematics has long been studied as a generalized problem [8–12]. For walking robots, [13, 14] provide example solutions for multi-legged robots with 4 DOF legs. Similarly, solutions for 6 or higher DOF limbs used for manipulation have been developed [15, 16]. While locomotion and manipulation share many common goals, the constraints used to resolve redundant kinematic solutions may vary significantly with respect to the two tasks. In particular, orientation of the end effector is typically more critical for manipulation or bipedal walking than for point-foot locomotion. Correspondingly, humanoid limbs commonly have 6 or 7 DOFs while quadruped legs frequently have only 3 or 4. In this work, we study the problem of exploiting high-DOF limbs (designed to be capable of humanoid dexterity) toward improving feasible step length and step speed in statically-stable quadruped walking.

The rest of this paper is organized as follows. Section 2 describes the kinematics and dynamics of RoboSimian, focusing on the advantages and challenges resulting from its design. Section 3 presents heuristic guidelines we have used in determining initial inverse kinematics (IK) tables for rough terrain locomotion. This is followed by a description of gradient-based algorithm to build IK tables in a more systematic way, presented in Section 4. We present results for this algorithm in Section 5, which starts with a simple and intuitive 1D (x motion only) table for end effector yaw, only, and develops successively more complex tables, eventually providing solutions for roll, pitch, and yaw as functions of the 3D (x, y, z) end effector coordinates. Here, we also compare the resulting speed of completing various paths in task space using these solutions. In Section 6 we highlight similarities and differences between our heuristic and algorithmic IK table solutions briefly and outline key aspects to be addressed in future work. Finally, Section 7 gives a brief summary of the paper.

2 KINEMATIC PLANNING FOR ROBOSIMIAN

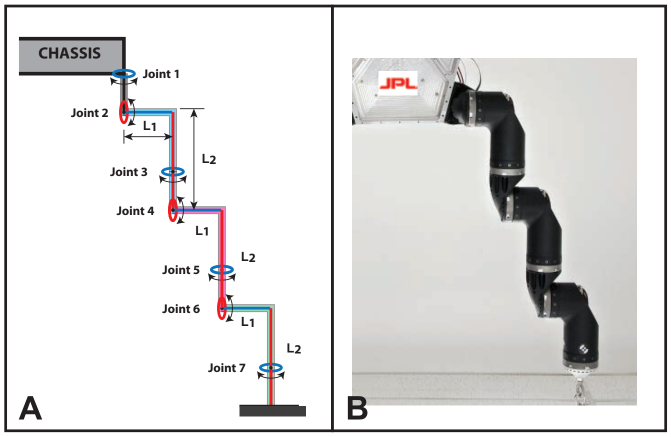

FIGURE 2. Each RoboSimian limb, shown in schematic in A and photo in B, consists of 7 links forming 3 identical right triangles with 7 DOF. Note: Joint 1 is not visible in B due to a 90 degree rotation about Joint 2.

Robosimian is a robot with four identical limbs that can each be used for both manipulation or locomotion. Each limb, shown in Figure 2, has seven actuated degrees of freedom (DOFs). Six of these can be grouped (pairwise) into 3 identical right triangular links with $L_1 = 0.117$ (m), $L_2 = 0.286$ (m) and a resulting hypotenuse of 0.309 (m). Thus, the reachable volume for each limb is contained within a sphere of $R = 0.927$ (m). The seventh joint twists the "wrist" at the distal-most location of the limb. Each limb is designed to support up to 1/2 the weight of robot at full extension. In the alpha prototype "Clyde", joint velocity is limited to 1 rad/sec which both restricts the achievable walking speed and prevents "dynamic walking" gaits. An enhanced beta prototype is currently in development.

Planning motions for RoboSimian's limbs presents several challenges. Because each of the four identical limbs has seven rotational actuators available to set the 6-DOF position (x_{ee}, y_{ee}, z_{ee}) and orientation $(\phi_{ee},\ \theta_{ee},\ \psi_{ee})$ of the end effector, Robosimian has kinematic redundancy that allows for flexibility in planning manipulation tasks. However, this redundancy also increases the complexity of planning for locomotion tasks, particularly since the orientation of a quadruped foot contact on the ground is not a hard constraint. Our general goal in this work is to create a set of rules for RoboSimian to reduce the planning dimensionality for quadruped walking from a 7-DOF one to a 3-DOF one, planning all joints as a function of $(x_{ee},\ y_{ee},\ z_{ee})$ via design of appropriate look-up tables. IK look-up tables of such solutions for end effector poses must avoid self-collisions and ensure that smooth trajectories in position of the end effector result in smooth, continuous joint trajectories. They should exploit kinematic redundancy to allow for rapid end effector motion and, correspondingly, for faster walking. Also, the reachable workspace for a foot during walking must be large to enable walking on extreme terrain, where footholds may be far apart and at different heights. To accomplish this we have created a set of IK tables intended for walking in simple environments (e.g., flat open terrain). To plan walking on more complex terrain, we employ a randomized search algorithm in conjunction with our IK tables to search for joint trajectories that allow the limbs and body to perform collision free movements. Using the IK tables allows us to reduce the dimensionality of the kinematic planning search algorithm from 28 DOFs to 16 DOFs (i.e., 14 actuated joints for two limbs, plus passive roll and pitch of one contact point) by using IK table solutions for two of the four limbs, so that these two "slave" limbs simply maintain ground contact at a fixed location and reconfigure in response to shifts in the body posture. This approach is described in much more detail in [17] and [18].

