Fast Interpolation and Time-Optimization with Contact

Kris Hauser

February 14, 2014

Abstract:

This paper presents a method for generating dynamically feasible, keyframe-interpolating motions for robots undergoing contact, such as in legged locomotion and manipulation. The first stage generates a twice-differentiable interpolating path that obeys kinematic contact constraints up to a user-specified tolerance. The second stage optimizes speeds along the path to minimize time while satisfying dynamic constraints. The method supports velocity, acceleration, and torque constraints, and polyhedral contact friction constraints at an arbitrary number of contact points. The method is numerically stable, and empirical running time is weakly linear in the number of degrees of freedom and polynomial in the time-domain grid resolution. Experiments demonstrate that full-body motions for robots with 100 degrees of freedom and dozens of contact points are calculated in seconds.

1 Introduction

Online trajectory optimization is the problem of calculating optimal motions in interactive time in reaction to changing environmental conditions, and it remains an ongoing challenge in robotics. Fast updates, on the order of seconds or even fractions of a second, are necessary to produce feasible, safe, and efficient behavior in response to sensor updates, unpredictable obstacles, and commands from human operators. Although progress has been made in the classical setting of free-space motions using numerical optimization [9, 31] and sampling-based planning techniques [14], fast optimization remains elusive for high dimensional robots and systems with contact. Contact, either with objects in manipulation settings or against the environment in legged locomotion, is challenging because it imposes both *kinematic* constraints that limit movement to a nonlinear submanifold[1]of

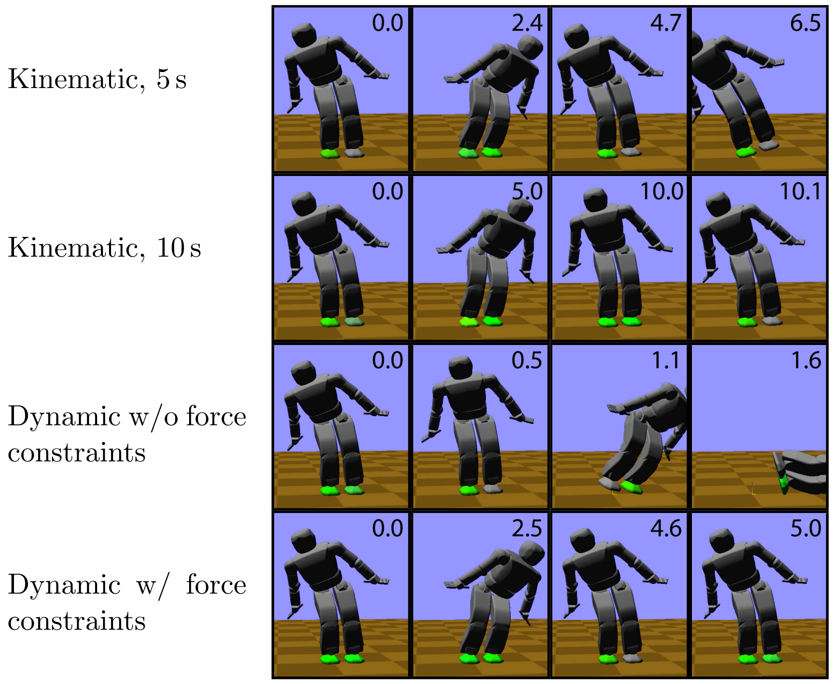


Figure 1: A side-to-side swaying motion on a humanoid robot. Pure kinematic interpolation abruptly starts and stops. Row 1: executed with uniform rate at 5 s duration, the robot falls over. Row 2: executed with a uniform rate at 10 s duration, the robot barely stays upright, wobbling onto its right foot at the end of the motion (last frame). Row 3: with dynamic interpolation under only acceleration constraints, the robot decelerates far too quickly and tips over halfway through the motion. Row 4: including dynamic contact constraints, the robot stays upright.

the robot's configuration space, as well as *dynamic* constraints that limit the speeds and accelerations that are attainable without breaking contact. Existing approaches to optimizing locomotion and manipulation trajectories [4,7,10,23,25] use descent-based numerical approaches that are typically very slow, taking minutes or hours to complete, and usually require manual selection of initial trajectories to avoid local minima.

This paper presents a dynamic interpolation method that achieves interactive performance for robots with many degrees of freedom (DOF) under

[1]If the contact constraints are singular, movement may be restricted to a non-manifold subset. This paper assumes throughout that the contact constraints are nonsingular.

contact. Dynamic interpolation is a simplified optimization problem of generating a dynamically-feasible trajectory that interpolates start and goal configurations, or in general, a series of keyframes. It is useful as a subroutine for more global optimizers because it handles the dynamic aspects of the problem, and lets the global optimizer concentrate on the simpler problem of determining a (typically small) sequence of statically-feasible keyframes. Such keyframes might also be generated by sampling-based motion planners [22] or heuristic methods like inverse kinematics. Moreover, dynamic interpolation is useful in its own right in interactive motion design for robots and virtual characters.

The method uses a two-stage approach pioneered by [2] that decouples the shape of the path from the velocities along it. First, the method constructs a smooth interpolating path on the the contact submanifold between the keyframes. Second, it computes a time-scaling of the path to minimize execution time while respecting dynamic constraints. The resulting motion meets kinematic contact constraints up to a user-specified tolerance, and obeys a variety of dynamic constraints, including joint velocity, acceleration, torque, and frictional force limits.

Three technical contributions make this method fast enough for interactive use:

1. A *recursive Hermite projection* algorithm generates twice-differentiable interpolating paths up to an arbitrarily small tolerance $\epsilon > 0$ of a contact submanifold represented by a nonlinear implicit function.

2. A *convex time-scaling* formulation recasts time-scaling as a convex, linearly constrained optimization problem in a transformed space of rate parameters. This problem has a unique minimum, which is found using a numerically-stable sequential linear programming (SLP) algorithm.

3. A *dynamic feasible rate precomputation* algorithm determines a minimal set of linear constraints relating each parameter to its successor in the time-scaling problem using a polytope projection algorithm. This drastically reduces the number of time-scaling constraints to a number that is only weakly dependent on dimensionality.

The method is very scalable, and empirical time complexity shows a linear relationship with the number of degrees of freedom, with a relatively small linear coefficient.

The approach supports a wide variety of articulated systems under contact, and can be applied to multi-limbed locomotion, full-body contact, and non-prehensile manipulations like lifting and pushing. It is guaranteed to find solutions when the interpolated path is quasi-statically balanced, but can also naturally handle systems for which no quasi-static solutions exist, such as in dynamic walking. Experiments are conducted on a variety of simulated robots, including humanoids of up to 63DOF and 60 contact points in locomotion tasks. The resulting motions are generated quickly, have relatively low execution time, and maintain actuator and balance constraints (Fig. 1).

2 Background and Related Work

Interpolation on Manifolds. Smooth interpolation on manifolds is well studied in the case where the manifold is equipped with a geodesic (e.g., SO(3) [6, 27]). For many-DOF robots under arbitrary contact constraints, geodesics are difficult to derive. Other authors have considered the problem of interpolating contact-constrained robot motions for manipulation tasks in the context of planning [1, 29, 32]. Like the current work, these methods are also fast and produce paths that satisfy kinematic constraints within a given resolution, but the paths are only C^0 continuous and dynamic constraints are not considered. Higher-order polynomials like the cubic splines used in our work better match the curvature of contact submanifolds, and appear to generate paths with lower maximum error.

Online path and trajectory optimization. Classic trajectory optimization has been applied extensively to robot vehicles and manipulators operating in free-space under a number of objective functions (e.g., to minimize time, minimize effort, or maximize safety) under kinematic and dynamic constraints (e.g., collision avoidance, actuator constraints, multibody physics equations, goal conditions). However, for many-DOF manipulators in complex environments they tend not to be fast enough for online use. More recently there has been work in interactive-time methods for path optimization in complex environments (e.g., [24]) but techniques that incorporate dynamics, contact, and frictional contact constraints remain elusive.

Trajectory optimization with contact. Numerical approaches have been applied to systems with contact, primarily in legged robot locomotion [4, 7, 10, 23, 25] and animation of virtual characters [16, 18, 19]. Certain techniques are also able to flexibly determine the sequence of contacts dynamically during optimization [18,19,23]. But global nonlinear optimization in high-dimensional parameter spaces is computationally challenging due to problem size and the prevalence of many local minima, so these techniques appear best suited for generating gait cycles [4, 23, 25] or offline primitives to be reused later [7, 10].

Existing approaches differ in methods for maintaining loop closure constraints at contacts. Unlike in character animation, where limb lengths can be shrunk or stretched to maintain the appearance of contact [18,19], robots must obey loop closure strictly. Some authors parameterize the manifold of closed-loop configurations at each possible contact state [4, 7, 10], while others enforce loop closure implicitly by adding nonlinear equality constraints at collocation points along the trajectory [23]. Parameterization is manageable for a small number of contact states (e.g., left foot, right foot, and two foot support) but is challenging and tedious when hands, knees, and elbows may be involved in contact. Collocation methods are more versatile, but the accuracy of closure is proportional to the number of chosen grid points. Fine grids greatly increase computational cost and increase the prevalence of local minima in the optimization landscape. Our approach decouples the loop closure constraint from the optimization problem, obtaining arbitrarily tightness on constraints.

Optimal time-scaling. *Time-scaling* is a simplified optimization problem that considers optimizing the execution speed of a given path under dynamic constraints. It is often used in two-stage trajectory generation [2,5] in which the first stage computes a path and then the second stage optimizes its speed. The decoupled approach may fail to achieve the same quality as full trajectory optimization because there is a risk of computing a path in stage 1 that will not yield a fast time parameterization in stage 2, but in practice they yield orders of magnitude improvements in computational time while delivering satisfactory results. The classical Bobrow method integrates the time scaling variable along dynamic limits [2] both backward and forward in time to search for a continuous time parameterization. However, as identified in [15, 26], this method was found to suffer from numerical instability issues at dynamic singularities.

Recent work formulated a more robust convex optimization approach that casts time-scaling as a second-order cone program (SOCP) [30]. This paper uses a similar convex formulation that can be solved via sequential linear program (SLP), for which fast and robust implementations are widely available. More importantly, scalability to very high-DOF robots is improved by quickly pruning irrelevant constraints. As a result the method solves 100D problems about as quickly as the prior authors [30] solved a 6D one. Moreover, it is able to handle contact force constraints in the same framework.

Simplification of Contact Conditions. Static equilibrium dictates that an objects center of mass must lie over its support polygon, and polytope projection methods have been devised to compute this polygon for arbitrarily complex contact formations [3]. We derive a dynamic version of this condition in the space of rates and accelerations along a fixed path. A similar dynamic contact constraint commonly used in bipedal walking is the zero-moment point (ZMP) condition, which states that the acceleration of a robot's center of mass is directly related to its center of pressure on flat ground (the ZMP). It is used by humanoid robots to optimizing a center of mass trajectory so that the ZMP always lies in the support foot [13]. A generalized condition was considered by [28] in the analysis of a planar rectangle with two non-planar dynamic contacts. The condition was suggested for use in time-scaling. Our new method is a practical, general algorithm that is applicable to uneven terrains, walking with hand support, and simultaneous navigation and manipulation.

3 Problem Statement

In standard *kinematic interpolation*, a continuous geometric path $p(s) : [0,1] \to \mathbb{R}^n$ is computed to connect start and goal configurations q_s and q_g. To execute such a path on a robot, some processing is needed to map time t to the parameter s. We refer to the process of devising a dynamically feasible interpolating trajectory as the *dynamic interpolation* problem.

3.1 Dynamic Interpolation in Free Space

The *dynamic interpolation* problem asks to generate a trajectory $y(t) : [0,T] \to \mathbb{R}^n$ connecting start and goal configurations q_s and q_g, starting and ending at rest. (Throughout this paper we will denote geometric paths with $p(s)$ and time-parameterized trajectories with $y(t)$.) In general, one or more intermediate keyframes may also be specified, and nonzero terminal velocities $\dot{q}_s$ and $\dot{q}_g$ may be prescribed. The duration T is unknown, and in this paper we wish to minimize T. *Dynamic constraints* impose restrictions on the first and second derivatives of $y(t)$, and are typically written in the canonical form:

$$f(y(t), \dot{y}(t), \ddot{y}(t)) \leq 0. \tag{1}$$

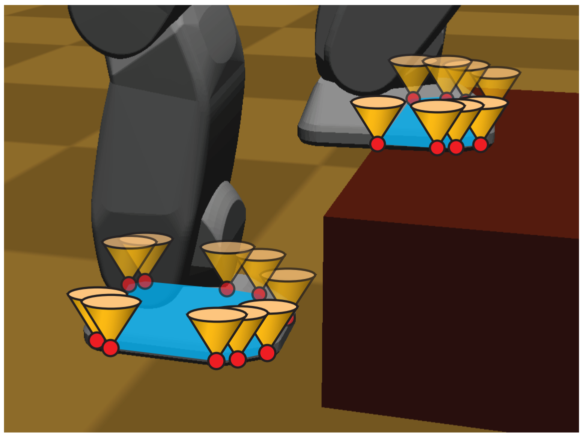


Figure 2: Our method supports arbitrary sets of frictional point contacts.

