

A General, Fast, and Robust Implementation of the Time-Optimal Path Parameterization Algorithm

Quang-Cuong Pham

School of Mechanical and Aerospace Engineering
Nanyang Technological University, Singapore

Abstract—Finding the Time-Optimal Parameterization of a given Path (TOPP) subject to kinodynamic constraints is an essential component in many robotic theories and applications. The objective of this article is to provide a general, fast and robust implementation of this component. For this, we give a complete solution to the issue of dynamic singularities, which are the main cause of failure in existing implementations. We then present an open-source implementation of the algorithm in C++/Python and demonstrate its robustness and speed in various robotics settings.

I. INTRODUCTION

Time-optimal motion planning plays a key role in many areas of robotics and automation, from industrial to mobile, to service robotics. While the problem of (optimal) *path planning under geometric constraints* can be considered as essentially solved in both theory and in practice (see e.g. [1]), general and efficient solutions to the (optimal) *trajectory planning under kinodynamic¹ constraints* [2] are still lacking. We argue that Time-Optimal Path Parameterization² (TOPP) may constitute an efficient tool to address the latter problem.

There are at least three types of kinodynamic motion planning problems where TOPP is useful or even indispensable. First, some applications such as painting or welding require specifically tracking a predefined path. Second, even when there is no *a priori* necessity to track a predefined path, it can be efficient to *decouple* the optimal trajectory planning problem into two simpler, more tractable subproblems: (i) generate a set of paths in the robot configuration space, (ii) optimally time-parameterize these paths and pick the path with the fastest optimal parameterization [3], [4]. Third, it was recently suggested that TOPP can also be used to address the *feasibility* problem [5], i.e. finding *one* feasible trajectory in a challenging context, as opposed to selecting an optimal trajectory in a context where it is relatively easy to find many feasible trajectories.

Since the path is constrained, the only “degree of freedom” to optimize is the scalar function $t \mapsto s(t)$ which represents the “position” on the path at each time instant. If the system dynamics and constraints are of second-order, one can next search for the optimal function in the 2-dimensional space (s, \dot{s}) . There are basically three families of methods to do so.

Dynamic programming – The first family of methods divide the (s, \dot{s}) plane into a grid and subsequently uses a dynamic programming approach to find the optimal trajectory in the (s, \dot{s}) plane [6].

Convex optimization – The second family of methods discretize only the s -axis (into N segments) and subsequently convert the original problem into a convex optimization problem in $O(N)$ variables and $O(N)$ equality and inequality constraints [7], [8], [9]. These methods have the advantage of being versatile (they can for instance trade off time duration with other objectives such as energy or torque rate) and can rely on existing efficient convex optimization packages.

¹Geometric constraints – such as joints limits or obstacle avoidance – depend only on the configuration of the robot, while kinodynamic constraints – such as bounds on joint velocity, acceleration and torque, or dynamic balance – involve also higher-order time derivatives of the robot configuration.

²Parameterizing a given geometric path consists in finding a time law to traverse the path, thereby transforming it into a *trajectory*. Time-optimal parameterization seeks to minimize the traversal time under given kinodynamic constraints.

Numerical integration – The third family of methods are based on the Pontryagin Maximum Principle: the optimal trajectory in the (s, \dot{s}) plane is known to be “bang-bang” and can thus be found by integrating successively the maximum and minimum accelerations \ddot{s} , see Section II-B for details. This approach is theoretically faster than the previous two since it exploits the bang-bang structure of the optimization problem (and we shall show that it is indeed faster in practice). However, to our knowledge, there is no available general and efficient implementation, perhaps because of the programming difficulties involved by this approach and of the robustness issues associated with the so-called *dynamic singularities* [10], [11], [12], see details in Section II.

Note that all three families can be applied to a wide variety of robot dynamics and constraints, such as manipulators subject to torque bounds [13], humanoids subject to joint velocity and accelerations bounds [11], [8], mobile robots or humanoids subject to balance and friction constraints [14], [15], non-holonomic robots [16], etc.

