

Fast Interpolation and Time-Optimization on Implicit Contact Submanifolds

Kris Hauser

School of Informatics and Computing, Indiana University at Bloomington

Email: hauserk@indiana.edu

Abstract—This paper presents a method for generating smooth, efficiently-executable trajectories for robots under contact constraints, such as those encountered in legged locomotion and object manipulation. It consists of two parts. The first is an efficient, robust method for constructing C^1 interpolating paths between configuration/velocity states on implicit manifolds. The second is a robust time-scaling method that solves for a minimum-time parameterization using a novel convex programming formulation. Simulation experiments demonstrate that the method is fast, scalable to high-dimensional robots, and numerically stable on humanoid robot locomotion and object manipulation examples.

I. INTRODUCTION

Trajectory optimization helps robots complete tasks faster, consume less energy, appear more human-like, and interact with humans more naturally. Although trajectory optimization of free-space motions is a well-studied classical problem [7, 16], less attention has been given to the setting of robots that make and break contact. The major challenge with contact is that it imposes kinematic constraints that limit motion to a *nonlinear submanifold* in the robot’s joint space. Prior work on optimizing robot locomotion and manipulation trajectories [3, 6, 8, 11, 12] relies on descent-based approaches that are often extremely slow, taking minutes or hours to complete, and require careful choice of initial trajectories to avoid falling into local minima. As a result, these methods are best suited for off-line trajectory generation. Major improvements in reliability and speed are needed to approach the goal of *on-line* trajectory generation, which requires sub-second running time and no manual intervention.

This paper presents a new method for generating time-optimized interpolating trajectories under contact constraints specified as an implicit function. It uses a two-stage approach [2] that decouples geometric and temporal aspects of the problem: first, it handles kinematic constraints to generate a geometric path, and then it performs a time-scaling optimization to minimize the execution time along the path while respecting joint-wise velocity and acceleration limits.

New technical contributions in each stage makes the method fast enough for on-line use. First, I present a recursive spline-based projection method that generates twice-differentiable paths on a contact submanifold represented by an implicit nonlinear function. The path is proven to achieve an arbitrarily tight tolerance $\epsilon > 0$ on constraint violation errors. Second, I present a new time-scaling formulation as a convex, linearly constrained optimization problem in a *squared-rate* parameter

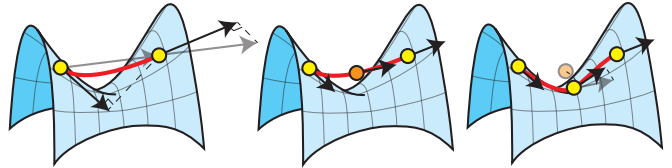


Fig. 1. Interpolation begins with a smooth Hermite curve connecting the endpoints, and recursively bisects and projects the midpoint and its derivative onto the constraint manifold.

space. The globally optimal time-scaling can be computed reliably using sequential linear programming. Moreover, empirical evidence suggests that the number of relevant constraints is only weakly linear in dimensionality, with a small leading coefficient (approximately 0.02 in the examples in this paper). Major scalability improvements come from an intermediate processing step that eliminates irrelevant constraints quickly.

Experiments on a variety of examples demonstrate that the method generates smooth, dynamically-feasible trajectories in interactive fashion for robots with up to 100 DOF and multiple limbs in contact. Trajectories for a 63-DOF humanoid robot performing manipulation and locomotion tasks are interpolated and optimized in fractions of a second.

II. RELATED WORK

Smooth interpolation on manifolds is well studied in the case where the manifold is equipped with a geodesic (e.g., $SO(3)$ [5]). For many-DOF robots under arbitrary contact constraints, geodesics are difficult to derive. Other authors have considered the problem of planning contact-constrained robot motions for manipulation tasks [1, 14, 17]. 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 cubic splines better match the curvature of contact submanifolds, and appear to generate paths with lower maximum error.

A common trajectory optimization approach for legged robots is to formulate a large nonlinear optimization problem over both joint trajectories and timing [3, 6, 8, 11, 12]. Because of their computational expense, these methods appear best suited for generating gait cycles [3, 11, 12] or offline primitives to be reused later [6, 8]. Full trajectory optimization poses other challenges as well. Most nonlinear optimization solvers are only able to find local optima, and hence must

be provided with good initial trajectories. Another problem is maintaining loop closure constraints at contacts. Some authors parameterize the manifold of closed-loop configurations at each possible contact state [3, 6, 8], while others enforce loop closure implicitly by adding nonlinear equality constraints at collocation points along the trajectory [11]. 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 is 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.