For example, joint-wise velocity and acceleration bounds are written:

$$v_L \leq \dot{y}(t) \leq v_U \text{ for all } t \in [0,T] \tag{2}$$

$$a_L \leq \ddot{y}(t) \leq a_U \text{ for all } t \in [0,T] \tag{3}$$

where all inequalities are taken element-wise.

For free-space motions (i.e. without contact), torques τ are calculated in the standard Lagrangian form

$$B(q)\ddot{q} + C(q,\dot{q}) + G(q) = \tau(q,\dot{q},\ddot{q}) \tag{4}$$

where B is the mass matrix, C is the Coriolis force vector, and G is the generalized gravity vector. Torque bounds are expressed using the equation.

$$\tau_L \leq \tau(y(t),\dot{y}(t),\ddot{y}(t)) \leq \tau_U. \tag{5}$$

3.2 Dynamic Interpolation with Contact

Contact introduces two new complications. First, it constrains $y(t)$ *kinematically* to lie on a lower dimensional subset of the configuration space, and second, it couples derivatives, torques, and contact forces *dynamically* such that differential constraints can no longer be expressed in canonical form (1). This paper models contact in a highly general manner that allows arbitrary arrangements of contact on the robot, environment, and/or rigid objects, and with polyhedral force constraints, including unilateral, frictional contact (Fig. 2).

Assumptions. For simplicity of notation, we will assume for the moment that contact must be maintained throughout the duration of the motion. Later we will describe an extension to handle making and breaking contact. Rigid objects are modeled by augmenting the robot with an additional 6 generalized coordinates for translation and rotation, which are included in q. The velocity and acceleration bounds on these coordinates are infinite, while the torque bounds are zero. A free-floating base, e.g., the torso of a humanoid robot, is modeled similarly. Note that such systems are under-actuated and hence there will be no solution of (5), in general. Instead, contact forces must be explicitly modeled.

Assume m contact points. Each robot-world contact matches a point on the robot $c(q) \in \mathbb{R}^3$ with a point in space $d \in \mathbb{R}^3$, and a robot-robot contact matches two points on the robot $c(q), d(q) \in \mathbb{R}^3$. All configurations q must satisfy $c_i(q) = d_i(q)$ for all contacts $i = 1, \ldots, m$.

Kinematic constraint. The above conditions impose a kinematic constraint that must be met at all configurations q along the trajectory:

$$\mathcal{C}(q) = 0, \tag{6}$$

with $\mathcal{C} : \mathbb{R}^n \to \mathbb{R}^k$, $k \leq n$ a nonlinear vector field that has roots when all of the m contact constraints are met. When many contacts lie on a single link, many of the constraints are linearly dependent, so we reduce the number of constraints on each individual link of the robot to at most six by determining the affine hull of the contact positions $aff(\{c_1^L, \ldots, c_k^L\})$, where c_i^L is the contact point expressed in the local frame of the link. If the hull is 2-D or 3-D we fix the link's position and orientation (a 6-D constraint), when the hull is 1-D we restrict the link to rotate about the contact axis (5-D), and when the hull is 0-D we impose a point constraint (3-D).

In general, our algorithm accepts any arbitrary smooth, nonlinear constraint such that the Jacobian of $\mathcal{C}$ is not degenerate when $\mathcal{C}(q) = 0$. If this nondegeneracy condition is met, then the set of solutions this equation is a *submanifold* of configuration space.

It is also assumed that there exists a Lipschitz constant M such that $\|\mathcal{C}(p) - \mathcal{C}(q)\| \leq M\|p - q\|$ for all $p, q \in \mathbb{R}^n$. For example, an upper bound on the distance traveled by a given contact point $c_k(q)$ on a serial robot with revolute joints is bounded by $\sum_{i=1}^{k} L_{i,k}|p_i - q_i|$ where $L_{i,k}$ is the maximum outstretched length between joint axis i and the point $c_k(q)$. We can then take $M = \sum_{i=1}^{k} L_{1,k}$.

Dynamic contact constraint. At each contact (c_i, d_i) a force f_i may be applied to the robot at each instant in time. By the principle of virtual work, the Lagrange equation becomes

$$B(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau + \sum_{i=1}^{m} J_i(q)^T f_i \tag{7}$$

where J_i is the Jacobian matrix of the constraint $c_i(q) - d_i(q) = 0$. In addition, the individual forces are required to respect friction cone constraints:

$$f_i \in FC_i(q) \text{ for } i = 1, \ldots, m \tag{8}$$

where each friction cone has an apex at the origin and axis equal to the contact normal. Note that, particularly for manipulation tasks, the normal may change with q as the object or robot link rotates. We employ the standard polyhedral friction cone approximation to represent each friction cone constraint as a set of linear inequalities. Pin joints may also be modeled by dropping the friction cone constraints altogether.

Since contact forces are a priori unknown, the instantaneous force/torque constraint becomes

$$\begin{gathered} \text{There exist } (\tau, f_1, \ldots, f_m) \text{ s.t.} \\ B(q)\ddot{q} + C(q, \dot{q}) + G(q) = \tau + \sum_{i=1}^{m} J_i(q)^T f_i \\ \tau_L \leq \tau \leq \tau_U \\ f_i \in FC_i(q) \text{ for } i = 1, \ldots, m. \end{gathered} \tag{9}$$

Note that this dynamic constraint is no longer easily represented in the canonical form (1). Checking this condition at a particular state $(q, \dot{q}, \ddot{q})$ is equivalent to testing the non-emptiness of a convex polytope of $\mathbb{R}^{n+3m}$, e.g., by solving a linear program. Unfortunately this is relatively expensive because it must be tested for *each velocity/acceleration variation* and at *each point in time.* Our approach relies on polytope projection techniques to precompute the set of velocities and accelerations that satisfy this equation along a fixed path. Perhaps surprisingly, the feasible set turns out to be a convex 2D polygon of low complexity that can be computed quickly. Although feasibility still must be checked at each point in time, testing membership in a convex polygon is extremely fast.

3.3 Summary of Approach

The approach operates in four stages:

1. *Interpolate keyframes.* Constructs a continuously differentiable kinematic path $p(s) : [0,1] \to \mathbb{R}^n$ that connects the keyframes and satisfies $\mathcal{C}(q) = 0$.

2. *Discretize time-domain.* Defines a time parameterization $s(t) : [0,T] \to [0,1]$ of p such that $y(t) = p(s(t))$ will be the resulting curve. A piecewise quadratic representation of s on N grid intervals yields an optimization variable $\theta = (\theta_0, \ldots, \theta_N)$.

3. *Precompute dynamic feasible sets.* A minimal, nonredundant set of feasibility constraints in each (θ_i, θ_{i+1}) plane, for $i = 0, \ldots, N-1$ are precomputed.

4. *Optimize time-scaling.* Optimizes θ such that $y(t) = p(s(t))$ satisfies dynamic constraints at all points t and minimizes T.

The remaining sections describe three technical contributions in more detail. First, we describe keyframe interpolation with a recursive Hermite projection technique that projects a cubic spline onto the constraint manifold. Next, we describe how to precompute dynamic feasible sets using polytope projection. Finally, we describe the formulation of time-scaling as a convex optimization problem and efficient solution techniques.

The algorithm has several favorable theoretical properties, summarized here:

- The path $p(s)$ satisfies $\|\mathcal{C}(p(s))\| < \epsilon$ for all s, for a user-supplied threshold $\epsilon > 0$.

- The algorithm correctly returns failure in finite time if any keyframes are in separate connected components of the solution set to $\mathcal{C}(q) = 0$.

- The output of Stage 3 and 4 is the globally optimal time-scaling for the parameterization of s given by θ. Furthermore, as N grows, the result approaches the globally optimal time-scaling (assuming $p(s)$ is nondegenerate).

- If $p(s)$ is quasi-statically stable — that is, (9) has a solution with $\ddot{q} = \dot{q} = 0$ — then the method will find a dynamically-feasible trajectory.

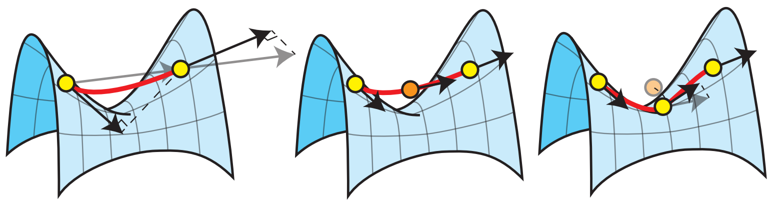


Figure 3: Interpolation begins with a smooth Hermite curve connecting the endpoints, and recursively bisects and projects the midpoint and its derivative onto the constraint manifold.

- The running time of Stage 4 is not directly dependent on the configuration space dimensionality n nor the number of contacts m, but rather on $k \cdot N$, where k is the average complexity of the nonredundant dynamic feasible sets. In practice, k is usually much less than m or n, and can be treated as essentially a constant.

Performance is chiefly governed by two parameters: the constraint violation threshold $\epsilon > 0$ and the grid size N. ϵ determines how closely the constraint manifold is followed, and as it becomes smaller, the interpolating path gets finer and Stage 1 takes longer. As N grows, the number of grid points is increased and time-scaling optimization becomes slower, but closer to the globally optimal $s(t)$. Another consideration is that the method needs a good set of input keyframes, or else it may fail to find a kinematically-feasible interpolant or a dynamically-feasible time-scaling. For example, when climbing stairs, the foot needs to be raised above the stair edge before it shifts forward and gets placed downward.

The method can be extended to handle a sequence of changes of contact by introducing keyframes at each contact transition. First, paths are independently interpolated between the sequence of keyframes at each contact state, including transition keyframes. The paths are then concatenated, and finally time-scaling is run across the entire path.

4 Interpolation via Recursive Hermite Projection

Recursive projection generates a continuously differentiable, piecewise polynomial path that satisfies $\mathcal{C}(q) = 0$ within a user-specified tolerance ϵ. It begins with an interpolating cubic Hermite curve in $\mathbb{R}^n$ and recursively bisects while projecting midpoints onto $\mathcal{C}(q) = 0$ (Fig. 3).

Hermite curves $p(s)$ are cubic polynomials controlled by endpoints x_0, x_1

and tangent velocities v_0, v_1 such that the curve satisfies $p(0) = x_0$, $p'(0) = v_0$, $p(1) = x_1$, and $p'(1) = v_1$. They can also be perfectly bisected into two Hermite curves $p^A(s)$ and $p^B(s)$ connected at the midpoint $p(0.5)$ with tangent $p'(0.5)$.Moreover, they correspond precisely to Bezier curves with knot points $P_0 = x_0$, $P_1 = x_0 + \frac{1}{3}v_0$, $P_2 = x_1 - \frac{1}{3}v_1$, and $P_3 = x_1$, and the length of the Bezier polygon $len(p) = \|P_0 - P_1\| + \|P_1 - P_2\| + \|P_2 - P_3\|$ is an upper bound on the length of the Hermite curve.

Given a threshold ϵ on constraint violation errors, the Lipschitz constant M on $\mathcal{C}$, and a growth limit $\beta \in (0.5, 1)$, recursive projection proceeds as follows:

1. Begin with a Hermite curve $p(s)$ connecting the start and end configurations. If no tangent directions are prescribed, obtain tangents by projecting the straight-line direction onto the nullspace of the Jacobians of $\mathcal{C}(q_s)$ and $\mathcal{C}(q_g)$.
2. If $len(p) \leq 2\epsilon/M$, return 'success'.
3. Otherwise, subdivide $p(s)$ into two Hermite curves p^A and p^B, meeting at the midpoint $(x, v) = (p(0.5), p'(0.5))$.
4. Project x onto $\mathcal{C}(x) = 0$ using a Newton-Raphson solver. Project v onto the nullspace of the Jacobian of $\mathcal{C}(x)$. Let the resulting configuration and velocity be x_m and v_m respectively.
5. Adjust p^A and p^B to meet at x_m with derivatives v_m.
6. If $\max(len(p^A), len(p^B)) \leq \beta \cdot len(p)$, return 'convergence failure'. Otherwise, repeat Lines 2–5 on both halves.