The goal of this article is to provide a *general, fast and robust* implementation of TOPP. For this, we follow the theoretically faster numerical integration approach. To make it robust, we address the aforementioned critical issue of *dynamic singularities*. Such singularities arise in a large proportion of real-world problem instances of TOPP, and are one of the main causes of failure of existing implementations of the numerical integration approach. Note that dynamic singularities also cause jitters in the convex optimization approach (see e.g. Fig. 4 of [7]) but are probably not as critical there as in the numerical integration approach. Yet, in most works devoted to the TOPP algorithm, from original articles [17], [10], [18], [11] to reference textbooks [19], the characterization and treatment of these singularities were not done completely correctly. In Section II, we derive a complete characterization of dynamic singularities and suggest how to appropriately address these singularities. The development extends our previous contribution in the case of torque bounds (presented at IROS 2013 [12]) to the general case. In Section III, we present an open-source implementation in C++/Python. We show that our implementation is robust and about one order of magnitude faster than existing implementations of the convex optimization approach [7], [9], [8]. This improvement is particularly crucial for the global trajectory optimization problem or the feasibility problem mentioned earlier, which require calling the TOPP routine tens or hundreds of thousands times. Finally Section IV concludes by briefly discussing the obtained results and future research directions.

II. IMPROVING THE ROBUSTNESS OF THE NUMERICAL INTEGRATION APPROACH

A. General formulation of the TOPP problem

Let \mathbf{q} be a n -dimensional vector representing the configuration of a robot system. Consider second-order inequality constraints of the form

$$\mathbf{A}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \mathbf{B}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{f}(\mathbf{q}) \leq 0, \quad (1)$$

where $\mathbf{A}(\mathbf{q})$, $\mathbf{B}(\mathbf{q})$ and $\mathbf{f}(\mathbf{q})$ are respectively an $M \times n$ matrix, an $n \times M \times n$ tensor and an M -dimensional vector.

Note that “direct” velocity bounds of the form

$$\dot{\mathbf{q}}^\top \mathbf{B}_v(\mathbf{q})\dot{\mathbf{q}} + \mathbf{f}_v(\mathbf{q}) \leq 0, \quad (2)$$

can also be taken into account, see footnote 3 and [20], [21].

Inequality (1) is very general and may represent a large variety of second-order systems and constraints. As an example, consider an n -dof manipulator with dynamics

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top \mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}. \quad (3)$$

Assume that the manipulator is subject to lower and upper bounds on the joint torques, that is, for any joint i and time t ,

$$\tau_i^{\min} \leq \tau_i(t) \leq \tau_i^{\max}.$$

Clearly, these torque bounds can be put in the form of (1) with

$$\mathbf{A}(\mathbf{q}) = \begin{pmatrix} \mathbf{M}(\mathbf{q}) \\ -\mathbf{M}(\mathbf{q}) \end{pmatrix}, \mathbf{B}(\mathbf{q}) = \begin{pmatrix} \mathbf{C}(\mathbf{q}) \\ -\mathbf{C}(\mathbf{q}) \end{pmatrix}, \mathbf{f}(\mathbf{q}) = \begin{pmatrix} \mathbf{g}(\mathbf{q}) - \boldsymbol{\tau}^{\max} \\ -\mathbf{g}(\mathbf{q}) + \boldsymbol{\tau}^{\min} \end{pmatrix},$$

where $\boldsymbol{\tau}^{\max} = (\tau_1^{\max}, \dots, \tau_n^{\max})^\top$ and $\boldsymbol{\tau}^{\min} = (\tau_1^{\min}, \dots, \tau_n^{\min})^\top$.

Consider now a path \mathcal{P} – represented as the underlying path of a trajectory $\mathbf{q}(s)_{s \in [0, s_{\text{end}}]}$ – in the configuration space. Assume that $\mathbf{q}(s)_{s \in [0, s_{\text{end}}]}$ is C^1 - and piecewise C^2 -continuous (note that how to generate such smooth initial trajectories, especially for closed kinematic chains, is an interesting problem on its own). We are interested in *time-parameterizations* of \mathcal{P} – or *time-reparameterizations* of $\mathbf{q}(s)_{s \in [0, s_{\text{end}}]}$ – which are increasing *scalar functions* $s : [0, T'] \rightarrow [0, s_{\text{end}}]$. Differentiating $\mathbf{q}(s(t))$ with respect to t yields

$$\dot{\mathbf{q}} = \mathbf{q}_s \dot{s}, \quad \ddot{\mathbf{q}} = \mathbf{q}_{ss} \dot{s}^2 + \mathbf{q}_s \ddot{s}, \quad (4)$$

where dots denote differentiations with respect to the time parameter t and $\mathbf{q}_s = \frac{d\mathbf{q}}{ds}$ and $\mathbf{q}_{ss} = \frac{d^2\mathbf{q}}{ds^2}$. Substituting (4) into (1) then leads to