Two-stage trajectory generation [2, 4] first computes a path and then optimizes the speed according to dynamic constraints in a second stage. This 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 results are usually satisfactory in practice given their orders of magnitude speed up. The classical method integrates the time scaling variable along dynamic limits [2], but as identified in [9, 13], 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 optimization as a second-order cone program (SOCP) [15]. This paper presents a simpler convex formulation that can be solved via sequential linear program (SLP), for which fast and robust implementations are widely available. 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 [15] solved a 6D one.

III. PROBLEM STATEMENT

Consider the problem of generating a trajectory $y(t) : [0, T] \rightarrow \mathbb{R}^n$ connecting start and goal configurations q_s and q_g and velocities \dot{q}_s and \dot{q}_g . It is required to satisfy kinematic constraints $C(q) = 0$, with $C : \mathbb{R}^n \rightarrow \mathbb{R}^m$ a set of smooth, nonlinear constraints, and its first and second derivatives are required to satisfy velocity and acceleration bounds:

$$v_L \leq \dot{y}(t) \leq v_U \text{ for all } t \in [0, T] \quad (1)$$

$$a_L \leq \ddot{y}(t) \leq a_U \text{ for all } t \in [0, T] \quad (2)$$

where all inequalities are taken element-wise. It is assumed that derivatives of C are nondegenerate when $C(q) = 0$ and that there exists a Lipschitz constant M such that $\|C(p) - C(q)\| \leq M\|p - q\|$ for all $p, q \in \mathbb{R}^n$.

The approach operates in two stages:

- 1) *Recursive Hermite projection.* Constructs a continuously differentiable geometric path $p(s) : [0, 1] \rightarrow \mathbb{R}^n$ that connects q_s and q_g and satisfies $C(q) = 0$.
- 2) *Convex optimization time scaling.* Optimizes a time parameterization $s(t) : [0, T] \rightarrow [0, 1]$ of p such that $y(t) = p(s(t))$ satisfies (1–2) and minimizes T .

These two operations are roughly independent, and in particular, the time-scaling method can be applied to arbitrary cubic

splines. Each will be described in more detail in the following sections.

IV. RECURSIVE HERMITE PROJECTION

Recursive projection generates a continuously differentiable, piecewise polynomial path that satisfies $C(q) = 0$ 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 $C(q) = 0$ (Fig. 1).

Hermite curves $p(u)$ 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 $p^A(u)$ and $p^B(u)$ connected at the midpoint $p(0.5)$ with tangent $p'(0.5)$.

Given a threshold ϵ on constraint violation errors and a growth limit $\beta \in (0.5, 1)$, recursive projection proceeds as follows:

- 1) Begin with a Hermite curve $p(u)$ connecting the start and end configurations. If no tangent directions are prescribed, obtain tangents by projecting the straight-line direction onto the nullspace of $C(q_s)$ and $C(q_g)$.
- 2) If $\text{len}(p) \leq 2\epsilon/M$, return ‘success’.
- 3) Otherwise, subdivide $p(u)$ 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 $C(x) = 0$ using a Newton-Raphson solver. Project v onto the nullspace of $C(x)$. 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 $\max(\text{len}(p^A), \text{len}(p^B)) \leq \beta \cdot \text{len}(p)$, return ‘convergence failure’. Otherwise, repeat Lines 2–5 on both halves.

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

The algorithm terminates when $\text{len}(p)$ is small enough, where $\text{len}(p)$ is an upper bound on the length of the Hermite curve (Fig. 2). To obtain this bound, it computes the length of corresponding Bezier polygon, $\text{len}(p) = \|P_0 - P_1\| + \|P_1 - P_2\| + \|P_2 - P_3\|$. The growth condition in Line 5 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 $\text{len}(p) \leq \beta^d \text{len}(p_0)$ must hold, so given the tolerance in Line 2, the terminal depth is no more than

$$d_{\max}(\epsilon, \beta) = \frac{\log(2\epsilon/M \cdot \text{len}(p_0))}{\log(\beta)} + 1 \quad (3)$$

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 $\beta = 0.9$.