Finally, it is important to make some final comments on the IK solutions for RoboSimian. First, we note that although there are seven actuators per limb, only the first six determine the position and orientation of the most distal rigid limb link, shown in green in Figures 3 and 4. For locomotion, the last actuator merely twists the end effector to allow the foot to avoid yaw rotation on the ground while the last rigid link itself reorients. Even this 6-to-6 mapping allows for up to eight distinct IK solutions, which are shown in Figure 3. Each family has its own dexterous workspace, and there are discrete "jumps" in solutions in any one family, so staying within one family does not guarantee the smoothness of resulting joint trajectories. These redundant solutions exist because there are (up to) three choices akin to "which way to bend an elbow" that can be made in solving directly for an IK solution. The poses in Figure 3 are arranged so top and bottom images represent an "elbow flip" of the red and blue limb segments for a fixed alignment of the red and green limb segments.

In this work, we use only IK families 3 and 8, shown at the far right in 3, because empirical testing indicates they generally provide the best configurations to avoid self-collisions and collisions with terrain while maximizing reach. Our heuristically-designed table uses IK-3, based on initial intuition, while results for our algorithms focus on IK-8. Figure 4 shows IK-3 and IK-8 solutions overlaid on one another (center). Graphically, each IK solution is found by determining intersection points along two spheres, also shown in Figure 4 and determining which of these points results in a 90-degree intersection between the blue and red segments of the limb.

The change in IK family choice was made after we tested our algorithms across various IK families. We found both 3 and 8 provided particularly large reachable workspaces; however, we have focused throughout our algorithmic work primarily on IK-8 for the following reasons. First, we found that "seeding" feasible

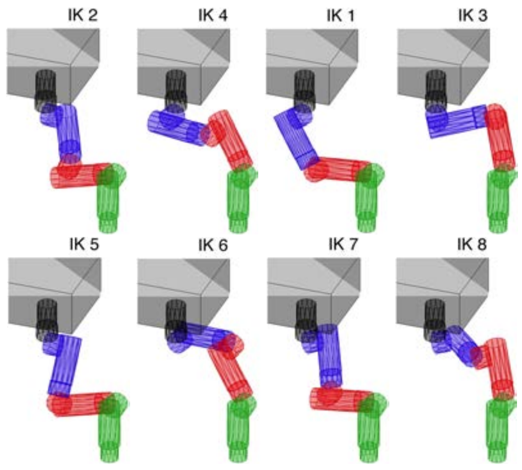

FIGURE 3. Redundant IK solution families for RoboSimian.

solutions to initialize tables for IK-3 is more challenging than for IK-8. Searching for initial "seed" solutions is an important topic for future work but is outside the focus of this paper, and we believe our results can be presented most clearly using IK-3. Second, we found empirically that solution trajectories using IK-8 allowed for faster overall limb motions, which is our goal in generating tables. We note that the ability to quantify one solution's performance versus another's is an added benefit of using an algorithmic process to develop tables.

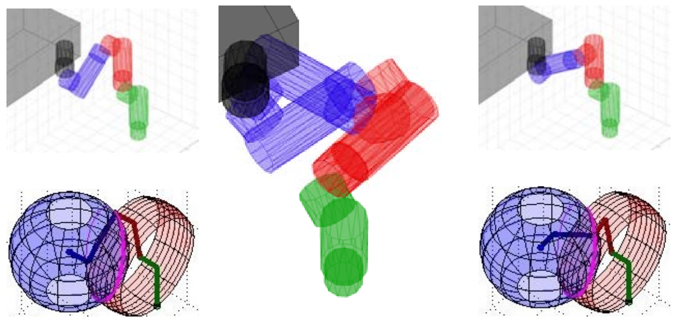

FIGURE 4. 3rd (left) vs 8th (right) IK solution. The center image overlays the two solutions.

3 HEURISTIC INVERSE KINEMATIC PLANNING

We have developed several IK look-up tables heuristically, each designed with specific design requirements. For example, on rough terrain, we wish to maximize the reachable workspace of the end effector, while walking through a doorway requires that the links of the limbs are preferentially aligned fore and aft and/or underneath the body, rather than out to the sides. For any table solution, the most important criteria are to ensure smoothness and avoid self-collisions. In practice, this is achieved over a large workspace primarily through planning the yaw of the most distal links in the limbs (shown in green), and to a lesser extent by commanding a slight pitch angle as the limb reaches far forward. Figure 5 illustrates these ideas graphically.

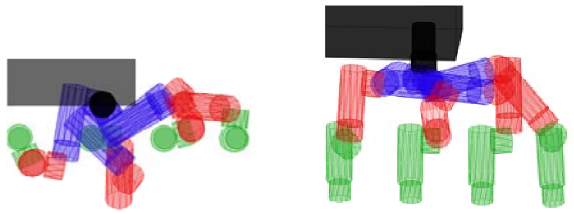

FIGURE 5. Example solutions from our hand-build IK table. The same pose is shown overhead (left) and from the side.

In this work, we focus on IK tables specific to rough terrain walking. Our heuristic table was created by first developing a set of rules for the yaw of the end effector in a polar coordinate system, centered at the armpit of the right, front limb. The yaw was set essentially to "unfurl" the entire limb as it moved radially outward and to command some pitch at large radii. These heuristic guidelines are particularly effective at producing a very large reachable workspace. Solutions for different θ values then require modifying only the first actuator value, q_i. Near the armpit, however, a polar coordinate solution requires dq_i/dt to become arbitrarily large, so we use a simple Cartesian solution at small radii values ($r \leq 0.15$ m), blend the two smoothly inside an annular region ($0.15 < r < 0.45$), and use the polar solution for $r \geq 0.45$ meters.