See Fig. 3 for an illustration of these steps. The output is a sequence of Hermite curves $p_1, \ldots, p_k$ as well as the fraction of the range $[0, 1]$ maintained by each subsegment $\Delta_1, \ldots, \Delta_k$ to map the original range into the subdivided sequence.

The algorithm terminates when $len(p)$ is small enough (Fig. 4). The growth condition in Line 6 ensures termination in finite time, and that the projected path's length is no more than a finite multiple of the length of the original curve p_0 constructed in Line 1. Notice that at recursion depth d the condition $len(p) \leq \beta^d len(p_0)$ must hold, so given the tolerance in Line 2, the terminal depth is no more than

$$d_{max}(\epsilon, \beta) = \frac{\log(2\epsilon/M \cdot len(p_0))}{\log(\beta)} + 1 \tag{10}$$

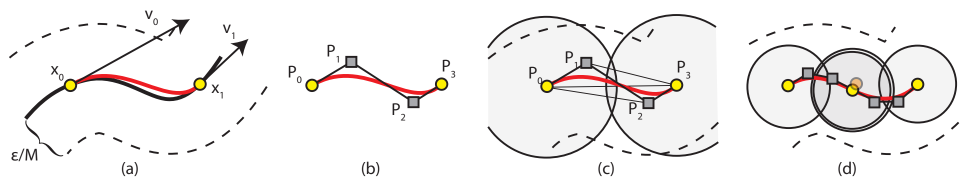


Figure 4: (a) Illustrating the recursion termination criterion, proving that the path p lies within distance ϵ/M of the submanifold in configuration space, where M is a Lipschitz constant. (b) The Bezier polygon corresponding to p. (c) p is guaranteed to lie within the union of spheres centered at the endpoints, with radius equal to half the length of the Bezier polygon. The spheres fail to satisfy the ϵ/M condition, so recursion continues. (d) After subdivision, both subsegments satisfy the condition and recursion stops.

Hence the subdivided path may be no more than $\gamma(\epsilon, \beta) = (2\beta)^{d_{max}(\epsilon,\beta)}$ times as long as p_0. All experiments use the value $\beta = 0.9$.

4.1 Properties of the Interpolator

If the algorithm is successful, the resulting spline:

- Is C^1 continuous.
- At all curve endpoints, derivatives are tangent to the constraint manifold.
- Has arc-length no greater than $\gamma(\epsilon, \beta) \cdot len(p_0)$ where p_0 is the initial curve.
- Satisfies $\|\mathcal{C}(p(s))\| \leq \epsilon$ for all $s \in [0, 1]$.

We now prove the latter claim.

Theorem 1. *The output path $p(s)$ satisfies $\mathcal{C}(p(s)) \leq \epsilon$ for all $s \in [0, 1]$.*

Proof. The path is composed of small segments $p_1, \ldots, p_k$ such that $len(p_i) \leq 2\epsilon/M$ for all i and $\mathcal{C}(p_i(0)) = \mathcal{C}(p_i(1)) = 0$. We shall prove that for any Hermite curve p, $\max_{0 \leq s \leq 1} \|\mathcal{C}(p(s))\| \leq M \cdot len(p)/2$, which implies the theorem due to the termination condition in Step 2.

By the Lipschitz condition,

$$\|\mathcal{C}(p(s))\| = \|\mathcal{C}(p(s)) - \mathcal{C}(x_0)\| \leq M \min \|p(s) - x_0\|, \|p(s) - x_1\| \qquad (11)$$

and hence the problem is one of bounding the distance between points on p and the endpoints.

Hermite curves are equivalent to cubic Bezier curves with control points $P_0 = x_0$, $P_1 = x_0 + \frac{1}{3}v_0$, $P_2 = x_1 - \frac{1}{3}v_1$, and $P_3 = x_1$ (Fig. 4), and by the convex hull property, $p(s)$ is contained entirely within the convex hull of $P_0, \ldots, P_3$. The edges of this convex hull are some subset of $\overline{P_0P_1}$, $\overline{P_0P_2}$, $\overline{P_0P_3}$, $\overline{P_1P_2}$, $\overline{P_1P_3}$, and $\overline{P_2P_3}$. The next step in the proof ensures that all of these edges are within the union of the balls B_0 and B_3 of radius $len(p)/2$ centered at each of p's endpoints.

Obviously, $\overline{P_0P_3}$ is contained within $B_0 \cup B_3$ because $len(p) \geq \|P_3 - P_0\|$. Next, since a triangle with vertices A, B, C is completely contained within balls centered at A, B with radius $\frac{1}{2}(\|A-B\| + \|B-C\|)$, the edges $\overline{P_0P_1}$, $\overline{P_1P_3}$, $\overline{P_0P_2}$, and $\overline{P_2P_3}$ are contained witihn $B_0 \cup B_3$. Finally, $\overline{P_1P_2}$ is contained within $B_0 \cup B_3$ because the midpoint of $\overline{P_1P_2}$ can be reached by walking along the Bezier polygon a distance $len(p)/2$ from either endpoint.

Since the hull is contained within a distance of $len(p)/2$ of either endpoint of p, the entirety of $p(s)$ is contained within as well, and therefore $\|\mathcal{C}(p(s))\| \leq M \cdot len(p)/2$ as desired. □

As described above, the algorithm terminates in no more than $2^{d_{max}(\epsilon,\beta)}$ steps. If the algorithm fails, it could be because 1) a midpoint becomes stuck in a local minimum of $\|\mathcal{C}(q)\|$ during projection, or 2) the recursion fails to make progress along the manifold (Fig. 5).

Correct failure can occur if there exists no interpolating path, i.e., the start and goal lie in separate connected components of the constraint submanifold. This indicates that the keyframes have been chosen poorly.

Incorrect failure can occur when the start and goal are actually in the same connected component of the constraint submanifold, but the algorithm fails to make progress. We empirically evaluated how often the algorithm fails to connect two points on a torus, with target points taken on a regular grid. Over 99% of the torus is successfully reached, and Fig. 6 shows that failures occur only in a few narrow bands. The failure rate is higher for other cases. Fig. 6 also shows a more challenging case where interpolation from one side of the manifold to the other gets stuck in local minima. One possible direction for future work might augment this procedure with surface-following techniques to increase its success rate for such problems.

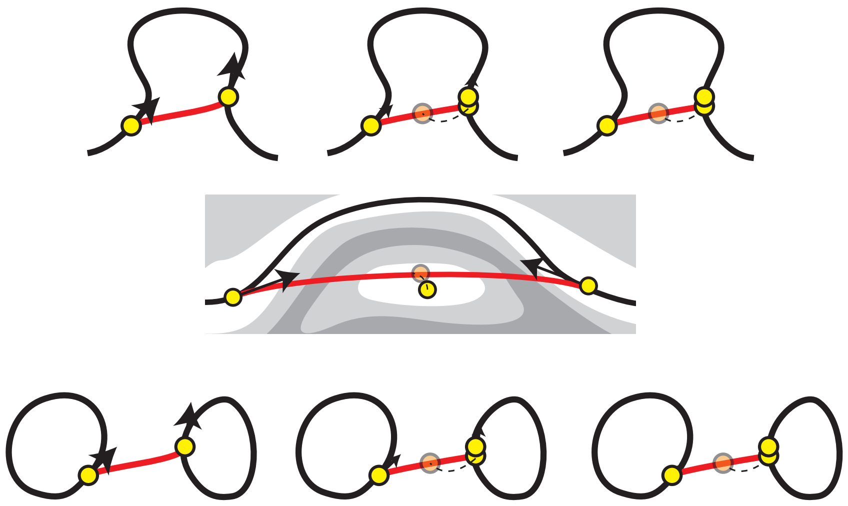


Figure 5: Recursive projection can fail when recursion fails to make progress along a manifold (Top), a condition detected in Line 6, or when a midpoint falls into a local minimum (Middle) when calling the Newton-Raphson solver in Line 4. When the manifold is disconnected, the algorithm *correctly* terminates with failure in finite time (Bottom).

4.2 Interpolating on Submanifolds of Riemannian Manifolds

The interpolation algorithm can be extended to handle submanifolds of any geodesically complete Riemannian manifold rather than $\mathbb{R}^n$. These are needed to handle the base rotations of mobile manipulators (SO(2)) and free-floating bases of legged robots and free objects (SO(3)), for which straight lines do not properly interpolate along geodesics.

Let an arbitrary Riemannian manifold $\mathcal{M}$ be equipped with a distance metric $d(a,b)$ which provides the arc-length of the length minimizing path connecting points a and b, and a geodesic function $g(u;a,b)$ which interpolates between a and b along such a length-minimizing path, with $u \in [0,1]$. The exponential map $\exp_q$ is defined such that $g(u;a,b) = \exp_a(u\dot{g}(u;a,b))$. The partial derivatives of g w.r.t. u, a, and b must also be supplied. Closed-form expressions exist for SO(2) and SO(3) [21].

Hermite interpolation on $\mathcal{M}$ is performed using the classic de Casteljau construction of a Bezier curve [6]. Let $T_x\mathcal{M}$ denote the tangent space of $\mathcal{M}$ at x. Given end points $x_0, x_1 \in \mathcal{M}$ and tangents $v_0 \in T_{x_0}\mathcal{M}$, $v_1 \in T_{x_1}\mathcal{M}$, an interpolating curve is constructed first by calculating the Bezier control

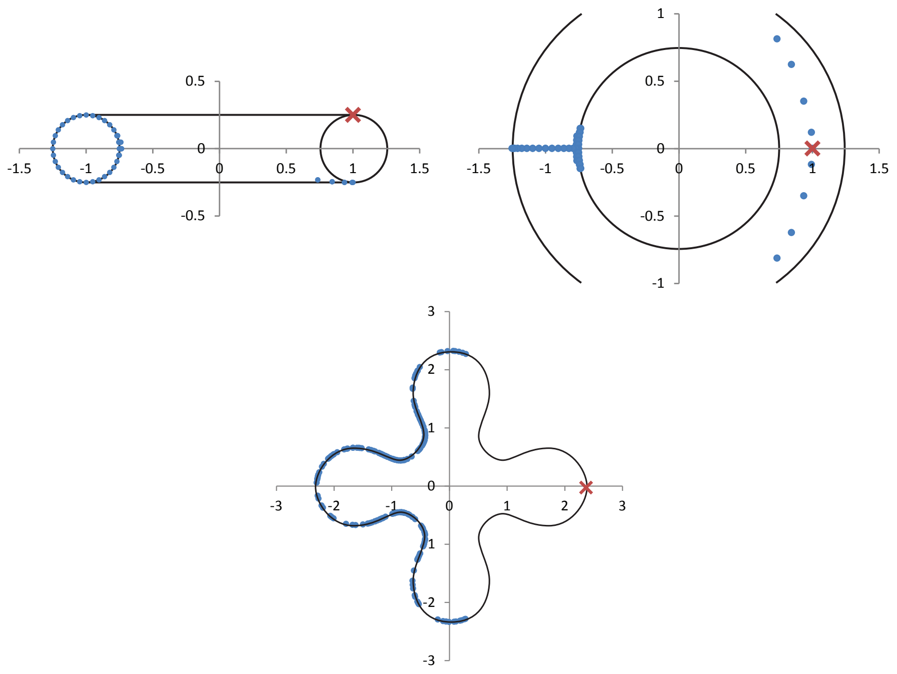


Figure 6: Top row: Interpolating on an implicit torus embedded in $\mathbb{R}^3$. Recursive projection successfully connects the source point (X) to almost all other points on the torus, except for the indicated failure points (circles). Views from the side and above. Bottom: A more challenging manifold for interpolation. About 50% of the manifold fails to be reached from the source point.

points:

$$\begin{array}{ll} P_0 = x_0 & P_1 = \exp_{x_0}\left(\frac{1}{3}v_0\right) \\ P_2 = \exp_{x_1}\left(-\frac{1}{3}v_1\right) & P_3 = x_1 \end{array} \tag{12}$$

where P_1 and P_2 are extrapolated from the endpoints along the terminal tangent vectors. Then, the path $p(s)$ is evaluated via the de Casteljau recurrences:

$$\begin{aligned} p(s) &= g(s; P_{012}(s), P_{123}(s)) \\ P_{012}(s) &= g(s; P_{01}(s), P_{12}(s)) \\ P_{123}(s) &= g(s; P_{12}(s), P_{23}(s)) \\ P_{01}(s) &= g(s; P_0, P_1) \\ P_{12}(s) &= g(s; P_1, P_2) \\ P_{23}(s) &= g(s; P_2, P_3) \end{aligned} \tag{13}$$

With this expression, the recursive bisection algorithm proceeds as usual except distances are replaced by $d(\cdot,\cdot)$ and $p'(0.5)$ in Line 3 is computed via repeated application of the chain rule:

$$\begin{aligned} p'(s) &= \left(\dot{g} + \frac{\partial g}{\partial a} P'_{012}(s) + \frac{\partial g}{\partial b} P'_{123}(s)\right)\Bigg|_{s, P_{012}(s), P_{123}(s)} \\ P'_{012}(s) &= \left(\dot{g} + \frac{\partial g}{\partial a} P'_{01}(s) + \frac{\partial g}{\partial b} P'_{12}(s)\right)\Bigg|_{s, P_{01}(s), P_{12}(s)} \\ P'_{01}(s) &= \dot{g}(s; P_0, P_1) \end{aligned} \tag{14}$$

with similar formulas holding for P'_{123}, P'_{12}, and P'_{23}. All of the examples in this paper use the above construction to generate singularity-free curves for the robots' base orientation. The 3 Euler angle parameters are converted to rotation matrices, matrices are Hermite interpolated using geodesics on SO(3), and then converted back to Euler angles. Derivatives of the Euler angle parameters are also properly converted to and from the Lie algebra so(3).