To interpolate multiple points, standard techniques can be used to generate a sequence of Hermite curves with smooth

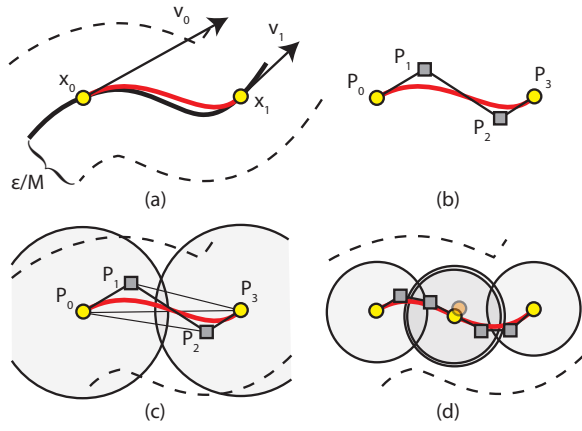


Fig. 2. (a) Illustrating the recursion termination criterion, proving that the path p lies within distance ϵ/M of the submanifold in configuration space, where M is a Lipschitz constant. (b) The Bezier polygon corresponding to p . (c) p is guaranteed to lie within the union of spheres centered at the 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.

tangent vectors at each point, e.g., Cardinal splines. Tangents are projected to the manifold before recursive projection begins on each curve. The method can also be adapted to check collision and other infeasibility constraints during bisection, e.g., for use in motion planning. To do so, collisions are checked at each midpoint x_m in Line 4 and along all leaf curve segments at the end.

A. 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 $\gamma(\epsilon, \beta) \cdot \text{len}(p_0)$ where p_0 is the initial curve.
- Satisfies $\|C(p(u))\| \leq \epsilon$ for all $u \in [0, 1]$.

I now prove the latter claim.

Theorem 1: The output path $p(u)$ satisfies $C(p(u)) \leq \epsilon$ for all $u \in [0, 1]$.

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

By the Lipschitz condition,

$$\|C(p(u))\| = \|C(p(u)) - C(x_0)\| \leq M \min \|p(u) - x_0\|, \|p(u) - x_1\| \quad (4)$$

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. 2), and by the convex hull property, $p(u)$ is contained entirely within the convex hull of P_0, \dots, P_3 . The

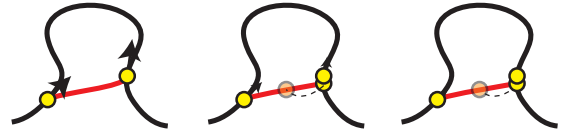


Fig. 3. Recursive projection can fail when recursion fails to make progress along a manifold. This condition is detected in Line 5.

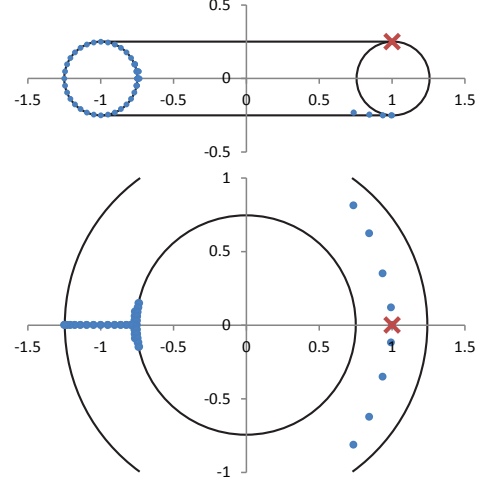


Fig. 4. Interpolating on an implicit torus embedded in \mathbb{R}^3 . Recursive projection successfully connects the source point (X) to almost all other points on the torus, except for the indicated failure points (circles). Views from the side and above.

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 $\text{len}(p)/2$ centered at each of p 's endpoints.

Obviously, $\overline{P_0P_3}$ is contained within $B_0 \cup B_3$ because $\text{len}(p) \geq \|P_3 - P_0\|$. 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 within $B_0 \cup B_3$. 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 $\text{len}(p)/2$ from either endpoint.

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

The interpolator is not complete; it may fail if 1) a midpoint becomes stuck in a local minima of $\|C(q)\|$ during projection, and 2) the recursion fails to make progress along the manifold (Fig. 3). I 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. 4 shows that failures occur only in a few narrow bands.

B. 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 (SE(2)) and free-floating bases of legged robots (SE(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 $g(t; a, b)$ which interpolates between a and b along such a length-minimizing path, with $t \in [0, 1]$. The exponential map \exp_q is defined such that $g(t; a, b) = \exp_a(t\dot{g}(t; a, b))$. The partial derivatives of g w.r.t. t , a , and b must also be supplied. Closed form expressions exist for SO(2) and SO(3) [10].