4 ALGORITHMIC APPROACH TO BUILD IK TABLES

The process of developing each IK table is time consuming. Also, while our heuristics focus on increasing the size of the workspace and on ensuring smooth joint trajectories, they do not directly address the issue of enabling fast motions. This is particularly important for us, because RoboSimian has significant joint velocity limits (1 rad/s). Typically walking speeds from our planners are on the order of 6 ft/min (0.0305 m/s), but if we optmize a walking gait, we know we can achieve walking speeds over 15 ft/min (0.076 m/s) [19]. This motivates the development of an automated algorithm to speed up development time for new IK tables, toward optimizing task-specific cost functions.

In this section, we describe a novel algorithm to build deterministic look-up tables for joint angles as functions of Cartesian end effector location. Our approach begins with some initial definition of end effector orientation (roll, pitch and yaw) and uses a local gradient search to optimize these choices. Figure 6 illustrates the high-level concept. Details follow below.

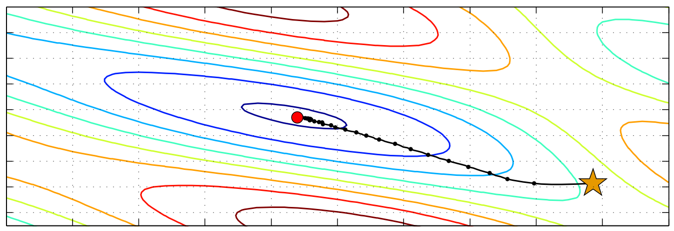

FIGURE 6. At each step in the algorithm, and for each element in an n-element mesh, we calculate the magnitude of the change on a total cost function that a small perturbation in each of m orientation variables of the end effector at the mesh point will have. The set of all $N = n \times m$ such partial derivatives forms an N-dimensional vector, determining a "direction" to change the entire mesh at this iteration. The concept is entirely analogous to a standard gradient search, as depicted above. If we start at the yellow star, we would proceed downhill in small steps, each proportional in length to the local gradient, until arriving near the local minimum shown as the red dot. Black dots highlight every 5^{th} step in the walk, to illustrate how the speed of progress slows down as the local slope gets more shallow.

First, we select a particular region of interest and create a finite, Cartesian mesh of points for end effect location. In this work, we consider both a 1D mesh, varying only x_{ee} along a line in space, and a 3D mesh, across (x_{ee}, y_{ee}, z_{ee}). The 1D case is examined to more clearly illustrate the method, while the 3D case is of more practical, real-world use.

Rather than searching directly for joint angles at each point in the mesh, we first develop an intermediate table that assigns a deterministic end effector pose, (ϕ_{ee},θ_{ee}, ψ_{ee}), to each mesh point. Each mesh point is thereby associated with a full 6-DOF pose, and the combination of 6-DOF pose and IK family maps uniquely to at most one IK solution. To initialize the algorithm, we calculate the full solution for joint angles for the currently-defined orientation, (ϕ_{ee},θ_{ee}, ψ_{ee}), for each mesh point, to ensure the initialization results in a feasible set of solutions.

In building the IK tables, we can optionally allow for some subset of the roll, pitch and yaw to be left as open parameters, while the rest are constrained. For example, since the yaw angle is the most critical element in our heuristic tables, roll and pitch of the end effector can simply be constrained to keep the most distal link of the limb aligned with the local z axis of the body coordinate system. Searching over all three (roll, pitch and yaw) should intuitively result in better IK table solutions, but it might also increase the time for the algorithm to run. We investigate both options in this work.

For an n-element mesh in which m of end effector orientations are free to change, the algorithm is essentially picking a direction in an $N = n \times m$ dimensional space. At each step, the algorithm calculates a set of partial derivatives, which are combined appropriately to determine a local "direction" in this N-dimensional space in which to modify the currently-defined set of end effector orientations for each mesh point. The look-up values for the m orientations for each mesh element are then updated, and the results are tested for kinematic feasibility. If the new solution is not feasible for a particular node, its previous solution is retained. All feasible solutions are adopted and the new IK solutions are calculated and saved. This process is repeated until the solution converges within some tolerance threshold.

To determine the "direction" in which to update the gradient search, we first calculate a set of weights ($w_{i,j,k}$) describing the effect that a change in the j^{th} orientation component, α_j would have on some total cost function, $C = \sum_{\forall i} c_i$:

$$\begin{aligned} w_{i,j,k} &= \frac{\partial c_i}{\partial \alpha_j} \\ i &\in \{1,2,...N\} \\ j &\in \{1,...m\} \\ k &\in \{1,...7\} \end{aligned} \tag{1}$$

Here, k identifies the particular joint on the limb, and α_j serves as an index to the end effector orientation, i.e., $\alpha_1 = \psi_{ee}$, $\alpha_2 = \theta_{ee}$, and $\alpha_3 = \phi_{ee}$. For this work, our goal is to reduce the required joint velocities for motions of the end effector. Toward achieving this, *our cost is simply the sum of the squares of the differences in joint angle values between neighbors in the mesh.* For example, consider a 1D toy example with a single joint angle, q, illustrated in Figure 7.