4.3 Interpolating Multiple Points

A simple extension can be used to generate a smooth interpolant through multiple points $p_0, \ldots, p_k$ on the manifold at parameters $u_0, \ldots, u_k$. The first step is to generate a sequence of derivatives $v_0, \ldots, v_k$ that are tangent to $\mathcal{C}(q) = 0$. To reduce the overall curvature of the path, a convenient technique chooses each intermediate v_i, $i = 1, \ldots, k-1$ to be tangent to a quadratic fit to p_{i-1}, p_i, and p_{i+1}. This interpolator gives

$$v_i = \frac{\Delta u_{i-1}^2 (p_{i+1} - p_i) - \Delta u_i^2 (p_{i-1} - p_i)}{\Delta u_{i-1} \Delta u_i (\Delta u_{i-1} + \Delta u_i)} \tag{15}$$

where $\Delta u_i = u_i - u_{i-1}$. v_i is then projected onto the nullspace of the Jacobian of $\mathcal{C}(q) = 0$ at p_i.

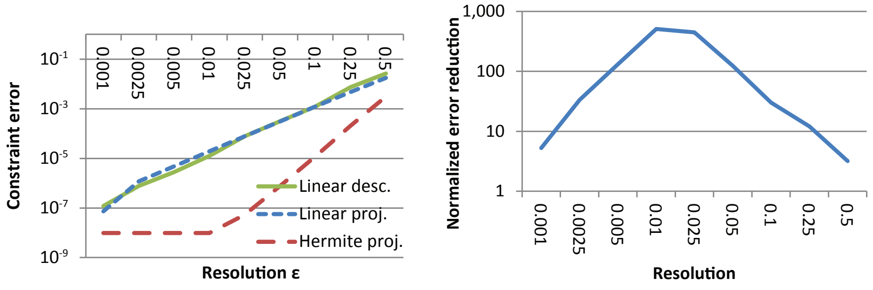


Figure 7: Left: Comparing the error of recursive Hermite projection against two piecewise linear interpolation techniques as ϵ is varied. Points are projected numerically with tolerance 10^{-8}. Right: Hermite projection still outperforms linear methods by up to several orders of magnitude when normalizing for additional overhead (see text). Values are plotted on a log scale.

At the endpoints, a quadratic fit in the ambient C-space is obtained by setting $v_0 = \frac{1}{2}(p_0 + p_1 - \Delta u_0 v_1/3)$, with a similar formula holding for v_k. Finally, the recursive projection algorithm is called to generate an interpolating curve between subsequent points p_i, p_{i+1} with derivatives $v_i \Delta u_i$ and $v_{i+1} \Delta u_i$.

4.4 Application as a Motion Planning Subroutine

Recursive projection is appropriate for use as a *local planner* in sample-based motion planners to connect keyframes, such as the approach outlined in [22]. It can also be used as a "shortcutting" technique for postprocessing jerky paths with smooth curves, similar to the B-spline path smoother presented in [20]. In such applications, a minor modification allows it to check for collisions during path construction, which allows it to discard infeasible paths quickly. This change would simply check collisions at each midpoint x_m during recursion, and along all leaf curve segments at the end. For infeasible paths, the routine is likely to exit early, rather than constructing the entire curve and then sequentially checking collision along it.

4.5 Performance Tests

We tested the performance of this method on several toy problems, including the manifold of solutions to $y = \sin(x)$, a sphere, and a torus. In each case, given the same resolution ϵ, the path produced by Hermite projection has lower maximum error than either the linear descent technique of [29] or piecewise linear projection (Fig. 7). For a fair comparison, however, it should be noted that the linear techniques are approximately 50% faster at a given ϵ; they terminate sooner because line segments are shorter than corresponding Bezier polygons, and they avoid overhead in handling tangent vectors. To normalize for overhead, the linear descent data was modeled as a linear fit of time vs. log error. Fig. 7 compares the error ratio of the linear fit vs Hermite projection for an equivalent computation time on the $y = \sin(x)$ scenario, demonstrating that the benefits of Hermite projection outweigh the added overhead. Similar results were observed in other tested scenarios.

5 Precomputing Feasible Sets of First and Second Time-Scaling Derivatives

Given the kinematic path $p(s) : [0,1] \to \mathbb{R}^n$, the goal of time-scaling is to optimize a monotonically-increasing time-scaling $s(t) : [0,T] \to [0,1]$, with an unknown trajectory duration T to be minimized. The time-parameterized trajectory is denoted $y(t) = p(s(t))$ and must satisfy (2,3,9).

This section describes a subroutine for precomputing the feasible sets of first and second derivatives of s at a point in time, which will be used in the time optimization. Specifically, we show that the set of feasible values of $(\dot{s}^2, \ddot{s})$ are convex polygons, and we present a procedure for computing the edges of polygon. In other words, given a value s, the procedure outputs a set of vectors $a(s)$, $b(s)$, and $c(s)$ such that the feasible set is precisely the solution to:

$$a(s)\dot{s}^2 + b(s)\ddot{s} \leq c(s). \tag{16}$$

5.1 Feasible Sets for Basic Dynamic Constraints

To develop some intuition for the method, we will discuss how to compute $a(s)$, $b(s)$, and $c(s)$ for free-space dynamic constraints (2,3,5). This method has slightly lower overhead than the computation for contact force constraints presented in the next section, and hence is preferable if contact forces are not time-limiting (e.g., with force closure grasps).

Rate constraints. First, as usual in time scaling, the trajectory derivatives are written in terms of the derivatives of $p(s)$ and $s(t)$ following the chain rule:

$$\dot{y}(t) = p'(s(t))\dot{s}(t) \tag{17}$$
$$\ddot{y}(t) = p''(s(t))\dot{s}(t)^2 + p'(s(t))\ddot{s}(t). \tag{18}$$

We also make use of the following relation between the time scaling derivatives and torque:

$$\tau(s, \dot{s}, \ddot{s}) = B \cdot (p''(s)\dot{s}^2 + p'(s)\ddot{s}) + C\dot{s}^2 + G \tag{19}$$

where $B(p(s))$, $C(p(s), p'(s))$, and $G(p(s))$ are independent of $\dot{s}$ and $\ddot{s}$. This equation uses the fact that the Coriolis force term satisfies $C(q, \alpha\dot{q}) = \alpha^2 C(q, \dot{q})$ for any scalar α.

Linear constraints in squared-rate / acceleration space. Substituting (17–19) into (2,3,5), we obtain:

$$v_L \leq p'(s)\dot{s} \leq v_U \tag{20}$$
$$a_L \leq p''(s)\dot{s}^2 + p'(s)\ddot{s} \leq a_U \tag{21}$$
$$\tau_L \leq B \cdot (p''\dot{s}^2 + p'(s)\ddot{s}) + C\dot{s}^2 + G \leq \tau_U \tag{22}$$

where $B(p(s))$, $C(p(s), p'(s))$, and $G(p(s))$ are independent of $\dot{s}$ and $\ddot{s}$. By inspection, it is clear that constraints (21–22) are linear in the squared-rate $\dot{s}^2$ and rate derivative $\ddot{s}$. In fact, any dynamic constraint that is linear in acceleration and parabolic in velocity:

$$f(q, \dot{q}, \ddot{q}) = c_0(q) + \dot{q}^T C_1(q)\dot{q} + c_2(q)\ddot{q} \leq 0 \tag{23}$$

can be transformed in a similar manner.

Finally, it is also a simple matter to convert (20) to squared-rate constraints:

$$0 \leq \dot{s}^2 \leq \left(\min_{i=1}^{n} \max \frac{v_{L,i}}{p'(s)_i}, \frac{v_{U,i}}{p'(s)_i} \right)^2 \tag{24}$$

where the subscript i denotes an element of each vector.

Irrelevant constraint removal via halfplane intersection. Especially for high-DOF problems, most of the $4n$ halfplane constraints (21,22) will be *irrelevant*, i.e., will not bound the feasible set F in the $(\dot{s}^2, \ddot{s})$ plane. These correspond to the dynamic constraints that are non-limiting, e.g., joints that are moving slowly relative to others. It is advantageous to remove these

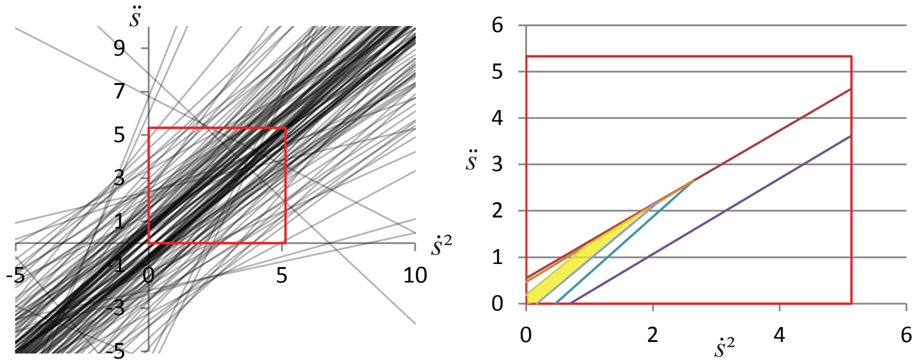


Figure 8: Of over 500 inequality constraints relating squared-rates and accelerations, all but 5 are irrelevant to the feasible set. A halfplane intersection procedure calculates the feasible polygon (shaded) by repeated cutting.

constraints because they do not affect the solution and the time-scaling optimization will be solved more quickly with fewer constraints.

To do so, we invoke a halfplane intersection routine. F is incrementally built, starting from the vertical strip satisfying (24). Then, each halfplane is added one-by-one to cut away portions of the polygon (Fig. 8). At the end of this process the supporting halfplanes are the boundaries of F. The complexity of this procedure is $O(kn)$ where k is the maximum number of edges in the intermediate polygons. We observe a low value of $k << n$ in practice. This added cost is usually overwhelmed by the savings because time-scaling runs in time superlinear in the number of constraints.

5.2 Feasible Sets for Contact Constraints

Alternative methods are needed to handle contact force constraints, which are no longer expressed in the form (1). We find that the set of $\dot{s}^2$ and $\ddot{s}$ for which *some* set of feasible torques and contact forces exist (9) is a convex polygon, which corresponds to the projection of a $2+n+3m$ dimensional convex polytope onto a plane. To calculate this polygon we use an efficient recursive expansion algorithm.

Transformation to rate/force/torque space. Performing the substitution $y(t) = p(s(t))$ into (7), we find the following instantaneous relation between time scaling derivatives, torques, and forces:

$$B \cdot (p'' \dot{s}^2 + p' \ddot{s}) + C\dot{s}^2 + G = \tau + \sum_{i=1}^{m} J_i^T f_i. \tag{25}$$

Here, p', p'', B, C, G, and each J_i depend only on s and are independent of $\dot{s}$ and $\ddot{s}$, so this equation is linear in the $(\dot{s}^2, \ddot{s}, \tau, f_1, \ldots, f_m)$ space.

Including force and torque constraints, we are interested in the feasible set F of $(\dot{s}^2, \ddot{s}, \tau, f_1, \ldots, f_m)$ that satisfy the simultaneous equations:

$$\begin{gathered} B \cdot (p'' \dot{s}^2 + p' \ddot{s}) + C\dot{s}^2 + G = \tau + \sum_{i=1}^{m} J_i^T f_i \\ \tau_L \le \tau \le \tau_U \\ f_i \in FC_i(p(s)) \text{ for } i = 1, \ldots, m. \end{gathered} \tag{26}$$

as well as velocity and acceleration constraints (20) and (21). It is slightly faster to eliminate τ as an optimization variable by moving the force terms in (25) to the left hand side and then bounding the l.h.s. by the torque limits τ_L and τ_U. This reduces the dimension of F from $2 + n + 3m$ to $2 + 3m$.