Hermite interpolation on \mathcal{M} is performed using the classic de Casteljau construction of a Bezier curve [5]. 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 points:

$$\begin{aligned} P_0 &= x_0 & P_1 &= \exp_{x_0}(\frac{1}{3}v_0) \\ P_2 &= \exp_{x_1}(-\frac{1}{3}v_1) & P_3 &= x_1 \end{aligned} \quad (5)$$

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

$$\begin{aligned} p(u) &= g(u; P_{012}(u), P_{123}(u)) \\ P_{012}(u) &= g(u; P_{01}(u), P_{12}(u)) \\ P_{123}(u) &= g(u; P_{12}(u), P_{23}(u)) \\ P_{01}(u) &= g(u; P_0, P_1) \\ P_{12}(u) &= g(u; P_1, P_2) \\ P_{23}(u) &= g(u; P_2, P_3) \end{aligned} \quad (6)$$

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:

$$\begin{aligned} p'(u) &= \left(\dot{g} + \frac{\partial g}{\partial a} P'_{012}(u) + \frac{\partial g}{\partial b} P'_{123}(u) \right) \Big|_{u, P_{012}(u), P_{123}(u)} \\ P'_{012}(u) &= \left(\dot{g} + \frac{\partial g}{\partial a} P'_{01}(u) + \frac{\partial g}{\partial b} P'_{12}(u) \right) \Big|_{u, P_{01}(u), P_{12}(u)} \\ P'_{01}(u) &= \dot{g}(u; P_0, P_1) \end{aligned} \quad (7)$$

with similar formulas holding for P'_{123} , P'_{12} , and P'_{23} .

V. CONVEX OPTIMIZATION TIME SCALING

The time-scaling solves for a time parameterization $s(t)$ of the path p that minimizes execution time T . The algorithm use a piecewise quadratic formulation of $s(t)$, and demonstrate that with the proper parameterization this gives rise to a convex program with linear inequalities. Dynamic feasibility is guaranteed over the entire time-scaled trajectory, rather than just at a finite set of points, due to the use of conservative interval bounding on the path's derivatives.

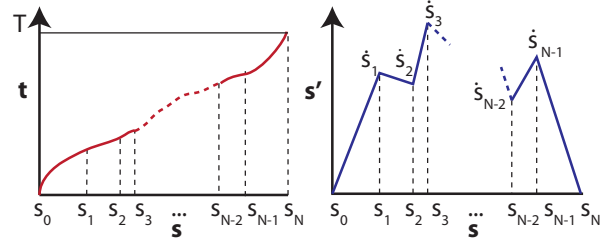


Fig. 5. A piecewise quadratic time-scaling $s(t)$ is parameterized by the rates $\dot{s}_0, \dots, \dot{s}_N$.

The time scaling formulation rewrites \dot{y} and \ddot{y} in terms of p and s as follows:

$$\dot{y}(t) = p'(s(t))s'(t) \quad (8)$$

$$\ddot{y}(t) = p''(s(t))s'(t)^2 + p'(s(t))s''(t). \quad (9)$$

With p fixed, the problem of finding a dynamically-feasible path is reduced to one of finding a monotonically increasing mapping $s(t)$ such that \dot{y} and \ddot{y} given by (9) satisfy the conditions (1,2). The problem requires that $\dot{s}(t)$ to be continuous so that y is twice differentiable, and also that the trajectory start and stop at zero velocity: $\dot{s}(0) = \dot{s}(T) = 0$.

A. Parameterizing the time scaling by gridding the path domain

Grid the path domain $[0, 1]$ into N intervals $[s_k, s_{k+1}]$, $k = 0, \dots, N-1$ and let $s(t)$ be piecewise quadratic on each interval. Let Δs_k be the duration of the k 'th interval. This parameterization formulates s as a piecewise quadratic, twice differentiable curve parameterized by $N+1$ rate parameters $\dot{s}_0, \dots, \dot{s}_N$ at segment division points (Fig. 5).