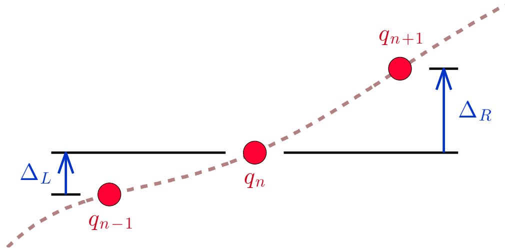

FIGURE 7. Example joint angles, q_i, along one direction in a grid of end effector positions. Here, $\Delta_L = q_n - q_{n-1}$ (difference in joints, to the left of q_n) is smaller than $\Delta_R = q_{n+1} - q_n$. Correspondingly, the algorithm seeks to increase the magnitude of q_n, toward minimizing changes in joint velocities, and it does so by looking at the local gradient effects (i.e., partial derivatives) of differential changes in end effector yaw [and pitch and roll] upon the resulting value of q_n.

The cost at mesh node n and its partial derivative with respect to q_n are, respectively:

$$\begin{aligned} c_n &= \frac{1}{2}\left[(q_n - q_{n-1})^2 + (q_{n+1} - q_n)^2\right] \\ \mu_n &\equiv \frac{\partial c_n}{\partial q_n} = (q_n - q_{n-1}) - (q_{n+1} - q_n) \end{aligned} \tag{2}$$

More generally, $c_{i,j,k}$ is defined as the sum over each of up to six adjacent neighbors in the mesh. Because the algorithm seeks to modify node assignments of roll, pitch, and yaw (α_j values) rather than modifying joint angles ($q_{i,k}$ values) directly, we also calculate partial derivatives describing the incremental change in each $q_{i,k}$ as a function of an incremental change in $\alpha_{i,j}$.

$$\beta_{i,j,k} = \frac{\partial q_{i,k}}{\partial \alpha_{i,j}} \approx \frac{q_{i,k}(\alpha_{i,j} + \delta) - q_{i,k}(\alpha_{i,j})}{\delta} \tag{3}$$

Then, we generate each of $7N$ desired weights (i.e., for each of 7 q_i values of a limb), $w_{i,j,k}$, and we sum up the 7 weights for each mesh element appropriately to create an N-element vector, $v_{i,j}$, which determines the "direction" to update the gradient search, analogous to the 2D gradient search illustrated in Figure 6. That is, the i_{th} node in the mesh updates its j^{th} orientation value, $\alpha_{i,j}$ by an amount proportional to $v_{i,j}$. Equation 4 shows these final steps in the gradient calculation. Note that $\alpha^+_{i,k}$ indicates the orientation angles to be used in the next computational loop, and κ is an iteration step size chosen by the user.

$$\begin{aligned} \mu_{i,j,k} &= \frac{\partial c_{i,j,k}}{\partial q_{i,j,k}} \\ w_{i,j,k} &= \beta_{i,j,k}\mu_{i,j,k} \\ v_{i,k} &= \sum_{j=1}^{7} w_{i,j,k} \\ \alpha^+_{i,k} &= \alpha_{i,k} + \kappa v_{i,k} \\ \kappa &\ll 1 \end{aligned} \tag{4}$$

5 ALGORITHM IMPLEMENTATION AND RESULTS

This section presents specific problem statements and results for four different implementations of our IK table algorithm, going from simple to more complex formulations.

In 5.1, we begin with generation of a look-up table for only end effector yaw, ψ_{ee}, along a 1D set of coordinates that vary only in x_{ee}. We refer to this as a "1-to-1" mapping function (from x_{ee} to ψ_{ee}). This example illustrates key aspects of the general algorithm most clearly, since we can plot end effector yaw and all seven joint angles easily as a function of only x_{ee}.

In 5.2, we next present solutions for all three orientation angles using the same 1D set of x_{ee} coordinates as in Sec. 5.1. As this generates three individual look-up tables (one each for roll, pitch and yaw) as a function of one independent variable, x_{ee}, we refer to this as a 1-to-3 mapping problem. Although each step of the algorithm now requires approximately three times the number of computations as in the 1-to-1 problem, we were surprised to discover that the number of algorithm steps required for convergence drops considerably (from 1500 to 375 steps), so that the overall computation time is over four times faster in the 1-to-3 case. Both of these cases use a set of 33 equally-spaced points, with $0 \leq x_{ee} \leq 0.8$ [m], $y_{ee} = 0.5$ [m], and $z_{ee} = -0.7$ [m].

Finally, we present initial results on a 3D mesh, solving for either yaw alone, referred to as the 3-to-1 case, or for the full orientation, known as the 3-to-3 case, in Sections 5.3 and 5.4, respectively. We use a small mesh with 60 points toward illustrating feasibility of the algorithm in the 3D case. As in the 1D mesh, we are curious about the extent to which including pitch and roll affects the solution. We also briefly compare the final 3-to-3 table solution to our original heuristic IK solution developed for rough terrain walking within Sec. 5.4.

5.1 1-to-1 Case: x_{ee} yields ψ_{ee}.

Here, we mesh along a line in the x direction in Cartesian space, and we allow only the yaw of the end effector to vary, producing the 1-to-1 function $\psi_{ee}(x_{ee})$. Recall that our cost function in this paper penalizes required joint velocity to transition between neighboring nodes, toward improving achievable end effector velocities across the reachable workspace. Because the 1D mesh also corresponds to one particular (1D) path an end effector could execute, it is natural to quantify the performance of the IK table solution by the time required to traverse this path. For this analysis, we assume a normalized joint velocity limit of $\dot{q}_{max} = 1$ rad/sec for each joint, and we approximate the time to traverse from one end of the mesh the other by finding the largest required change in joint angle for each pair of neighbor nodes, dividing this magnitude by $\dot{q}_{max}$ to estimate time of transition, and summing all node-to-node times to get a time-to-traverse (TTT) estimate for the entire path.