Testing whether a particular set of rates $(\dot{s}, \ddot{s})$ yields feasible forces and torques is equivalent to checking whether a $3m$-dimensional hyperplane intersects F. Another interpretation is that the value of $(\dot{s}, \ddot{s})$ must lie in the "shadow" of F when projected onto the $(\dot{s}^2, \ddot{s})$ plane. Next we show how to compute this 2-D projection, which we denote P. Specifically, with x denoting the $(\dot{s}^2, \ddot{s})$ component of this space and y denoting the rest, we have $P = \{x \quad | \quad (x, y) \in F\}$.

Polytope projection via recursive expansion. Since all constraints are convex, F is also a convex set, and P is a convex planar shape. With friction cones approximated as polyhedra, F is a convex polytope and P is a convex polygon (Fig. 9). We compute this polygon by the recursive expansion technique presented in [3] which bears a close resemblance to the Equality Set Projection algorithm of [12].

The algorithm incrementally grows tighter approximations to P by determining an extremum of F in a direction on the x plane at each iteration. Each extremizing step solves a linear program (LP) of the form

$$\max_{x,y} v^T x \text{ s.t. } (x, y) \in F \tag{27}$$

with v some direction in the plane.

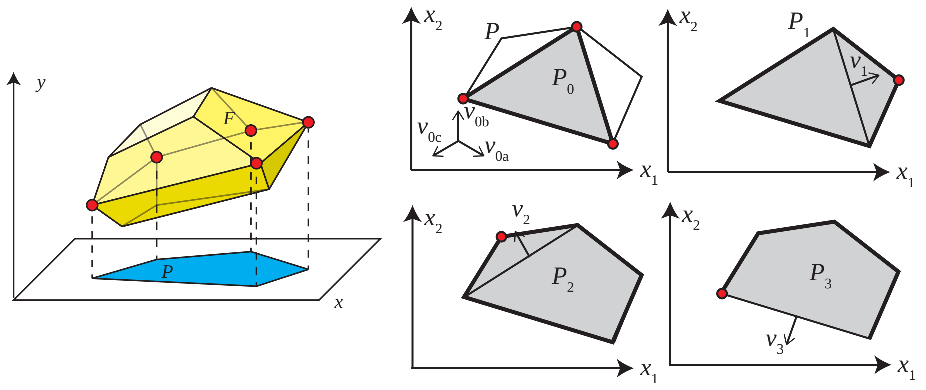


Figure 9: Illustrating the polytope projection step. The feasible polytope F is projected along the torque and force dimensions (denoted y) onto the plane of first and second derivatives the time scaling parameter (denoted x). To do so, the x parameter is maximized in three initial directions v_{0a}, v_{0b}, v_{0c} subject to the halfplane constraints of F to obtain an initial approximation P_0. Each edge is then recursively expanded in the direction of the outward normal to obtain the next approximation. When an edge fails to expand, recursion stops.

The algorithm begins at an initial polygon $P_0 \subseteq P$ by extremizing P in at least three directions whose positive span contains the origin in its interior. Our implementation uses the directions $v_{0a} = (1,0)$, $v_{0b} = (\cos(120°), \sin(120°))$, and $v_{0c} = (\cos(240°), \sin(240°)))$. If any of these LPs are infeasible, then P is empty and the dynamic constraints are inconsistent.

Next, it begins a recursion to grow finer approximations $P_1, P_2, \ldots$, at each step i extremizing along a direction v_i which is determined by choosing an outward-pointing normal to an unexpanded edge of P_i. The value of x that achieves the maximum is either contained in P_i or not. In the former case, that edge is marked as expanded in P_{i+1}. Otherwise, x is added as a new vertex to obtain P_{i+1}. This continues until no unexpanded edges remain, or a maximum number of iterations is reached.

The steps of this procedure are illustrated in Fig. 9. Once the edges of P are determined, each supporting plane is converted to a halfplane equation $a\dot{s}^2 + b\ddot{s} \leq c$.

Comments on complexity. Let n be the robot dimensionality, m be the number of contacts, and k be the number of planes in each approximated

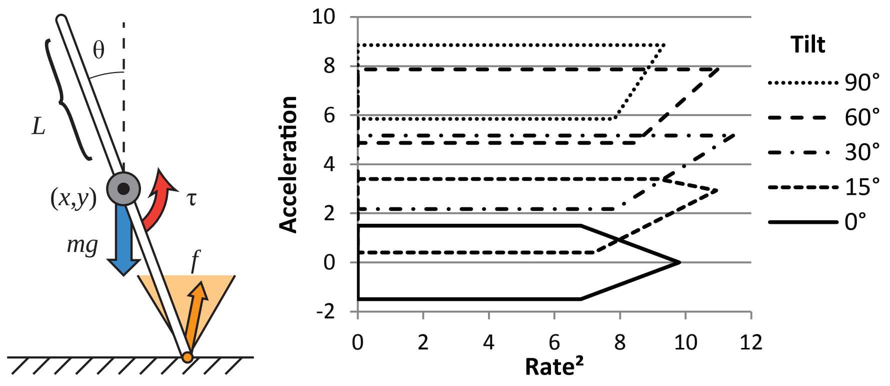


Figure 10: Simple rod example with a point contact.

friction cone. Because F lies in a $2+3m$-dimensional space and is bounded by $3n+km$ halfplane constraints, in the worst case the feasible set $\binom{3n+km}{2+3m}$ vertices. However, experiments suggest the number of boundaries z of the projected feasible sets are of much lower complexity, and can be considered essentially bounded by a constant. Surprisingly, $z \approx 7$ on average in our humanoid robot examples despite having dozens of degrees of freedom and contact points. The recursive expansion algorithm is output-sensitive, only making $O(z)$ calls to solve a relatively small linear program. So, empirical performance appears to be polynomial time.

5.3 Illustration: Rod with Point Contact

Fig. 10 illustrates a planar rod of mass m, inertia H, and length $2L$ making contact with the origin at one end. Its coordinates are given by (x, y, θ), where (x, y) is the center of mass and θ is the leftward lean angle. It can produce a torque τ_θ about the center of mass, but cannot exert translational forces on the rod directly. The contact force f is subjected to Coulomb friction with coefficient μ. The path is fully determined by θ via the equations $x = -L\sin\theta$, $y = L\cos\theta$.

The mass matrix B is a diagonal matrix with entries $[m, m, H]$, the Coriolis term C is zero, and the gravity vector G is $[0, mg, 0]^T$. Assuming a path with constant velocity $\theta'(s) = 1$ and $\theta''(s) = 0$, we have the torque τ given by:

$$\begin{bmatrix} -mx \\ -my \\ 0 \end{bmatrix} \dot{s}^2 + \begin{bmatrix} -my \\ mx \\ H \end{bmatrix} \ddot{s} + G - \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -y & x \end{bmatrix} f. \tag{28}$$

Since $\tau = [0, 0, \tau_\theta]^T$, the first two rows in this matrix must be equal to zero. In other words,

$$f = -m[x, y]^T \dot{s}^2 + m[-y, x]^T \ddot{s} + [0, mg], \tag{29}$$

and the third row gives:

$$\tau_\theta = H\ddot{s} + [-y, x]^T f. \tag{30}$$

Replacing the formula for f into the above expression, we have

$$\tau_\theta = (H + mL^2)\ddot{s} + mxg \tag{31}$$

after a bit of algebra and observing that $x^2 + y^2 = L^2$. This equation signifies that with zero torque, $\ddot{s} = -mxg/(H + mL^2)$. Adding torques increases the range of attainable accelerations.

Adding in the torque constraints $|\tau_\theta| \leq \tau_{max}$ and the friction constraints $[1, \mu]f \geq 0$ and $[-1, \mu]f \geq 0$, we observe four linear constraints in the $\dot{s}^2$ and $\ddot{s}$ space. $\dot{s}^2$ must also be nonnegative.

Fig. 10 plots the feasible set for different values of θ for $m = L = 1$, $H = m/3$, $\mu = 0.5$, and the magnitude of τ_θ limited to 2. Observe that forward and backward accelerations are only attainable in a small range of inclinations about the vertical; otherwise the torque exerted by gravity overwhelms the robot's ability to balance. Also observe that at faster rotation speeds (approximately $\dot{\theta} \geq \sqrt{7}$), the feasible set is bounded by diagonals, and at even faster speeds, there are no feasible solutions. A vertex at the upper and lower bounds signifies that at faster velocities, the contact would slide because of insufficient friction. A rightward facing point between the upper and lower bounds indicates that at faster speeds, additional *downward* forces are needed to maintain contact. Without such forces, the rod would break contact and fly away.

6 Convex Optimization Time Scaling

This section describes the final time-scaling component of the method, and proves that with the proper parameterization this gives rise to a convex program with linear inequalities.

6.1 Parameterizing the time scaling by gridding the path domain

For (18) to be well defined, y must be twice differentiable, and hence it is necessary for $\dot{s}(t)$ to be continuous. Terminal conditions dictate that the

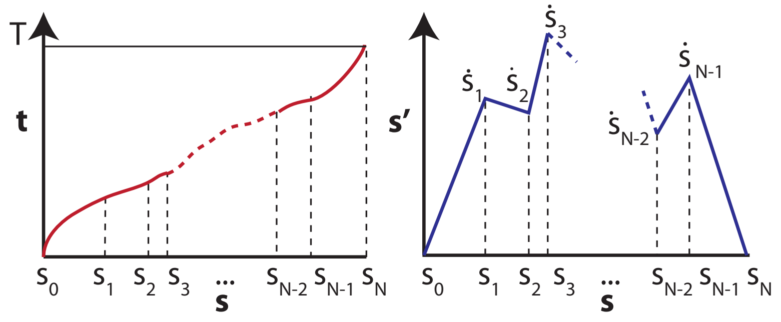


Figure 11: A piecewise quadratic time-scaling $s(t)$ is parameterized by the rates $\dot{s}_0, \ldots, \dot{s}_N$.

trajectory start and stop at zero velocity: $\dot{s}(0) = \dot{s}(T) = 0$.

We grid the path domain $[0,1]$ into N intervals $[s_k, s_{k+1}]$, $k = 0, \ldots, N-1$ and enforce constraints at the collocation points $s_0, \ldots, s_N$. We then formulate s as a piecewise quadratic, twice differentiable curve parameterized by $N+1$ *rate parameters* $\dot{s}_0, \ldots, \dot{s}_N$ at segment division points (Fig. 11). Although it is not immediately obvious, these parameters do indeed fully determine $s(t)$ and T as follows.

Consider an interval k of size $\Delta s_k = s_{k+1} - s_k$ and its unknown time interval $[t_k, t_{k+1}]$. Over this interval, $s(t)$ is fully determined by endpoint velocities $\dot{s}(t_k) = \dot{s}_k$ and $\dot{s}(t_{k+1}) = \dot{s}_{k+1}$. We require $s(t_k) = s_k$, $s(t_{k+1}) = s_{k+1}$, $\dot{s}(t_k) = \dot{s}_k$ and $\dot{s}(t_{k+1}) = \dot{s}_{k+1}$. Simple algebra verifies that the choices $\Delta t_k = t_{k+1} - t_k = \frac{2\Delta s_k}{\dot{s}_{k+1} + \dot{s}_k}$ and

$$s(t) = \frac{\dot{s}_{k+1}^2 - \dot{s}_k^2}{4\Delta s_k}(t - t_k)^2 + \dot{s}_k(t - t_k) + s_k \tag{32}$$

satisfy all of the boundary constraints.

Also, observe that $\dot{s}(t)$ is a linear interpolation between $\dot{s}_k$ and $\dot{s}_{k+1}$ and

$$\ddot{s}(t) = (\dot{s}_{k+1}^2 - \dot{s}_k^2)/(2\Delta s_k) \tag{33}$$

is constant over the entire interval. Finally, we have the total trajectory duration

$$T = \sum_{k=1}^{N} \Delta t_k = \sum_{k=1}^{N} \frac{2\Delta s_k}{\dot{s}_{k+1} + \dot{s}_k}. \tag{34}$$

6.2 Linearly Constrained Convex Program in the Squared-rate Space

The next step of the formulation transforms the optimization problem to *squared-rate* parameters $\theta_0 = \dot{s}_0^2, \ldots, \theta_N = \dot{s}_N^2$. Due to (33) we can transform each linear constraint in the form (16), evaluated at a grid point, to a linear constraint in the squared-rate space.