Consider an interval k and its (unknown) time interval $[t_k, t_{k+1}]$. Over this interval, $s(t)$ is fully determined by endpoint velocities $s'(t_k) = \dot{s}_k$ and $s'(t_{k+1}) = \dot{s}_{k+1}$. Verify that the following equation satisfies the desired terminal constraints on the interval $[t_k, t_{k+1}]$ with $\Delta t_k = t_{k+1} - t_k = \frac{2\Delta s_k}{\dot{s}_k + \dot{s}_{k+1}}$.

$$s(t) = \frac{\dot{s}_{k+1}^2 - \dot{s}_k^2}{4\Delta s_k} (t - t_k)^2 + \dot{s}_k (t - t_k) + s_k \quad (10)$$

So, s' is a linear interpolation between s_k and s_{k+1} and $s''(t) = (\dot{s}_{k+1}^2 - \dot{s}_k^2)/(2\Delta s_k)$ is a constant independent of u . Parameterizing in terms of $u(t) = (s'(t) - \dot{s}_k)/(\dot{s}_{k+1} - \dot{s}_k)$ to map $[t_k, t_{k+1}]$ to the range $[0, 1]$, (9) becomes:

$$\dot{y}(t(u)) = p'(s(u))((1-u)\dot{s}_k + u\dot{s}_{k+1}) \quad (11)$$

$$\begin{aligned} \ddot{y}(t(u)) &= p''(s(u))((1-u)\dot{s}_k + u\dot{s}_{k+1})^2 \\ &\quad + p'(s(u)) \frac{\dot{s}_{k+1}^2 - \dot{s}_k^2}{2\Delta s_k} \end{aligned} \quad (12)$$

These conditions must be satisfied for each interval $k = 1, \dots, N$ and $u \in [0, 1]$.

B. Derivative bounding

The next step converts the continuous constraints (1,2) into finite ones. An interval arithmetic bounding technique is used to provide a conservative but asymptotically tight bound, in that as the grid grows finer ($N \rightarrow \infty$) the discretized constraints approach the true constraints on s in the limit.

For all $u \in [0, 1]$, the interval k must satisfy:

$$v_L \leq p'(s(u))((1-u)\dot{s}_k + u\dot{s}_{k+1}) \leq v_U \quad (13)$$

$$a_L \leq p''(s(u))((1-u)\dot{s}_k + u\dot{s}_{k+1})^2 + p'(s(u)) \frac{\dot{s}_{k+1}^2 - \dot{s}_k^2}{2\Delta s_k} \leq a_U \quad (14)$$

Over the domain $[s_k, s_k + \Delta s_k]$, the derivatives of p are bounded via intervals $p'(s(u)) \in [v_k^L, v_k^U]$ and $p''(u) \in [a_k^L, a_k^U]$. Here the notation $[a, b]$ with vectors a and b indicates an axis-aligned box in \mathbb{R}^n . Since each Hermite curve 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 term in (13), observe that $\dot{s}(u)$ is a linear interpolation with extrema \dot{s}_k at $u = 0$ and \dot{s}_{k+1} at $u = 1$. Hence, (13) is proven feasible if the constraints

$$\begin{aligned} v_L &\leq v_k^L \dot{s}_k & v_L &\leq v_k^U \dot{s}_{k+1} \\ v_k^U \dot{s}_k &\leq v_U & v_k^L \dot{s}_{k+1} &\leq v_U \end{aligned} \quad (15)$$

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

Eq. (14) 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 $((1-u)\dot{s}_k + u\dot{s}_{k+1})^2$ are obtained at the endpoints \dot{s}_k^2 and \dot{s}_{k+1}^2 . Hence, (14) is proven feasible if the conditions:

$$\dot{s}_k^2 [a_k^L, a_k^U] + \frac{1}{2\Delta s_k} [v_k^L, v_k^U] (\dot{s}_{k+1}^2 - \dot{s}_k^2) \in [a_L, a_U] \quad (16)$$

$$\dot{s}_{k+1}^2 [a_k^L, a_k^U] + \frac{1}{2\Delta s_k} [v_k^L, v_k^U] (\dot{s}_{k+1}^2 - \dot{s}_k^2) \in [a_L, a_U] \quad (17)$$

are satisfied.

C. Linear Constraints in the Squared-rate Space

A transformation of the parameter space turns these quadratic constraints into linear inequalities. The *squared-rate parameter space* $\theta_0 = \dot{s}_0^2, \theta_1 = \dot{s}_1^2, \dots, \theta_N = \dot{s}_N^2$ allows (16,17) to be rewritten:

$$\theta_k [a_k^L, a_k^U] + \frac{1}{2\Delta s_k} [v_k^L, v_k^U] (\theta_{k+1} - \theta_k) \in [a_L, a_U] \quad (18)$$

$$\theta_{k+1} [a_k^L, a_k^U] + \frac{1}{2\Delta s_k} [v_k^L, v_k^U] (\theta_{k+1} - \theta_k) \in [a_L, a_U]. \quad (19)$$

Similarly, (15) is rewritten in terms of θ_k and θ_{k+1} :

$$\theta_k, \theta_{k+1} \in \left[0, \max \left(\frac{v_L}{v_k^L}, \frac{v_U}{v_k^U} \right)^2 \right] \quad (20)$$

with the divisions, maxima, and square taken element-wise.