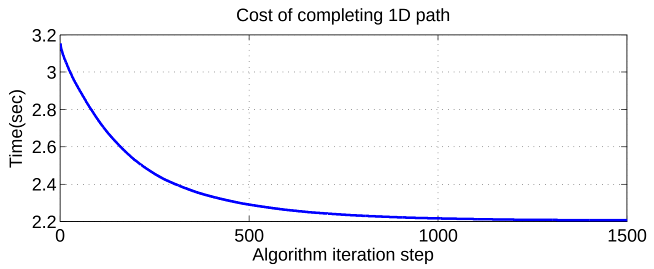

FIGURE 8. Convergence of cost function over time. Here, cost is equivalent to the time to complete the full 1D trajectory in x_{ee} (end effector motion), assuming a (normalized) maximum joint velocity of 1 (rad/s) for each joint.

Figure 8 shows the TTT decreasing smoothly and monotonically as the algorithm progresses step-by-step, converging from an initial total "cost" of 3.15 seconds (TTT) to 2.2 in after about 1500 algorithm steps. All algorithmic results shown in figures

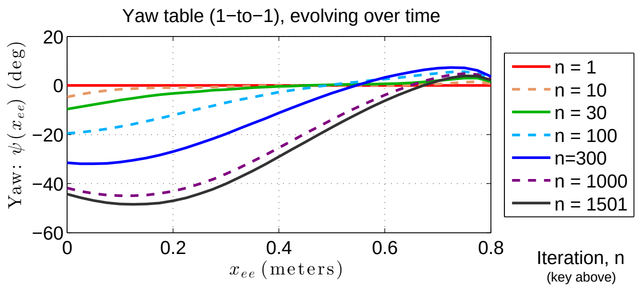

FIGURE 9. Resulting table values for optimal end effector yaw as a function of x_{ee}, $\psi(x_{ee})$, for the 1-to-1 case (i.e., meshing over x_{ee}, and allowing only yaw to vary).

are for the IK-8 family of solutions. We note briefly that performing the same test with IK-3 instead of IK-8 resulted in a final TTT of 2.5 seconds (14% slower). Figure 9 shows $\psi_{ee}(x_{ee})$ at various stages in the algorithm. Note how the changes in yaw develop with a wavelike propagation over time.

Figures 10 shows the joint angles for both the initialized mesh, with $\psi = 0$ throughout the mesh, and for the optimized solution. Note in particular that the total change in q_3 is significantly smaller in the optimized solution, accounting in large part for the overall increase in achievable trajectory speed.

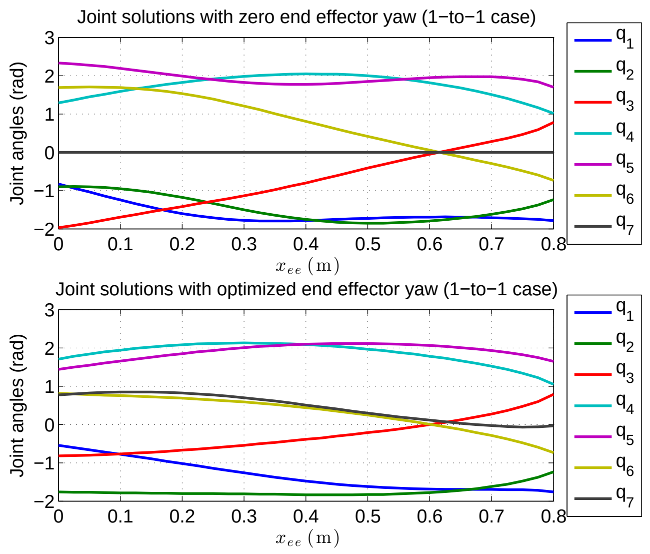

FIGURE 10. Required joint angles, q_i, as a function of end effector, x_{ee} shown before (top) and after (top) optimization by the algorithm.

5.2 1-to-3 Case: x_{ee} yields (ϕ_{ee}, θ_{ee}, ψ_{ee}).

For the 1-to-3 case, we now allow for pitch and roll of end effector along the same 1D trajectory in x. If our algorithm is working correctly, the TTT for our given path should now be at least as good as the one in which roll and pitch are constrained, but we wish to quantify the value of the additional algorithmic complexity and to verify the algorithm still works as dimensionality grows. Specific questions of particular interest are (1) how much do the additional DOFs improve trajectory speed, and (2) how much longer does the algorithm take to run. We note here that the algorithm time is not so essential for our particular application, since we will be pre-computing tables ahead of time; however, it may be of interest academically, toward future application to related, real-time trajectory planning problems.

Figure 11 shows the cost function (TTT) for the same 1D path as in Fig. 8. Since the mesh is still initialized with zero yaw at all nodes, the inital TTT is again 3.15 sec. However, the solution at convergence is now 0.61 sec, which is now significantly faster than in the 1-to-1 solution (i.e., TTT=2.2 sec), and the overall computation time to convergence is also improved, as previously mentioned. With a 3.6x improvement in end effector speed and a 4x improvement in algorithm speed, including additional DOFs in the algorithm is clearly beneficial for the 1D case. This is of particular interest since our original, heuristic IK tables exploited very little variation in pitch and roll. This result instills hope that an algorithmic approach can also significantly improve a variety of other performance more effectively than hand-tuned parameterization can.