$$\ddot{s} \mathbf{A}(\mathbf{q}) \mathbf{q}_s + \dot{s}^2 \mathbf{A}(\mathbf{q}) \mathbf{q}_{ss} + \dot{s}^2 \mathbf{q}_s^\top \mathbf{B}(\mathbf{q}) \mathbf{q}_s + \mathbf{f}(\mathbf{q}) \leq 0,$$

which can be rewritten as

$$\ddot{s} \mathbf{a}(s) + \dot{s}^2 \mathbf{b}(s) + \mathbf{c}(s) \leq 0, \quad \text{where} \quad (5)$$

$$\begin{aligned} \mathbf{a}(s) &= \mathbf{A}(\mathbf{q}(s)) \mathbf{q}_s(s), \\ \mathbf{b}(s) &= \mathbf{A}(\mathbf{q}(s)) \mathbf{q}_{ss}(s) + \mathbf{q}_s(s)^\top \mathbf{B}(\mathbf{q}(s)) \mathbf{q}_s(s), \\ \mathbf{c}(s) &= \mathbf{f}(\mathbf{q}(s)). \end{aligned} \quad (6)$$

Equation (5) constitutes another level of abstraction, which is particularly convenient for computer implementation: it suffices indeed to evaluate the M -dimensional vectors \mathbf{a} , \mathbf{b} and \mathbf{c} along the path and feed these vectors as inputs to the optimization algorithm. From this formulation, one can also remark that it is *not necessary* to evaluate the full matrices \mathbf{A} and tensors \mathbf{B} (which are of sizes $M \times n$ and $n \times M \times n$), but only their *products* (of sizes M) with \mathbf{q}_s and \mathbf{q}_{ss} . In the case of torque bounds, the Recursive Newton-Euler method for inverse dynamics [22] allows computing these products without ever evaluating the full \mathbf{A} and \mathbf{B} . Finally, this formulation allows very easily *combining* different types of constraints: for each s , it suffices to concatenate the vectors $\mathbf{a}(s)$ corresponding to the different constraints, and similarly for the vectors $\mathbf{b}(s)$ and $\mathbf{c}(s)$.

B. Review of the numerical integration approach

Each row i of equation (5) is of the form

$$a_i(s) \ddot{s} + b_i(s) \dot{s}^2 + c_i(s) \leq 0.$$

There are three cases:

- if $a_i(s) > 0$, then one has $\ddot{s} \leq \frac{-c_i(s) - b_i(s) \dot{s}^2}{a_i(s)}$. Define the *upper bound* $\beta_i = \frac{-c_i(s) - b_i(s) \dot{s}^2}{a_i(s)}$;
- if $a_i(s) < 0$, then one has $\ddot{s} \geq \frac{-c_i(s) - b_i(s) \dot{s}^2}{a_i(s)}$. Define the *lower bound* $\alpha_i = \frac{-c_i(s) - b_i(s) \dot{s}^2}{a_i(s)}$;
- if $a_i(s) = 0$, then s is a “zero-inertia” point [17], [10].

One then has a certain number of α_p and β_q . Their total number is $\leq M$, with equality when s is not a zero-inertia point. One can next define for each (s, \dot{s})

$$\alpha(s, \dot{s}) = \max_p \alpha_p(s, \dot{s}), \quad \beta(s, \dot{s}) = \min_q \beta_q(s, \dot{s}).$$

From the above transformations, one can conclude that $\mathbf{q}(s(t))_{t \in [0, T']}$ satisfies the constraints (1) if and only if

$$\forall t \in [0, T'] \quad \alpha(s(t), \dot{s}(t)) \leq \dot{s}(t) \leq \beta(s(t), \dot{s}(t)). \quad (7)$$

Note that $(s, \dot{s}) \mapsto (\dot{s}, \alpha(s, \dot{s}))$ and $(s, \dot{s}) \mapsto (\dot{s}, \beta(s, \dot{s}))$ can be viewed as two vector fields in the (s, \dot{s}) plane. One can integrate velocity profiles following the field $(\dot{s}, \alpha(s, \dot{s}))$ (from now on, α in short) to obtain *minimum acceleration* profiles (or α -profiles), or following the field β to obtain *maximum acceleration* profiles (or β -profiles).