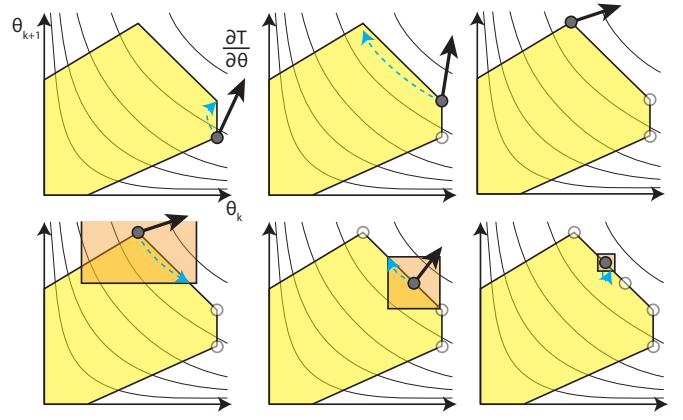


Fig. 6. Illustrating sequential linear programming on a hypothetical 2D slice. The feasible set is shaded, and contours of T are drawn. First, a negated gradient direction is computed (solid arrow), and an LP is solved to obtain the next step (dashed arrow). This continues until the LP move does not decrease T . The LP trust region (shaded square) is shrunk until a decreasing move is found. The process continues until convergence.

D. Solving the Convex Optimization Problem via SLP

Let the feasible region $F \in \mathbb{R}^{N+1}$ be the set of θ subject to (18–20) for $k = 0, \dots, N-1, \theta \geq 0$, and $\theta_0 = \theta_N = 0$. The final optimization problem minimizes time subject to dynamic feasibility:

$$\begin{aligned} \min_{\theta} T(\theta) &= \sum_{k=1}^N \Delta t_k = \sum_{k=1}^N \frac{2\Delta s_k}{\sqrt{\theta_{k+1}} + \sqrt{\theta_k}} \\ \text{s.t. } &\theta \in F. \end{aligned} \quad (21)$$

I now state the main result:

Theorem 2: The optimization problem (21) is convex.

Proof: Each interval statement in (18–20) can be written as up to $4n$ linear inequalities, and hence 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. ■

Over the unbounded positive space $\theta \geq 0$, T approaches a single global minimum value (namely, zero with $\theta \rightarrow \infty$). With bounded F , the following lemma holds:

Lemma 1: The global, unique minimum of (21) in F lies on the boundary of F .

This follows because F is convex and bounded while the unconstrained optimum is unbounded.

The problem is linearly constrained, making it 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. 6). 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

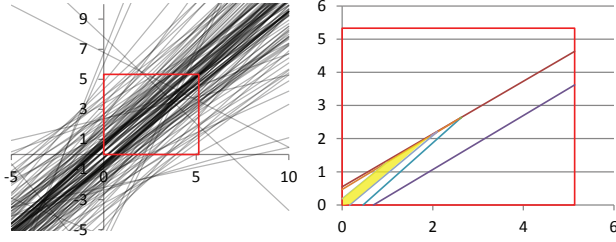


Fig. 7. Of over 500 inequality constraints relating adjacent parameters θ_k and θ_{k+1} , all but 5 are irrelevant to the feasible set. A halfplane intersection procedure calculates the feasible polygon (shaded) by iterated cutting.

- 2) Initialize the trust region size $r = \|\theta\|_\infty$.
- 3) Linearize the objective function about the current solution θ and solve an LP:

$$\min_x \frac{\partial T}{\partial \theta}(\theta)^T x \text{ s.t.} \quad (22)$$

$$x \in F \text{ and } \|x - \theta\|_\infty \leq r.$$

- 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.

E. Culling Irrelevant Constraints

Especially for high-DOF problems, most of the $12nN$ constraints (18–20) will be redundant. These correspond to the dynamic constraints that are non-limiting, e.g., joints that are moving slowly relative to others. LPs are solved more quickly if these are removed, so for each grid interval k we compute the feasible polygon F_k in the (θ_k, θ_{k+1}) plane, and prune out all constraints not supporting an edge of F_k .

To do so, a halfplane intersection routine is invoked. For each k , the halfplane equations of (18–20) that involve only θ_k and θ_{k+1} are calculated. The convex polygon F_k is then incrementally built, starting from the box (20), and planes are added one-by-one to cut away portions of the polygon (Fig. 7). At the end of this process the supporting halfplanes are the boundaries of F_k and are output to the LP.