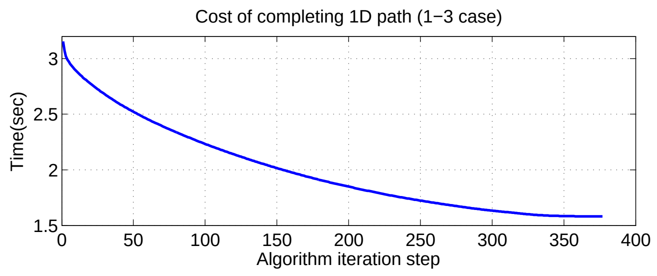

FIGURE 11. Convergence of cost function over time for the 1-to-3 case. Of note, the algorithm is significantly faster in converging once roll, pitch, and yaw are all allowed to vary, as compared with the results allowing only yaw to vary, shown in Fig. 8.

Figure 12 gives plots for all three orientation angles as function of x_{ee}, while Figure 13 shows the seven joint angles along the 1D path. Note the figure captions highlight important details about each data set.

5.3 3-to-1 Case: (x_{ee}, y_{ee}, z_{ee}) yields ψ_{ee}.

To generate a practical, real-world table, we need to ensure the algorithm also works across a 3D space. For the 3D mesh, it is not likely that the solution for the same 1D trajectory previously explored would be as fast as for the 1D constrained case, because we have also optimized the table to perform well across the other two Cartesian coordinate directions.

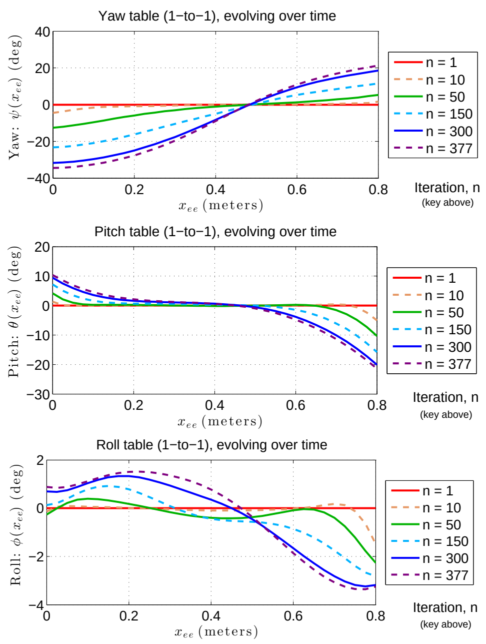

FIGURE 12. Resulting table values for optimal end effector yaw, pitch, and roll as functions of x_{ee}, for the 1-to-3 case (i.e., meshing over x_{ee}, and yaw, pitch and roll to all vary at each iteration step). Compared with Fig. 9, the optimal yaw is similar in trend but importantly distinct in shape, now that roll and pitch also contribute to the entire solution.

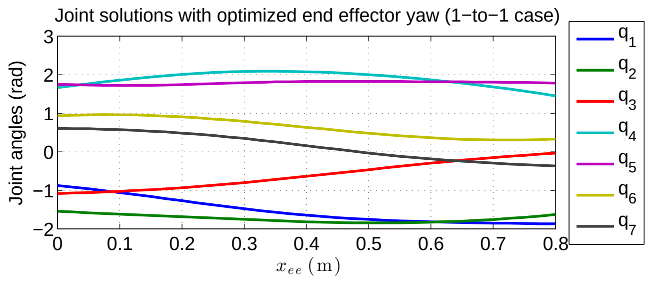

FIGURE 13. Required joint angles, $q_i(x_{ee})$, as a function of end effector for the 1-to-3 case. Compared with joint solutions for the 1-to-1 case, in Fig. 10, both q_3 and q_6 have lower magnitude slopes for $x_{ee} > 0.6$ [m] (to the right in the plot), contributing significantly to the faster overall trajectory speed possible when pitch and roll are allowed.

In this work, we use a rather coarse mesh in (x_{ee}, y_{ee}, z_{ee}). Across a 3D grid, the algorithm is simultaneously trying to reduce local jumps in joint velocity not only for motions in the x direction but also for motions in y or z. Thus, there is no longer a guarantee that any one, particular straight-line path through the 3-D space will be traversed as quickly as a dedicated 1-D solution, although intuitively, we should expect reasonable performance over any arbitrarily-chosen trajectory.

To benchmark algorithmic performance on a 3D mesh, we consider a test path along straight line in x_{ee}, analogous to the straight-line paths tested in benchmarking the convergence characteristics in Figures 8 and 11. For a path traveling from $x_{ee} = 0.4$ [m] to $x_{ee} = 0.7$ [m], the time-to-traverse drops from an initial value of 1.06 seconds to a final time of 0.562 seconds.

Figure 14 shows smoothly-varying yaw value solutions across a 2D slice of the full 3D mesh. For this 3D problem, the algorithm takes only about 122 steps to converge, compared with 1500 steps for the yaw-only solution in a 1D problem with only 33 mesh points. These results are encouraging toward the use of the algorithm in 3D problems.

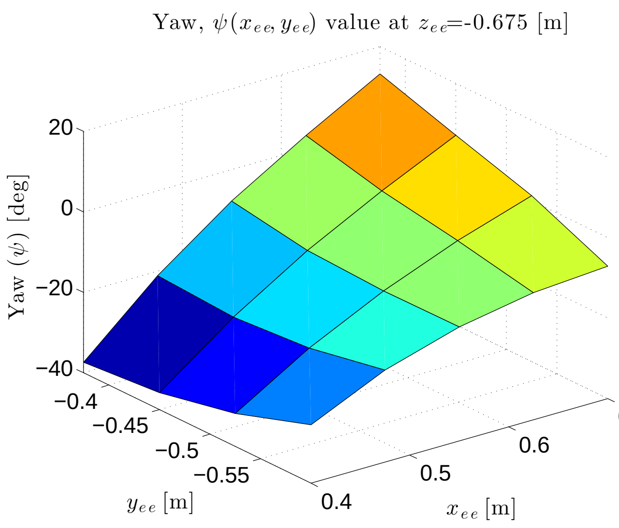

FIGURE 14. Resulting table values for optimal end effector yaw for the 3-to-1 case, taken at one slice within the full mesh.