In particular, at a grid point s_k, the constraints of (16) become

$$a(s_k)\theta_k + b(s_k)\frac{\theta_{k+1} - \theta_k}{2\Delta s_k} \leq c(s_k) a(s_k)\theta_k + b(s_k)\frac{\theta_k - \theta_{k-1}}{2\Delta s_{k-1}} \leq c(s_k). \quad (35)$$

(Only the first constraint is used at $k = 0$ and only the second is used at $k = N$.) Note that the acceleration constraints must be duplicated to account for differing second derivatives over interval k and $k - 1$.

The final optimization problem minimizes time subject to dynamic feasibility:

$$\begin{gathered}
\min_\theta T(\theta) = \sum_{k=1}^{N} \Delta t_k = \sum_{k=1}^{N} \frac{2\Delta s_k}{\sqrt{\theta_{k+1}} + \sqrt{\theta_k}} \\
\text{s.t. } \theta_0 = \theta_N = 0 \\
a(s_k)\theta_k + b(s_k)\frac{\theta_{k+1} - \theta_k}{2\Delta s_k} \leq c(s_k) \text{ for } k = 0, \ldots, N-1 \\
a(s_k)\theta_k + b(s_k)\frac{\theta_k - \theta_{k-1}}{2\Delta s_{k-1}} \leq c(s_k) \text{ for } k = 1, \ldots, N.
\end{gathered} \quad (36)$$

We now state the main result:

Theorem 2. *The optimization problem (36) is convex.*

Proof. Each equality and inequality in (36) is linear, and hence the feasible set F is a convex polytope. Convexity of T follows because the inverse function is convex and non-increasing, and the square root function is concave. □

In the following, we make the weak assumption that $p(s)$ is nondegenerate (that is, $p'(s) \neq 0$ for all s).

Lemma 1. *The feasible set F of (36) is bounded, and the global, unique minimum of (36) lies on the boundary of F.*

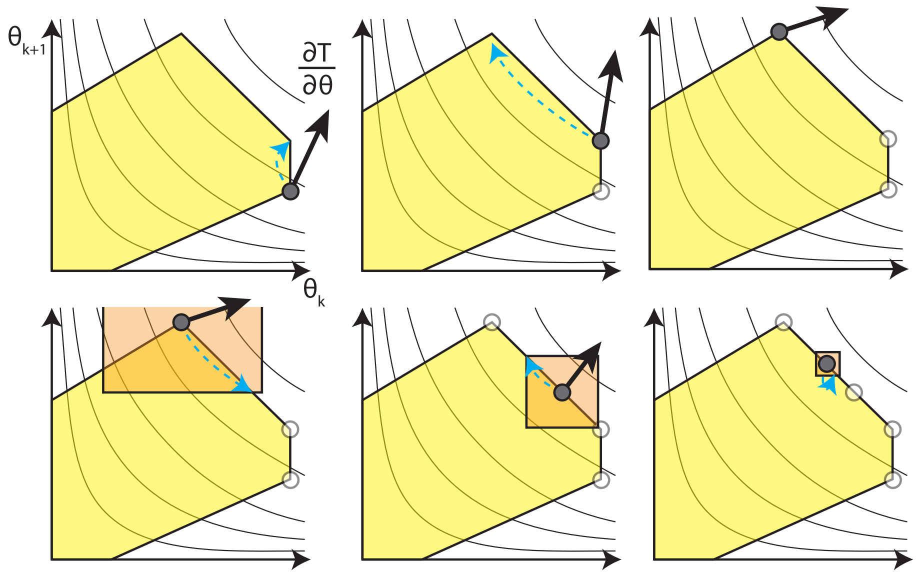


Figure 12: Illustrating sequential linear programming on a hypothetical 2D slice. The feasible set is shaded, and contours of T are drawn. First, a negated gradient direction is computed (solid arrow), and an LP is solved to obtain the next step (dashed arrow). This continues until the LP move does not decrease T. The LP trust region (shaded square) is shrunk until a decreasing move is found. The process continues until convergence.

Boundedness of F follows because (24) gives a finite upper bound to every θ_k. Furthermore, over the unbounded positive space $\theta \geq 0$, T approaches a single global minimum value (namely, zero with $\theta \to \infty$). Since F is bounded and convex, and T is convex, the optimum of the constrained problem must lie on the boundary.

These results are significant because if there exists a feasible solution θ, then it can be found reliably using convex programming techniques. Furthermore, as N increases, the representation of $s(t)$ provided by the optimized θ grows progressively more faithful to the true optimum.

We also prove that for paths in quasi-static balance, a feasible solution will be found.

Lemma 2. *If $p(s)$ is quasi-statically balanced for all $s \in [0, 1]$ (i.e. there exists a solution to* (9) *with $\dot{q} = \ddot{q} = 0$), and all velocity and acceleration bounds contain zero (i.e., $v_L \leq 0 \leq v_U$ and $a_L \leq 0 \leq a_U$), then the optimization program (36) has a feasible solution.*

Proof. The proof shows that the origin $\theta_0 = \cdots = \theta_N = 0$ is a feasible point, in other words, $\dot{s} = \ddot{s} = 0$ throughout the path.

The quasi-static balance condition states that for all points $p(s)$ on the path, there exist torques $\tau(s)$ and forces $f_i(s)$, $i = 1, \ldots, m$ that satisfy:

$$\begin{gathered} G(p(s)) = \tau(s) + \sum_{i=1}^{m} J_i(p(s))^T f_i(s) \\ \tau_L \leq \tau \leq \tau_U \\ f_i(s) \in FC_i(p(s)) \text{ for } i = 1, \ldots, m. \end{gathered} \tag{37}$$

Therefore, at every point s there exists a solution to (26) with $\dot{s} = \ddot{s} = 0$. Since $\dot{s} = \ddot{s} = 0$ also satisfies the velocity and acceleration bounds (20) and (21), the solution is feasible. □

Note that this proof does not necessarily indicate that there exists a feasible solution with finite duration T. For example, if the robot's center of mass lies precisely on the edge of the support polygon, it is balanced quasi-statically but there are no valid forces that can push the center of mass inwards. In such cases, the algorithm will return successfully but with an infinite-time solution.

6.3 Solution via Sequential Linear Programming

Linearly constrained convex programs are suitable for sequential linear programming (SLP) solvers. SLP starts at an initial point and linearizes the objective function about the point to obtain a descent direction, which is optimized by solving an LP. This process repeats, linearizing the objective about the new point (Fig. 12). A trust-region method is used to ensure that the process converges to optimal solutions that are not at a vertex of F. The implementation operates as follows:

Time Scaling SLP

1. Initialize a rough solution θ by greedily picking each subsequent θ_{k+1} according to the maximum value that satisfies all constraints involving itself and θ_k
2. Initialize the trust region size $r = \|\theta\|_\infty$.
3. Linearize the objective function about the current solution θ and solve an LP:

$$\begin{gathered} \min_x \frac{\partial T}{\partial \theta}(\theta)^T x \text{ s.t.} \\ x \in F \text{ and } \|x - \theta\|_\infty \leq r. \end{gathered} \tag{38}$$

4. If the LP is infeasible, return ‘failure’.
5. If $T(x) > T(\theta)$, set $r \leftarrow r/2$ and repeat from step 3. Otherwise set $r \leftarrow 1.5r$.
6. If $|T(x) - T(\theta)| < \epsilon_T$ or $\|x - \theta\|_\infty < \epsilon_\theta$ return ‘converged’.
7. Set $\theta \leftarrow x$ and repeat from step 3.

The algorithm terminates when the change in T or the change in θ decrease below user-specified convergence thresholds ϵ_T or ϵ_θ, respectively (Line 6). Also, given a fixed time budget, it may be terminated with θ containing a feasible solution any time after the first iteration.

Thanks to widely available and reliable LP implementations (e.g., CPLEX, GLPK), SLP is robust and fast. Moreover, it typically takes only a few iterations to converge. Lemma 1 helps explain why this is so: the optimum is often at a vertex of F, so the LP solution often reaches a maximum without ever needing to adapt the trust region size.

6.4 Conservative Enforcement of Exact Dynamic Feasibility

The collocation point method only enforces dynamic feasibility at finite points, which has the potential to miss constraint violations in-between. The severity of these violations shrinks as N grows, but it may be desirable to obtain dynamic feasibility regardless of the value of N.

The simplest approach to this problem is to add safety margins onto each constraint. This also helps the method tolerate execution and modeling errors. We slightly shrink the torque limits and friction coefficient estimates by a user-defined parameter. Also, we add a small offset to the each friction cone in the normal direction which enforces that each contact must apply a minimum force. Currently these parameters are chosen by manual tuning but in future work we would be interested in guaranteed feasibility. The humanoid robot experiments in this paper use a friction reduction of 50% and impose a minimum of 9.8 N force on each limb in contact.

A more sophisticated method uses interval analysis to conservatively bound the coefficients of the optimization parameters along each grid interval. We have implemented this technique for velocity and acceleration constraints (2,3) as follows. Extending this method to torque and force constraints is left for future work.

Over the domain $s \in [s_k, s_k + \Delta s_k]$, we compute bounds on the derivatives of p: $p'(s) \in [v_k^L, v_k^U]$ and $p''(s) \in [a_k^L, a_k^U]$. Here the notation $[a, b]$, where a

and b are vectors, indicates an axis-aligned box in $\mathbb{R}^n$. Since each Hermite curve comprising the path is a third-degree polynomial along each axis, velocity extrema are found either at the endpoints or at critical points where acceleration crosses zero. Acceleration extrema occur only the endpoints.

To bound the inner velocity term in (20), observe that $\dot{s}(t)$ is a linear interpolation with extrema $\dot{s}_k$ at $t = t_k$ and $\dot{s}_{k+1}$ at $t = t_{k+1}$. Hence, (20) is proven feasible throughout the interval if the constraints

$$\begin{aligned} v_L \leq v_k^L \dot{s}_k \qquad & v_L \leq v_k^L \dot{s}_{k+1} \\ v_k^U \dot{s}_k \leq v_U \qquad & v_k^U \dot{s}_{k+1} \leq v_U \end{aligned} \tag{39}$$

are satisfied. (The constraints become one-sided due to the requirement that $\dot{s} > 0$.)

The acceleration constraint (21) expands to a quadratic constraint on $\dot{s}_k$ and $\dot{s}_{k+1}$. Note that because the time scaling has positive derivative, the extrema of the term $\dot{s}(t)^2$ are obtained at the endpoints $\dot{s}_k^2$ and $\dot{s}_{k+1}^2$. Hence, (21) is proven feasible throughout the interval if the conditions:

$$\dot{s}_k^2 [a_k^L, a_k^U] + \frac{1}{2\Delta s_k} [v_k^L, v_k^U] (\dot{s}_{k+1}^2 - \dot{s}_k^2) \in [a_L, a_U] \tag{40}$$

$$\dot{s}_{k+1}^2 [a_k^L, a_k^U] + \frac{1}{2\Delta s_k} [v_k^L, v_k^U] (\dot{s}_{k+1}^2 - \dot{s}_k^2) \in [a_L, a_U] \tag{41}$$

are satisfied. Note that these conditions are also linear in θ_k and θ_{k+1}, and are amenable to irrelevant constraint removal.

7 Experiments

Performance testing is conducted on a 2.67 GHz PC and C++ implementation using the GLPK library to solve linear programs. The Featherstone articulated body dynamics algorithm [8] was adapted to compute the coefficients in (26).