VI. EXPERIMENTS

All examples in this paper are performed on a 2.67 GHz CPU on a single thread. The method is implemented in C++

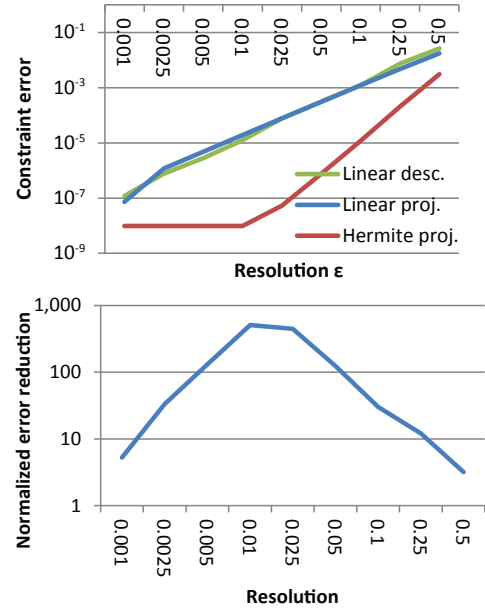


Fig. 8. Comparing the error of recursive Hermite projection against two piecewise linear interpolation techniques as ϵ is varied. Points are projected numerically with tolerance 10^{-8} . Hermite projection still outperforms linear methods by up to several orders of magnitude when normalizing for additional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

using the GLPK library to solve linear programs.

A. Interpolation on Manifolds

Given the same resolution ϵ , the path produced by Hermite projection has lower maximum error than either the linear descent technique of [14] or piecewise linear projection (Fig. 8). 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. 8 compares the error ratio of the linear fit vs Hermite projection for an equivalent computation time, demonstrating that the benefits of Hermite projection outweigh the added overhead.

B. Time Scaling

Fig. 9 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 grows approximately quadratically in N .

Irrelevant constraint pruning provides major speed advantages for high-DOF robots, since the number of relevant constraints is only weakly dependent on n (Fig. 10). For the 63 DOF humanoid described below, computation times are reduced by two orders of magnitude.

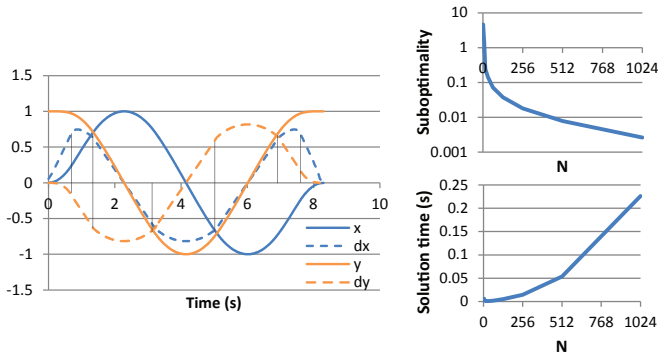


Fig. 9. Top: time-optimized trajectory on a unit circle rotating from 0 to 2π , with unit velocity and acceleration bounds in the x and y axes. Vertical lines indicate the points at which the active limits switch from \dot{x} to \dot{y} and vice-versa. Bottom: Suboptimality (on log scale) and running time of the time-scaling method with varying grid size N . Suboptimality is roughly $O(N^{-1.19})$ while running time is roughly $O(N^2)$.

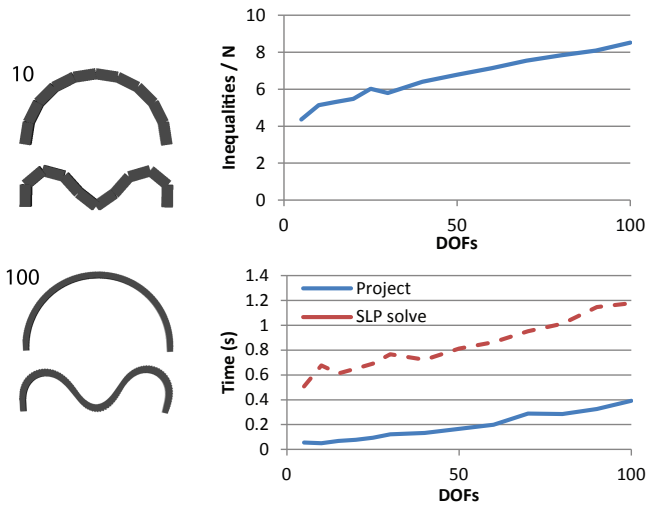


Fig. 10. Performance on planar, endpoint constrained, n -joint robots. Grid size is fixed at $N = 1024$. The number of inequality constraints grows with $12n$ but the number of relevant constraints grows slowly, approximately $O(n/50)$. In the 100D case, nearly 99% of the original constraints are irrelevant. By eliminating those constraints, the SLP solve time is only weakly dependent on n and can still solve 100D problems in approximately 1 s.