Next, observe that if $\alpha(s, \dot{s}) > \beta(s, \dot{s})$ then, from (7), there is no possible value for \dot{s} . Thus, to be valid, every velocity profile must stay below the maximum velocity curve³ (MVC in short) defined by

$$\text{MVC}(s) = \begin{cases} \min\{\dot{s} \geq 0 : \alpha(s, \dot{s}) = \beta(s, \dot{s})\} & \text{if } \alpha(s, 0) \leq \beta(s, 0), \\ 0 & \text{if } \alpha(s, 0) > \beta(s, 0). \end{cases}$$

It was shown (see e.g. [10]) that the time-minimal velocity profile is obtained by a *bang-bang*-type control, i.e., whereby the optimal profile follows alternatively the β and α fields while always staying below the MVC. More precisely, the algorithm to find the time-optimal parameterization of \mathcal{P} starting and ending with the desired linear velocities v_{beg} and v_{end} is as follows (see Fig. 1):

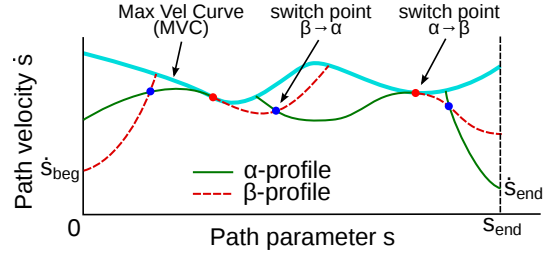


Fig. 1. MVC and α -, β -profiles in the numerical integration approach.

1. In the (s, \dot{s}) plane, start from $(s = 0, \dot{s} = v_{\text{beg}}/\|\mathbf{q}_s(0)\|)$ and integrate forward following β until hitting either

- the MVC, in this case go to step 2;
- the horizontal line $\dot{s} = 0$, in this case the path is not dynamically traversable;
- the vertical line $s = s_{\text{end}}$, in this case go to step 3.

2. Search forward along the MVC for the next candidate $\alpha \rightarrow \beta$ switch point (cf. Section II-C). From such a switch point:

- integrate *backward* following α , until *intersecting* a forward β -profile (from step 1 or recursively from the current step 2). The intersection point constitutes a $\beta \rightarrow \alpha$ switch point;
- integrate *forward* following β . Then continue as in step 1.

The resulting forward profile will be the concatenation of the intersected forward β -profile, the backward α -profile obtained in (a), and the forward β -profile obtained in (b).

3. Start from $(s = s_{\text{end}}, \dot{s} = v_{\text{end}}/\|\mathbf{q}_s(s_{\text{end}})\|)$ and integrate *backward* following α , until intersecting a forward profile obtained in steps 1 or 2. The intersection point constitutes a $\beta \rightarrow \alpha$ switch point. The final profile will be the concatenation of the intersected forward profile and the backward α -profile just computed.

From the above presentation, it appears that finding the $\alpha \rightarrow \beta$ switch points is crucial for the numerical integration approach. It was shown in [17], [18], [10] that a given point s is a $\alpha \rightarrow \beta$ switch point only in the following three cases:

³If “direct” velocity bounds such as in equation (2) are considered, then they induce another maximum velocity curve, noted $\text{MVC}_{\text{direct}}$. In this case, every velocity profile must stay below $\min(\text{MVC}, \text{MVC}_{\text{direct}})$. The treatment of $\text{MVC}_{\text{direct}}$ was initiated in [20], [21] and completed and implemented by us, see <https://github.com/quangounet/TOPP/releases>.

- the MVC is *discontinuous* at s . In this case s is labeled as a “discontinuous” switch point;
- the MVC is *continuous* but *undifferentiable* at s . In this case s is labeled as a “singular” switch point or a “dynamic singularity” (previous works labeled such switch points as “zero-inertia points” [18]; however we shall see below that not all zero-inertia points are singular);
- the MVC is *continuous* and *differentiable* at s and the *tangent vector* to the MVC at $(s, \text{MVC}(s))$ is collinear with the vector $(\text{MVC}(s), \alpha(s, \text{MVC}(s)))$ [or, which is the same since we are on the MVC, collinear with the vector $(\text{MVC}(s), \beta(s, \text{MVC}(s)))$]. In this case s is labeled as a “tangent” switch point.

Finding discontinuous and tangent switch points does not involve particular difficulties since it suffices to construct the MVC and examine whether it is discontinuous or whether the tangent to the MVC is collinear with α for all discretized points s along the path. Regarding the undifferentiable switch points, one approach could consist in checking whether the MVC is continuous but undifferentiable at s . However, this approach is seldom used in practice since it is comparatively more prone to discretization errors. Instead, it was proposed (cf. [17], [18], [10], [19], [11]) to equate undifferentiable points with *zero-inertia points*, i.e. the points s where $a_k(s) = 0$ for one of the constraints k , and to consequently search for zero-inertia points.