Investigating the importance of the mesh resolution is an important issue we plan to address in future work, and we provide some brief comments on this in Section 6. The mesh we examine here contains only 60 total points.

5.4 3-to-3 Case: (x_{ee}, y_{ee}, z_{ee}) yields (ϕ_{ee}, θ_{ee}, ψ_{ee}).

Finally, we allow roll, pitch, and yaw to all vary throughout 3D space. This is the general table solution we originally set out to find, although we only demonstrate results for a coarse mesh here. As in the 1-to-3 case, we are interested at quantifying the extent to which allowing the stance legs to vary away from an exactly vertical configuration improves the achievable speed of typical end effector trajectories.

Figure 15 shows yaw, pitch, and roll solutions across the same 2D slice shown for Figure 14. For the 3-to-3 solution, the yaw is qualitatively similar as in the 3-to-1, pitch varies smoothly across the grid, and roll has very little variability.

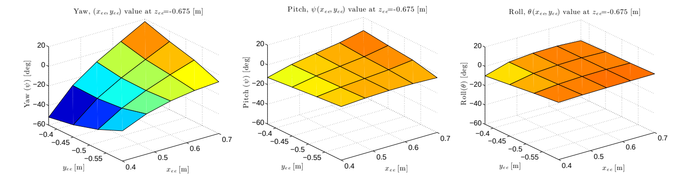

FIGURE 15. Resulting table values for optimal end effector yaw (left), pitch (center), and roll (right) for the 3-to-3 case, taken at one slice within the full mesh.

6 DISCUSSIONS AND FUTURE WORK

Having derived algorithmic solutions for 3D IK tables, it is of interest to compare the resulting table values with those from the hand-crafted, heuristic 3D table. Figure 16 shows yaw across the same 2D slice earlier documented. Qualitatively, the yaw variation is similar to that seen in Figures 14 and 15, even though solutions were derived for different families of IK solution, as previously mentioned. Intuitively this yaw similarity corresponds to the same "unfurling" behavior of the limb as it extends toward the edge of the workspace. Note that the pitch and roll across this 2D region are almost exactly zero for the heuristic case. Also, the heuristic solution in Fig. 16 is visibly less smooth than in the optimized case, as one might expect.

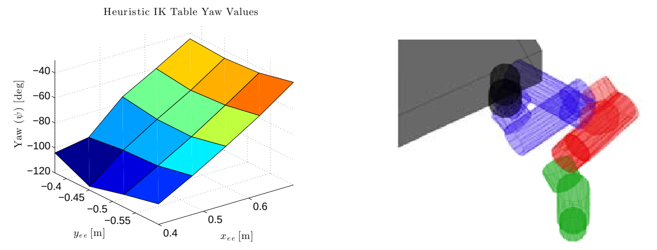

FIGURE 16. At left: Yaw values from the heuristic IK table (using IK-3 solutions), for the 2D section shown in both Figures 14 and 15 (which use IK-8). At right: IK-3 and IK-8 solutions are overlaid, to illustrate similarities in the lowermost limb segments.

Practical implementation of this algorithm on a 3D mesh is computationally expensive because the number of mesh points goes as n^D, where D is the dimensionality of the workspace for which the table is defined, and also (to a lesser extent) because we have to estimate gradients in each of D directions at each step in the algorithm. Encouragingly, however, the algorithm converges rapidly, in only about 140 algorithm steps. Reexamining Figures 9 and 12, we anticipate the number of algorithm steps is largely dependent upon the maximum number of *mesh elements in any one direction* within the mesh, since information must propagate spatially throughout the mesh (in the wavelike manner).

Creating a higher-resolution 3D mesh is an important future goal, and we note that the final IK tables we need must map to joint values, rather than to roll, pitch and yaw, since calculation of an IK solution currently depends on in-house software and is computationally expensive. Using a mesh as coarse as ours is not likely sufficient for interpolating to determine joint angles accurately. We anticipate an IK table for joint angles can best be implemented via a two-stage process. At the first stage, solutions for roll, pitch and yaw are determined using a coarse mesh. Next, values for $(\phi_{ee}, \theta_{ee}, \psi_{ee})$ would be interpolated to fill in a finer 3D mesh. This higher-resolution grid spacing would also be used to determine actual joint angles, q_i, for each joint. The joint solution at each mesh point is thereby calculated to result in the required $(x_{ee}, y_{ee}, z_{[ee]})$ value.

In future work, we plan to address several other open issues, as well, toward refining our automated IK table algorithm. For example, we might prefer optimizing for trajectories mostly in x at the expensive of motions in y and z, toward speeding up typical swing leg trajectories for walking. We might also define regions within the workspace at which we are likely to lift a leg (requiring fast motions in the z direction) versus moving forward (in x). Also, we might not care much about end effector speed in rarely-used regions of the reachable workspace, compared with frequently traversed zones. To achieve these goals, we anticipate modifying our cost function with directionally- and spatially-dependent weightings at each node.

Another goal is to allow the reachable workspace of the IK table to grow or shrink adaptively, as the algorithm runs. This would be essential in many cases, where we cannot know a priori what size workspace is reasonable to aim for. Also, we did not put hard limits on roll or pitch in this work, which would be essential for a more general algorithm, to ensure the stance legs are within some tolerance of being vertical which reduces the likelyhood of terrain contact along other parts of the limb.