7.1 Performance and Dimensionality

Irrelevant constraint pruning provides major speed advantages for high-DOF robots, since the number of relevant constraints is only weakly dependent on n (Fig. 13). The total number of inequality constraints grows with cNn, where c is a constant depending on how many constraints are enforced: $c = 2$ for acceleration bounds only, $c = 4$ for acceleration and torque bounds, and $c = 8$ for conservatively enforced acceleration bounds. However, the number

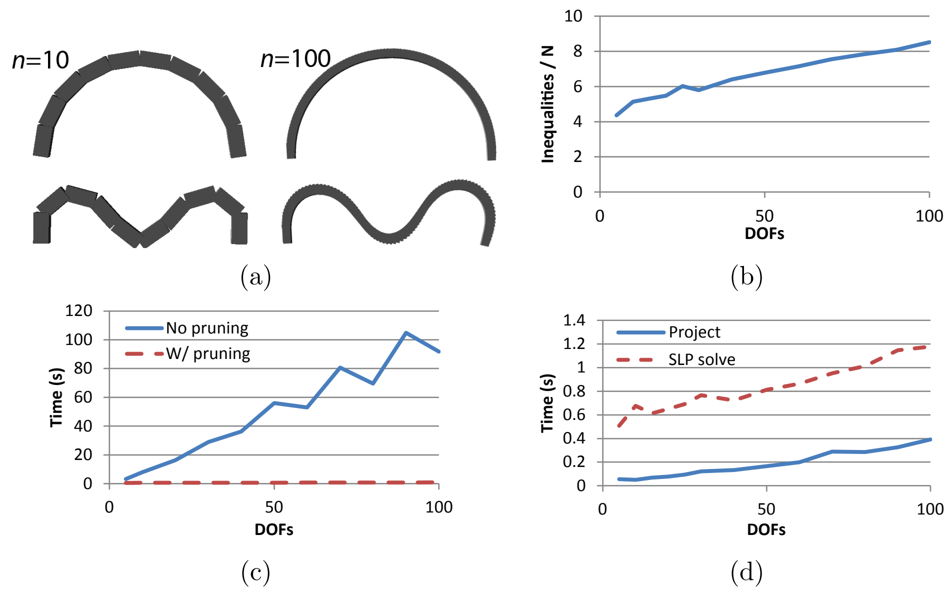


Figure 13: Performance on (a) planar, endpoint constrained, n-joint robots. (b) In the 100D case, nearly 99% of the original constraints are irrelevant. (c) When those constraints are pruned, the cost of time-scaling is only weakly dependent on n. (d) Including spline projection, our technique can solve 100D problems in approximately 1 s. (Grid size is fixed at $N = 1024$.)

of relevant constraints appears to have an empirical growth rate approximately equal to $Nn/50$. This speeds up LP solving dramatically, because for a fixed number of variables, LP running times scale approximately linearly in the number of constraints [17]. For the 63 DOF humanoid described below, computation times are reduced by two orders of magnitude.

Fig. 14 compares SLP time-scaling to the recent open source implementation by [15] of the classical exact time-scaling algorithm [2] (code accessed May 2012). B-spline path derivative calculations were integrated into the code, and the method was run on the B-spline paths produced in Fig. 13. It worked successfully for a majority of the examples but failed with numerical error in 5 of 13 runs. Failures did not follow any clear pattern. In contrast, with $N = 1024$ grid points, SLP produces trajectories of about 4% slower duration, but runs faster, more scalably, and more reliably.

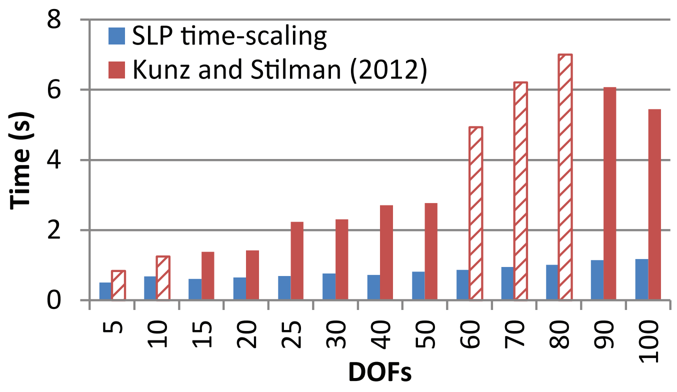
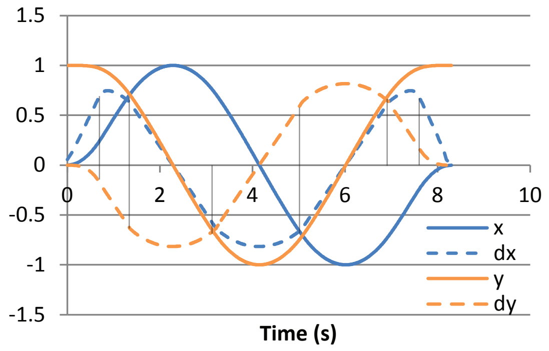


Figure 14: Computation times of the new time-scaling method compared to [15] on the example of Fig. 13. Striped bars indicate failure.



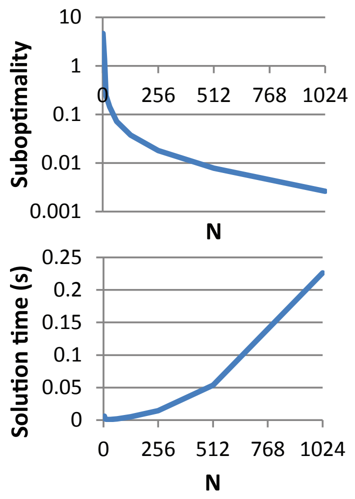
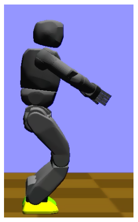
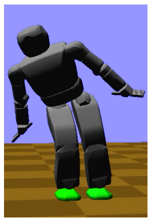
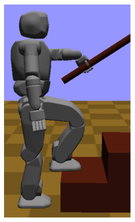
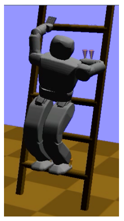


Figure 15: Left: time-optimized trajectory on a unit circle rotating from 0 to 2π, with unit velocity and acceleration bounds in the x and y axes. Vertical lines indicate the points at which the active limits switch from $\dot{x}$ to $\dot{y}$ and vice-versa. Right: Suboptimality $T - T_{opt}$ (on log scale) and running time of the time-scaling method with varying grid size N. Suboptimality scales roughly with $O(N^{-1.19})$ while running time is roughly $O(N^2)$.

Table 1: Performance on humanoid examples, w/o force constraints

Example	**Crouch**	**Sway**	**Stair**	**Ladder**
				
Keyframes	2	3	10	33
Manifolds	1	1	5	13
Contact tol. ϵ	2 mm	2 mm	2 mm	2 mm
Grid size N	128	128	350	1500
Interp. time (s)	0.15	0.09	0.24	2.61
SLP time (s)	0.23	0.24	0.79	1.50
Opt. duration (s)	1.76	9.31	10.8	35.2
Tri. vel. dur. (s)	2.40	12.6	24.2	96.0

7.2 Performance and Grid Size

Fig. 15 shows the solution for a unit circle path with axis-wise velocity and acceleration bounds. The optimal time scaling is acceleration-limited, with the slope of either $\dot{x}$ or $\dot{y}$ limited throughout. The solution at a given resolution is suboptimal, but approaches the optimum as the resolution grows finer. These experiments also suggest that running time of SLP grows approximately quadratically in N, which is consistent with empirical observations and smoothed analyses of linear program running time [17].

7.3 Humanoid Robot Locomotion

We tested the method on a simulated model of the KAIST Hubo-II+ humanoid robot. The physical robot is 130 cm tall and weighs 42 kg with 37 actuated degrees of freedom. The configuration space model includes individually actuated finger joints and the 6DOF base translation and rotation, leading to a 63-D configuration space $SE(3) \times \mathbb{R}^{57}$.

Table 1 displays timing results for four example motions, not including force constraints. In **Crouch** the hands are maintained at a constant relative translation and orientation, as though holding an object with a two-handed grasp. Both feet are constrained to lie on the floor with $\epsilon = 2$ mm. The start and end configurations are constructed to be kinematically feasible, i.e., quasi-statically balanced and collision free. **Sway**, also shown in Fig. 1, is a side-to-side swaying motion. **Stair** is a hand-supported stair climb that grasps the rail and takes a single step to the first step. The robot traverses five manifolds: LF+RF (left foot + right foot support), LH (left hand support)+LF+RF , LH+LF, LH+LF+RF, and LH+RF. A sequence of 10 kinematically-feasible configurations are provided to the algorithm. **Ladder** is a backwards ladder climb that moves 6 steps up a ladder. Steps alternate between 3-limb and 4-limb contact, so the robot moves between 13 contact submanifolds total. (The robot uses the 4-limb contact stages to shift its center of mass.) A sequence of 33 kinematically-feasible configurations are provided as input. To interpolate multi-step paths, configurations at each contact stage are interpolated and then the resulting paths are concatenated together. Short trajectories can be generated in a fraction of a second, whereas the longest ladder climbing trajectory takes approximately 4 s. Each segment with fixed contact points is optimized individually.

In all cases, it is worth the added expense in absolute terms to compute the optimal time scaling rather than to rely on simpler heuristics. The last row in Table 1 (Tri. vel. dur.) compares one such heuristic: a triangular velocity profile that speeds up and then slows down to a stop at each contact stage. The apex of the triangle governs the speed of the trajectory and is scaled to the dynamic limits of the robot. The method is only slightly faster to compute than time-scaling, yet produces substantially slower paths. For the ladder climbing example, time-scaling saves 59 s of computation + execution time.

Fig. 16 shows another ladder-climbing motion, but with force constraints included in the time-scaling. It consists of 16 changes of contact and climbs up two rungs of the ladder, and has an average of 32 contact points. The optimized motion is approximately 27 s in duration. Unlike the prior examples, in this example the SLP is jointly optimized across all changes of contact. Feasible set computation time increases in N, the number of DOFs, and the number of contact points. In these experiments the average number of feasible set boundaries per grid point ranges between 5–7, so the time-scaling cost is primarily only dependent on N.

7.4 Simulation Experiments

The next set of experiments demonstrate the value of dynamic interpolation for feasibility constraints such as balance, using physics simulation as an approximation of real-world conditions. We use the Klamp't simulator, which extends the Open Dynamics Engine rigid body simulator with im-

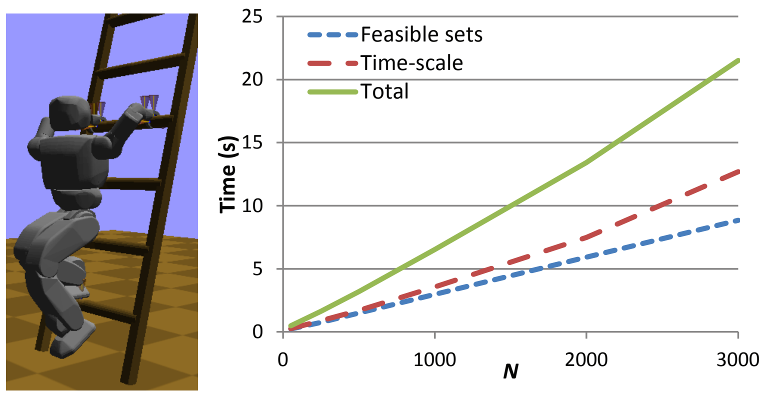


Figure 16: A ladder-climbing motion for the Hubo-II+ humanoid robot with contact force constraints. Variation in running time vs. the number of grid points N, broken into the computation of feasible sets and time-scaling.

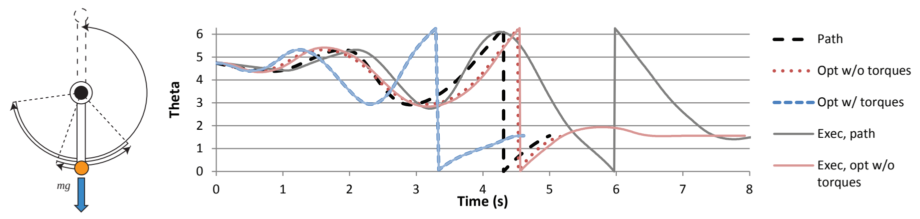


Figure 17: A pendulum swingup problem with simulated execution. Mapping the original spline path uniformly to a 5 s duration (Path) causes the robot to overshoot (Exec. path). Optimizing the time-scaling without considering torque limits (Opt. w/o torque) causes the robot to overshoot the target (Exec, opt. w/o torque). Considering torque limits, the optimized time scaling (Opt. w/ torque) is almost indistinguishable from its execution (Exec, opt. w/ torque).

proved contact handling for triangle-mesh collisions and emulation for robot actuators and sensors [11].

Fig. 17 shows simulated execution results for a simple pendulum swing-up task where input keyframes are not quasistatically stable. The pendulum has a mass of 1 kg, length of 1 m and moves under the influence of gravity. The motor's torque limit of 5.6 Nm is not sufficient to lift the pendulum directly. The robot performs trajectory following PD control with gravity compensation. When the kinematic path is uniformly mapped to a 5 s duration, the pendulum completely overshoots the target. With the path mapped to $\leq$4 s or $\geq$9 s duration, the robot does not even reach the upright position (not shown). Opt w/o Torque shows a dynamic interpolation with no torque limits and approximate acceleration bounds of 5.6 rad / s^2. The optimized motion had duration 5.28 s and does not take advantage of the dynamic effects of gravity. As a result, during execution the robot overshoots the target. With torque bounds, the optimized motion (Opt w/ Torque) has duration 4.6 s and is executed accurately. The optimized path is solved in 89 ms with N=100.