This method is however not completely correct: we shall see in Section II-C that *not all zero-inertia points are undifferentiable*. Thus, only zero-inertia points that are undifferentiable properly constitute *singular switch points* or *dynamic singularities*. Characterizing and addressing such switch points are of crucial importance since they occur in a large proportion of real-world TOPP instances – they are in particular much more frequent than discontinuous and tangent switch points together. Furthermore, since the α and β fields are *divergent* near these switch points (see Fig. 2), they are the cause of most failures in existing implementations of TOPP.

C. Characterizing dynamic singularities

Consider a zero-inertia point s^* , and assume that it is triggered by the k -th constraint, i.e. $a_k(s^*) = 0$. Without loss of generality, we make the assumption that $a_k(s) < 0$ in a neighborhood to the left of s^* and $a_k(s) > 0$ in a neighborhood to the right of s^* (the case when a_k switches from positive to negative can be treated similarly by changing signs at appropriate places).

We prove in the Appendix that, if the path is traversable, then $c_k(s^*) < 0$. We next distinguish two cases (see Fig. 2A).

Case 1: $b_k(s^*) > 0$. Define $\dot{s}^* = \sqrt{\frac{-c_k(s^*)}{b_k(s^*)}}$. Next, let \dot{s}^\dagger be the value of the MVC, had we removed constraint k . We prove in the Appendix that

- If $\dot{s}^\dagger < \dot{s}^*$, then constraint k does *not* trigger a dynamic singularity at s^* ;
- If $\dot{s}^\dagger > \dot{s}^*$, then the MVC is indeed undifferentiable at s^* , and s^* constitutes a dynamic singularity.

Case 2: $b_k(s^*) < 0$. We prove that, in this case, constraint k does *not* trigger a dynamic singularity at s^* .

D. Addressing dynamic singularities

1) *Previous treatments:* The next difficulty consists in the selection of the optimal acceleration to initiate the backward and forward integrations from a dynamic singularity s^* : indeed the fields α and β are not *naturally* defined at these points because of a division by $a_k(s^*) = 0$. In [17], no indication was given regarding this

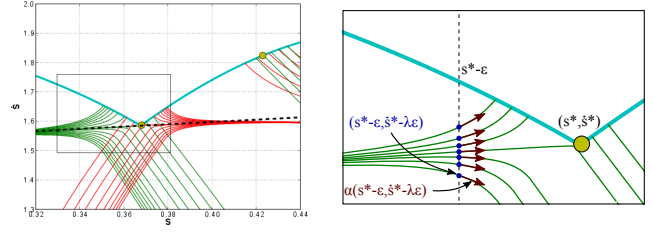


Fig. 2. **A:** α - and β -profiles (in green and red respectively) near zero-inertia points (yellow points). The left zero-inertia point is a singular switch point, while the right zero-inertia point is not singular. Note that in agreement with the definitions, at any point in the plane the slope of the red profile is higher than the slope of the green profile, except on the MVC where the two slopes are equal. The dotted line is the line that goes through the switch point and has slope λ computed by equation (10) (cf. Section II-D). **B:** close-up view (zoomed in the black box of **A**) centered around the singular switch point.

matter. In [18], it was stated that “[this] acceleration is not uniquely determined” and suggested to choose *any* acceleration to initiate the integrations. In [10] (and also in [19] which reproduced the reasoning of [10]), the authors suggested to select the following acceleration to initiate the backward integration (and a similar expression for the forward integration):

$$\min(\alpha^-, \alpha^+, \alpha_{\text{MVC}}), \text{ where} \quad (8)$$

$$\alpha^- = \lim_{s \uparrow s^*} \alpha(s, \text{MVC}(s^*)), \quad \alpha^+ = \lim_{s \downarrow s^*} \alpha(s, \text{MVC}(s^*)), \quad (9)$$

and α_{MVC} is computed from the *slope* of the MVC on the left of s^* .

However, observing the α -profiles near the dynamic singularity of Fig. 2A, it appears that the definition of α^- in equation (9) is arbitrary. Indeed, depending on the *direction* from which one moves towards $(s^*, \text{MVC}(s^*))$ in the (s, \dot{s}) plane, the limit of α is different: for instance, in Fig. 2A, if one moves from the top left, the limit, if it exists, would be positive, and it would be negative if one moves from the bottom left. In this context, the choice of equation (9) consisting in moving towards $(s^*, \text{MVC}(s^*))$ *horizontally* is no more justified than any other choice. More generally, it is *impossible* to extend α by continuity towards $(s^*, \text{MVC}(s^*))$ from the left because the α -profiles *diverge* when approaching $(s^*, \text{MVC}(s^*))$ from the left.