We note this algorithm can also be adapted to develop tables that treat other limb parts (e.g., elbow, forearm, or omnidirectional wheel) as the "end effector" to contact terrain, or to develop tables for determining good hand-contact orientation for ladder climbing, brachiation, or locomotion with hand support against a wall. Finally, we would like to include more complicated elements into our total cost function, for example to penalize a "bow-legged" stance that might be more prone to terrain collisions or to reward solutions that place the center of mass of the limb close to some nominal location in the body coordinate system, toward simplifying high-level gait planning.

A final issue is whether the overall squared cost function should penalize all joint angle deviations for any particular node, looking in any particular direction in the Cartesian mesh. After all, once a trajectory is planned, it is only the joint with the largest relative velocity that determines the achievable speed of the end effector. More refined cost functions may improve overall performance.

7 CONCLUSION

In this paper, we focus on the open problem of how to optimize among multiple solutions for leg kinematics for a robot with four or more legs in which there are redundant solutions to achieve a particular point of contact between the foot and terrain. We present a heuristic solution to the problem for a particular robot, RoboSimian, given a goal of walking on rough terrain. The key goals of such a solution are to reach a large set of possible end effector locations while both guaranteeing all paths through the workspace result in smooth joint trajectories and reducing the required joint velocity for typical trajectoriess of the end effector. We then formalize these goals into a gradient-based search algorithm that finds a locally optimal solution to minimize a cost function based on our heuristic goals. Our results show that the algorithm converges smoothly and provides an efficient potential avenue for developing a series of IK tables, each specifically designed to optimized particular goals.

ACKNOWLEDGMENT

This work is supported in part by the DARPA Robotics Challenge. We wish to thank the entire RoboSimian team at JPL, Caltech, Stanford and UCSB for their collaborative efforts.

REFERENCES

[1] Spong, M. W., Hutchinson, S., and Vidyasagar, M., 2006. *Robot modeling and control*. New York: John Wiley Sons.

[2] Grochow, K., and Martin, S., 2004. "Style-based inverse kinematics". *ACM Transactions on graphics,* ***23***(3), pp. 522–531.

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

[4] Byl, K., Shkolnik, A., Prentice, S., Roy, N., and Tedrake, R., 2009. "Reliable dynamic motions for a stiff quadruped". In Proc. International Symposium on Experimental Robotics (ISER) 2008, Vol. 54, pp. 319–328.

[5] Kolter, J. Z., and 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), pp. 150–174.

[6] Tarokh, M., and Lee, M., 2008. "Kinematics modeling of multi-legged robots walking on rough terrain". *2008 Second International Conference on Future Generation Communication and Networking Symposia*, pp. 12–16.

[7] Raibert, M., Blankespoor, K., Nelson, G., and Playter, R., 2008. "Bigdog , the rough-terrain quadruped robot". *Proceedings of the 17th World Congress*, pp. 6–9.

[8] Burdick, J. W., 1989. "On the inverse kinematics of redundant manipulators: Characterization of the self-motion manifolds". In *Advanced Robotics: 1989*. Springer, pp. 25–34.

[9] Chiacchio, P., Chiaverini, S., Sciavicco, L., and Siciliano, B., 1991. "Closed-loop inverse kinematics schemes for constrained redundant manipulators with task space augmentation and task priority strategy". *The International Journal of Robotics Research,* ***10***(4), pp. 410–425.

[10] Donald, B., Xavier, P., Canny, J., and Reif, J., 1993. "Kinodynamic motion planning". *Journal of the ACM,* ***40***(5), pp. 1048–1066.

[11] Walker, I. D., 2008. "Kinematically redundant manipulators". In *Springer Handbook of Robotics*. Springer, pp. 245–268.

[12] Antonelli, G., 2009. "Stability analysis for prioritized closed-loop inverse kinematic algorithms for redundant robotic systems". *Robotics, IEEE Transactions on,* ***25***(5), pp. 985–994.

[13] Roennau, A., Kerscher, T., and Dillmann, R., 2010. "Design and kinematics of a biologically-inspired leg for a six-legged walking machine". In Biomedical Robotics and Biomechatronics (BioRob), 2010 3rd IEEE RAS and EMBS International Conference on, IEEE, pp. 626–631.

[14] Son, D., , Jeon, D., Nam, W. C., Chang, Doyoung andn Seo, T., and Kim, J., 2010. "Gait planning based on kinematics for a quadruped gecko model with redundancy". *Robotics and Autonomous Systems,* ***58***(5), pp. 648–656.

[15] Bertram, D., Kuffner, J., Dillmann, R., and Asfour, T., 2006. "An integrated approach to inverse kinematics and path planning for redundant manipulators". In Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., pp. 1874–1879.

[16] Colome, A., and Torras, C., 2012. "Redundant inverse kinematics: Experimental comparative review and two enhancements". In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5333–5340.

[17] Satzinger, B., Lau, C., Byl, M., and Byl, K., 2014. "Experimental results for dexterous quadruped locomotion planning with RoboSimian". In Proc. International Symposium on Experimental Robotics (ISER).

[18] Satzinger, B. W., Reid, J. I., Bajracharya, M., Hebert, P., and Byl, K., 2014. "More solutions means more problems: Resolving kinematic redundancy in robot locomotion on complex terrain". In Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).

[19] Byl, K., Strizic, T., and Ha, P., 2014. "Feasibility and optimization of fast quadruped walking with one- versus two-at-a-time swing leg motions for RoboSimian". *Submitted.*