Fig. 1 illustrates the importance of dynamic balance constraints for humanoids. A side-to-side configuration-space path on the Hubo-II+ is executed in simulation using four time parameterizations:

- *Uniform, 5 s duration*: the robot slips at the beginning of the path as it jerks to a start, and falls over at the end of the path.
- *Uniform, 10 s duration*: ultimately, the robot does not fall over, but it still wobbles on its feet.
- *Dynamic interpolation, acceleration constraints only*: the robot still slows down too quickly and tips over.
- *Dynamic interpolation, all constraints*: the robot stays upright with an optimized duration of 4.99 s.

The latter method uses 59 contact points and $N = 100$, taking 2.40 s to precompute feasible sets and 2.46 s to solve the SLP. The dynamic feasible sets at each grid point averaged 9.1 boundaries. The shade of green on the feet indicates the force magnitude; note that in our method the robot decelerates as quickly as possible by shifting all of its weight onto its right foot (second to last frame, last row).

Fig. 18 shows an execution of the Crouch motion as simulated in a rigid body physics simulator. This is a fairly realistic test of how the method would perform on a physical robot, because realistic motor PID controllers

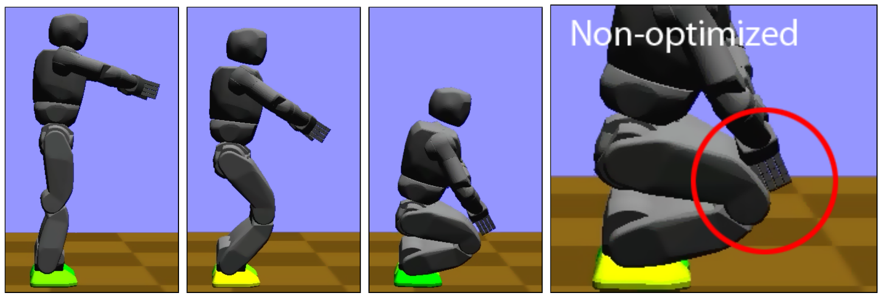


Figure 18: Left: Frames from a simulation of a dynamically-optimized trajectory for the Hubo to crouch while holding an object with both hands. Right: a non-optimized trajectory abruptly stops at the end of simulation, causing the legs and arms to overshoot the target and cause a collision.

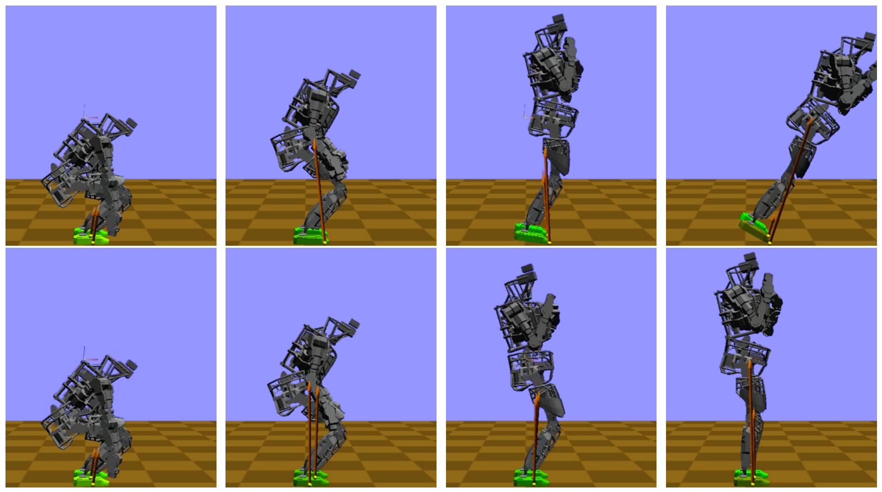


Figure 19: Standing-up motions on a simulated ATLAS humanoid. Top row: the robot tips over when executing a dynamic interpolation without contact force constraints. Bottom row: with force constraints, the robot stays balanced.

and frictional ground contact forces are simulated. The motion produced by the method is performed as desired, without incident. In contrast, direct execution of the path *without* dynamic optimization causes large jerks at the start and end of motion, causing the robot to both overshoot the endpoint and wobble.

In Fig. 19 the Boston Dynamics ATLAS robot is simulated standing up from a squat. The input motion has 45 DOF and the stance is modeled with 30 contacts. Without force/torque constraints, the optimized trajectory was generated about twice as quickly and has duration 3.43 s, but the robot falls over at the end of the trajectory. When considering force/torque constraints, the optimized trajectory was generated in 1.44s and has duration 6.63 s, and the robot stays upright.

7.5 Implementation and Usage

The technique presented in this paper is implemented in C++ the Manifold Interpolation and Time-Optimal Smoothing (MInTOS) library, which is publicly available at `http://www.iu.edu/~motion/mintos/`. Its API supports submanifold interpolation of 2 or more keyframes given arbitrary vector fields $C(q) = 0$, as well as time-scaling with 1) velocity and acceleration bounds enforced conservatively, and/or 2) torque and force constraints via the collocation point method.

Usage. In *basic* mode, Mintos performs projection and time-scaling with velocity and acceleration bounds in one step, and the user supplies bounds and subroutines for kinematic constraint evaluation to customize the procedure to a given robot. Altogether, the user input includes:

- The sequence of keyframes,
- Subroutines for evaluating $C(q)$ and its Jacobian $\frac{\partial C}{\partial q}(q)$,
- The termination threshold $2\epsilon/M$ used in recursive Hermite projection,
- The number of grid points N,
- The velocity and acceleration limits v_L, v_U, a_L, a_U.
- Optionally, subroutines for calculating geodesics and their derivatives for non-Cartesian configuration spaces may be provided.

In *contact* mode, Mintos accepts torque and frictional force constraints. It requires that the user performs kinematic interpolation and time-scaling in separate steps, and that the user supply contact information and subroutines to evaluate the components of the dynamic equation. In addition to the basic mode parameters, the user supplies:

- A list of grid points (in lieu of N, this provides finer control),
- Subroutines for calculating each item $B(q)$, $C(q, \dot{q})$), $G(q)$, and $J_i(q)$ in the robot's dynamic equation,

- Torque bounds τ_L, τ_U,
- A list of contact points, normals, and friction coefficients at each contact phase,
- The number of edges in the polyhedral approximation to the friction cone.

The Klamp't library provides a more convenient interface to Mintos that supplies the geodesic, kinematic constraints, and dynamics subroutines for a given robot model [11].

Parameter selection. The constraint tolerance $\epsilon > 0$ should be set depending on how much error in the contact equalities can be tolerated by the robot; higher values can be used with passively compliant joints, or with compliant surfaces in contact. Lower values should be used for stiff systems, at the expense of higher running time. The parameter β is set by default to 0.9. We find that bound values typically have a minor effect on performance.

In basic mode, the grid resolution N should be set to achieve the desired tradeoff between computation time and path optimality. In contact mode, N should be set according to achieve the desired tradeoff between computation time and severity of constraint violations between collocation points. In either case, experimental tuning is necessary. In our experiments, setting $N = 1000$ typically leads to nearly optimal solutions with approximately 1 s solution times in basic mode. In contact mode, a much lower value is recommended (say, $N = 100$) to achieve interactive performance due to the significant extra cost of feasible set precomputation.

8 Conclusion

This paper presented a fast method for interpolating robot keyframes under kinematic and dynamic contact constraints. Three contributions make it possible: recursive Hermite projection for calculating a kinematic path within a given tolerance, polytope projection to compute dynamically feasible sets of timing derivatives, and convex optimization for time-scaling. The algorithm computes optimized trajectories in seconds for high-dimensional humanoid robots and complex contact formations. It is implemented in the open-source Manifold Interpolation and Time-Optimal Smoothing (MInTOS) library, available at `http://www.iu.edu/~motion/mintos/`. In future work we plan to embed the method into higher-level optimizations to choose keyframes, study its tradeoffs between resolution and robustness, and to apply it to other problems such as dynamic locomotion on uneven terrain and nonprehensile manipulation.

Acknowledgment

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) award # N65236-12-1-1005 for the DARPA Robotics Challenge. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[1] D. Berenson, S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning on constraint manifolds. In *IEEE Int. Conf. Rob. Aut.*, May 2009.

[2] J. E. Bobrow. Optimal robot path planning using the minimum-time criterion. *IEEE J. of Robotics and Automation*, 4:443–450, 1988.

[3] T. Bretl and S. Lall. A fast and adaptive test of static equilibrium for legged robots. In *IEEE Int. Conf. Rob. Aut.*, Orlando, FL, 2006.

[4] P. H. Channon, S. H. Hopkins, and D. T. Pham. A variational approach to the optimization of gait for a bipedal robot. *Proc. Inst. Mech. Eng., Part C: J. Mech. Eng. Sci.*, 210(2):177–186, 1996.

[5] E. A. Croft, B. Benhabib, and R. G. Fenton. Near-time optimal robot motion planning for on-line applications. *J. Robotic Systems*, 12(8):553–567, 1995.

[6] P. Crouch and F. S. Leite. The dynamic interpolation problem: On riemannian manifolds, lie groups, and symmetric spaces. *J. Dynamical and Control Systems*, 1:177–202, 1995.

[7] A. Escande, A. Kheddar, S. Miossec, and S. Garsault. Planning support contact-points for acyclic motions and experiments on hrp-2. In *Int. Symp. Experimental Robotics*, pages 293–302, 2008.

[8] R. Featherstone. *Rigid body dynamics algorithms*, volume 49. Springer Berlin, 2008.

[9] H. Geering, L. Guzzella, S. Hepner, and C. Onder. Time-optimal motions of robots in assembly tasks. *IEEE Trans. Automatic Control*, 31(6):512 – 518, June 1986.

[10] K. Harada, K. Hauser, T. Bretl, and J.-C. Latombe. Natural motion generation for humanoid robots. In *IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, pages 833 –839, Oct. 2006.

[11] K. Hauser. Robust contact generation for robot simulation with unstructured meshes. In *International Symposium on Robotics Research*, Singapore, Dec. 2013.

[12] C. Jones, E. Kerrigan, and J. Maciejowski. Equality set projection: A new algorithm for the projection of polytopes in halfspace representation. Technical Report CUED/F-INFENG/TR.463, ETH Zurich, 2004.

[13] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE Int. Conf. Rob. Aut.*, volume 2, pages 1620–1626. IEEE, 2003.

[14] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, 2010.

[15] T. Kunz and M. Stilman. Time-optimal trajectory generation for path following with bounded acceleration and velocity. In *Robotics: Science and Systems*, July 2012.

[16] C. K. Liu. Dextrous manipulation from a grasping pose. *ACM Transactions on Graphics (SIGGRAPH)*, 28(3), Aug. 2009.

[17] N. Megiddo. On the complexity of linear programming. In *Advances in economic theory*, pages 225–268. 1987.

[18] I. Mordatch, Z. Popović, and E. Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12, pages 137–144, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association.

[19] I. Mordatch, E. Todorov, and Z. Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (SIGGRAPH)*, 31(4), 2012.

[20] J. Pan, L. Zhang, and D. Manocha. Fast smoothing of motion planning trajectories using b-splines. In *Robotics: Science and Systems*, 2011.

[21] F. C. Park and B. Ravani. Smooth invariant interpolation of rotations. *ACM Trans. Graphics*, 16:277 – 295, July 1997.

[22] Q.-C. Pham and Y. Nakamura. Kinodynamic planning in the configuration space via velocity interval propagation. In *Robotics: Science and Systems*, 2013.

[23] M. Posa and R. Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Workshop Algo. Found. Robotics*, Cambridge, MA, 2012.

[24] N. Ratliff, M. Zucker, J. A. D. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[25] L. Roussel, C. Canudas-de Wit, and A. Goswami. Generation of energy optimal complete gait cycles for biped robots. In *IEEE Int. Conf. Rob. Aut.*, volume 3, pages 2036 –2041, May 1998.

[26] Z. Shiller and H. Lu. Computation of path constrained time-optimal motions with dynamic singularities. *ASME J. Dyn. Syst. Measure. Control*, 114:34–40, 1992.

[27] K. Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19(3):245–254, 1985.

[28] S. Srinivasa, M. Erdmann, and M. T. Mason. Control synthesis for dynamic contact manipulation. In *IEEE Int. Conf. Rob. Aut.*, pages 2523 – 2528. IEEE, Apr. 2005.

[29] M. Stilman. Task constrained motion planning in robot joint space. In *IEEE/RSJ Int. Conf. Intel. Rob. Sys.*, 2007.

[30] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Trans. Automatic Control*, 54(10):2318 –2327, Oct. 2009.

[31] M. Vukobratovic and M. Kircanski. A method for optimal synthesis of manipulation robot trajectories. *J. Dyn. Sys. Measure. Control*, 104(2):188–193, 1982.

[32] J. H. Yakey, S. LaValle, and L. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Trans. Robot. and Autom.*, 17(6):951–958, 2001.