In practice, because of this flow divergence, choosing a slightly incorrect value for α and β at s^* may result in strong oscillations (see Fig. 5), which in turn can make the algorithm incorrectly terminate (because the the velocity profile would cross the MVC or the line $\dot{s} = 0$). In fact, this is probably one of the main reasons of failure in existing implementations.

2) *Proposed new treatment:* Fig. 2B shows in more detail the α -profiles near a singular switch point s^* .

We first show in the Appendix that, if the singularity is triggered by constraint k , then $\alpha = \alpha_k$ on the left of s^* and $\beta = \beta_k$ on the right of s^* . Consider next the intersections of the vertical line $s = s^* - \epsilon$, where ϵ is an arbitrary small positive number, with the α -profiles. An α -profile can reach (s^*, \dot{s}^*) only if its *tangent vector* at the intersection *points towards* (s^*, \dot{s}^*) . This can be achieved if there exists a real number λ such that

$$\frac{\alpha_k(s^* - \epsilon, \dot{s}^* - \lambda\epsilon)}{\dot{s}^* - \lambda\epsilon} = \lambda.$$

Replacing α_k by its expression yields the condition

$$\frac{-b_k(s^* - \epsilon)[\dot{s}^* - \lambda\epsilon]^2 - c_k(s^* - \epsilon)}{a_k(s^* - \epsilon)[\dot{s}^* - \lambda\epsilon]} = \lambda, \text{ i.e.}$$

$$-b_k(s^* - \epsilon)[\dot{s}^* - \lambda\epsilon]^2 - c_k(s^* - \epsilon) = \lambda a_k(s^* - \epsilon)[\dot{s}^* - \lambda\epsilon].$$

Computing the Taylor expansion of the above equation at order 1 in ϵ and recalling that $-b_k(s^*)\dot{s}^{*2} - c_k(s^*) = 0$ and $a_k(s^*) = 0$, one obtains the condition

$$2\lambda b_k(s^*)\dot{s}^* + b'_k(s^*)\dot{s}^{*2} + c'_k(s^*) = -\lambda a'_k(s^*)\dot{s}^*.$$

Solving for λ , one finally obtains

$$\lambda = -\frac{b'_k(s^*)\dot{s}^{*2} + c'_k(s^*)}{[2b_k(s^*) + a'_k(s^*)]\dot{s}^*}. \quad (10)$$

Following the same reasoning on the right of s^* , one has to solve

$$\frac{\beta_k(s^* + \epsilon, \dot{s}^* + \lambda\epsilon)}{\dot{s}^* + \lambda\epsilon} = -\lambda,$$

which leads to the same value as in equation (10). Thus the optimal backward and forward acceleration at (s^*, \dot{s}^*) is given by equation (10). One can observe in Fig. 2A that the black dotted line, whose slope is given by λ , indeed constitutes the “neutral” line at (s^*, \dot{s}^*) .

Based on the previous development, we propose the following algorithm when encountering a zero-inertia point s^* , with $a_k(s^*) < 0$ on the left of s^* and $a_k(s^*) > 0$ on the right of s^* :

▲ If $b_k(s^*) < 0$, then constraint k does not trigger a singularity;
 ▲ If $b_k(s^*) > 0$, then compute \dot{s}^* by equation (11) and \dot{s}^\dagger by removing constraint k and evaluating again the MVC at s^* .

- If $\dot{s}^* > \dot{s}^\dagger$, then constraint k does not trigger a singularity;
- If $\dot{s}^* < \dot{s}^\dagger$, then s^* is a dynamic singularity. Next, compute λ by equation (10) and
 - integrate the constant field $(\dot{s}^*, \lambda\dot{s}^*)$ *backward* for a small number of time steps. Then continue by following α , as in the original algorithm;
 - integrate the constant field $(\dot{s}^*, \lambda\dot{s}^*)$ *forward* for a small number of time steps. Then continue by following β .

Note that after moving a small number of steps away from s^* , the fields α and β become smooth, so that there is no problem in the integration.

3) *About Kunz and Stilman’s conjecture:* Kunz and Stilman [11] were first to remark – in the particular case of TOPP with velocity and acceleration bounds and paths made of straight segments and circular arcs – that the algorithm proposed in [10] could not satisfactorily address dynamic singularities. They conjectured instead that the correct acceleration at the singularity is 0.