Fig. 11 compares SLP time-scaling to the recent open source implementation by Kunz and Stilman [9] 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. 10. 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.

C. Examples on a Humanoid Robot

Finally, the method is tested on a simulated model of the KAIST Hubo-II+ humanoid robot. The physical robot is 130cm tall and weighs 42kg with 37 actuated degrees of freedom. The configuration space model includes individually

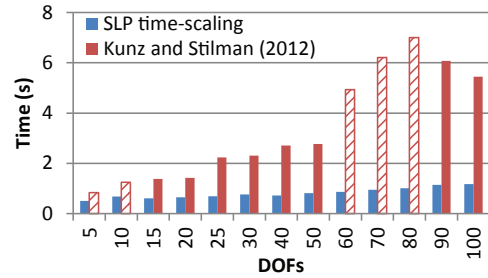


Fig. 11. Computation times of the new time-scaling method compared to Kunz and Stilman [9] on the example of Fig. 10. Striped bars indicate failure.

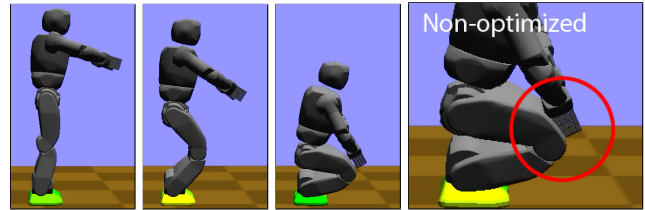



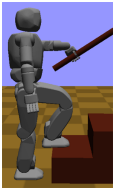
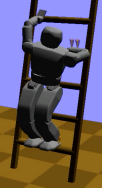
Fig. 12. Left: Frames from a simulation of a dynamically-optimized trajectory for the Hubo to crouch while holding an object with both hands. Right: a non-optimized trajectory abruptly stops at the end of simulation, causing the legs and arms to overshoot the target and cause a collision.

actuated finger joints and the 6DOF base translation and rotation, leading to a 63-D configuration space $SE(3) \times \mathbb{R}^{57}$.

The first example shows a motion with the hands 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. Fig. 12 shows an execution of the 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 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.

Table I displays timing results for this and two more examples. The second example 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. The third example 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

TABLE I
EXAMPLES ON A HUMANOID ROBOT

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

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.

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 I (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.

VII. CONCLUSION

This paper presented a method for generating smooth interpolating trajectories on contact submanifolds represented by implicit nonlinear equality constraints. Contact constraints are satisfied up to an arbitrary tolerance, the robot's velocity and acceleration bounds are strictly satisfied, and computation times scale well to very high-DOF systems. Example videos and an implementation are provided in the Manifold Interpolation and Time Optimal Smoothing (MIntOS) website <http://www.iu.edu/~motion/mintos/>. Future work should study methods for generating interpolation points that yield feasible paths, path optimization, and extending the time scaling approach to consider torque and frictional constraints.

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] 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. URL <http://pic.sagepub.com/content/210/2/177.abstract>.

[4] 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. ISSN 1097-4563. URL <http://dx.doi.org/10.1002/rob.4620120805>.

[5] 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. ISSN 1079-2724. URL <http://dx.doi.org/10.1007/BF02254638>.

[6] 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. URL https://sites.google.com/site/adrienescandehomepage/publications/2008_ISER_Escande.pdf.

[7] 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. ISSN 0018-9286. doi: 10.1109/TAC.1986.1104333.

[8] 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, October 2006. doi: 10.1109/IROS.2006.281733.

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

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

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

[12] 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. doi: 10.1109/ROBOT.1998.680615.

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

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

[15] 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, October 2009. ISSN 0018-9286. doi: 10.1109/TAC.2009.2028959.

[16] M. Vukobratovic and M. Kircanski. A method for optimal synthesis of manipulation robot trajectories. *J. Dyn. Sys. Measure. Control*, 104(2):188–193, 1982. doi: 10.1115/1.3139695.

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