From equation (14) of [11], the correspondences between the parameters of [11] and those of the present article are given in Table I.

TABLE I
PARAMETERS CORRESPONDENCES

This article		Kunz and Stilman [11]
$a_k(s)$	\leftrightarrow	$f'_k(s)$
$b_k(s)$	\leftrightarrow	$f''_k(s)$
$c_k(s)$	\leftrightarrow	$-\ddot{q}_k^{\max}$ or \ddot{q}_k^{\max}

Remark that the zero-inertia points in [11] are all located in the circular portions. In such portions, the coefficients a_k and b_k have the following form (using our notations):

$$\begin{aligned} a_k(s) &= -\frac{C_1}{r} \sin\left(\frac{s}{r}\right) + \frac{C_2}{r} \cos\left(\frac{s}{r}\right), \\ b_k(s) &= \frac{C_1}{r^2} \cos\left(\frac{s}{r}\right) - \frac{C_2}{r^2} \sin\left(\frac{s}{r}\right), \end{aligned}$$

where r , C_1 and C_2 are three constants independent of s in a

neighborhood around s^* . Differentiating b_k next yields

$$b'_k(s) = -\frac{C_1}{r^3} \sin\left(\frac{s}{r}\right) + \frac{C_2}{r^3} \cos\left(\frac{s}{r}\right) = \frac{1}{r^2} a_k(s).$$

One thus has $b'_k(s^*) = 1/r^2 a_k(s^*) = 0$ at a zero-inertia point. If this zero-inertia point is actually a dynamic singularity then, from equation (10), one obtains that $\lambda = 0$, which proves Kunz and Stilman’s conjecture.

III. IMPLEMENTATION AND EVALUATION

A. Open-source implementation

We provide an implementation of TOPP in C++ that integrates the developments of Section II. We also provide an interface in Python for an easy and interactive use. Currently, our implementation supports pure velocity and acceleration bounds, torque bounds, ZMP constraints, multi-contact friction constraints, and any combination thereof. The dynamics computations are handled by OpenRAVE [23]. Thanks to the general formulation of Section II-A, new system dynamics and constraints can be easily added. The implementation and test cases are open-source and available at <https://github.com/quangounet/TOPP/releases>.

Fig. 3 illustrates the utilization of TOPP in a multi-contact task where a humanoid robot (HRP2, 36 dofs including the 6 coordinates of the free-flyer) steps down from a 30 cm high podium. Velocity and torque bounds were considered for each joint, as well as friction cone constraints on the left foot and the right hand (131 inequality constraints in total). The original trajectory had duration 1 s and the grid size was $N = 100$. The time parameterization part (excluding the dynamics computations and the constraints projection step [15]) took 0.005 s on our computer (Intel Core i5 3.2GHz, 3.8GB memory, GNU/Linux).

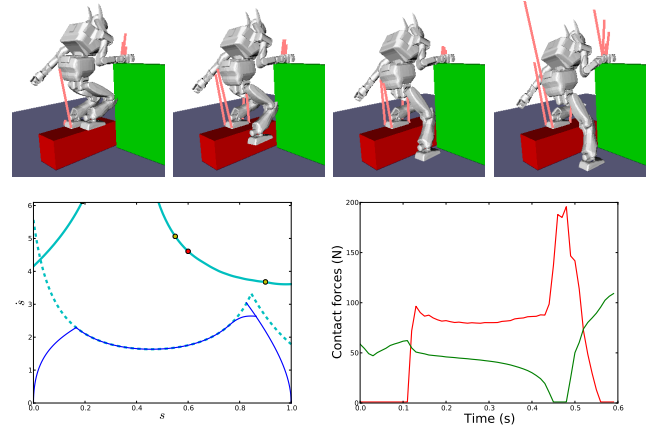


Fig. 3. TOPP with velocity, torque, and friction constraints in a multi-contact task with the HRP2 robot. **Top:** snapshots of the time-parameterized trajectory taken at equal time intervals. The pink lines represent the contact forces. **Left:** (s, \dot{s}) space. Same legends as in Fig. 2. The solid and dotted bold cyan lines are velocity limits that are imposed respectively by the torque and friction constraints [the MVC computed from equation (1)] and by the “direct” velocity constraints [computed from equation (2)]. The superimposed dotted blue line represents the final (s, \dot{s}) profile, which for some parts followed the α - and β -profiles and for some other parts “slid” along the “direct” velocity limit. **Right:** normal components of the reaction forces at the front left corner of the left foot (red) and the front left corner of the right hand (green). The normal components were constrained to be ≥ 1 N. Note how this constraint was saturated at the foot and hand contact points at different moments in time.

B. Comparison with previous treatments of dynamic singularities

For this, we tested the algorithm on a model of the 7-dof Barrett WAM. The velocity and torque bounds are those given by the

constructor. Fig. 4 shows a smooth behavior around the dynamic singularity, for both the (s, \dot{s}) profile and the torque profiles.

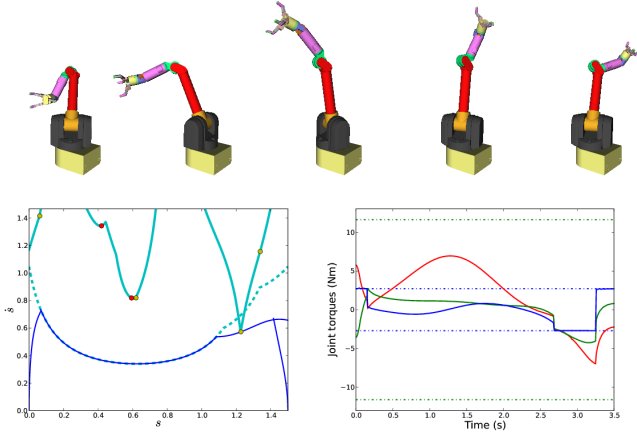


Fig. 4. TOPP with velocity and torque constraints for the 7-dof Barrett WAM. **Top:** snapshots of the time-parameterized trajectory taken at equal time intervals. **Left:** (s, \dot{s}) space. Same legends as in Fig. 3. **Right:** torque profiles for shoulder roll (solid red), wrist yaw (solid green) and wrist roll (solid blue). The dotted lines represent the torque bounds in corresponding colors.

Next, to demonstrate more clearly the improvements permitted by our algorithm, we compared the results given by our algorithm and that given by the algorithm of [10], which incorrectly proposes to “slide” along the MVC near dynamic singularities. Fig. 5 shows that using the correct acceleration values significantly decreases the jitters around the dynamic singularity, even at a coarse discretization time step.

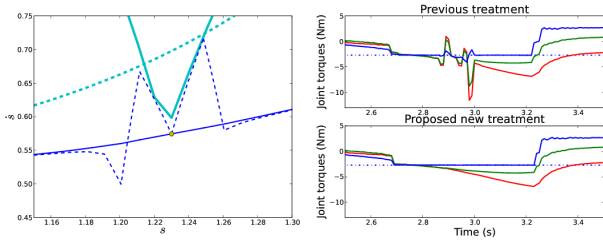


Fig. 5. Close-up views of the (s, \dot{s}) profiles and the torque profiles near a dynamic singularity. The computations were done at a time step of 0.01 s. **Left:** (s, \dot{s}) profiles. Dotted lines: profile computed using the method of [10]. Solid lines: profile computed using our proposed new method. **Right, top:** torque profiles corresponding to the method of [10]. **Right, bottom:** torque profiles corresponding to our method. Note that our method allows suppressing the jitters even at a coarse time step.

Finally, we tested the overall robustness of the implementation in two settings: the 7-dof WAM with velocity and torque bounds as above, and a 7-dof kinematic system with velocity and acceleration bounds (set to respectively $4 \text{ rad}\cdot\text{s}^{-1}$ and $20 \text{ rad}\cdot\text{s}^{-2}$, which are typical values for industrial manipulators). In both settings, we tested 1000 random trajectories of duration 1 s and grid size $N = 200$. The initial trajectories were Bezier curves that interpolate between two random points in $[-\pi, \pi]^7$. Table II reports the number of failures – which occur when the profiles do not cover the whole segment $[0, s_{\text{end}}]$ [11] – as well as the number of dynamic singularities encountered in the 1000 instances.

TABLE II
TESTING THE IMPLEMENTATION ON 1000 TRAJECTORIES

System	# failures	# singularities
WAM torque constraints	0	307
Kinematic constraints	1	539

C. Comparison with the convex optimization approach

We first conducted an “informal” comparison of our implementation with the MATLAB-based implementations of Verschuer et al [7] and of Debrouwere et al [9] on the problem of torque bounds. Note that, for the convex optimization part itself, these implementations used a library, YALMIP, which in turn calls an external solver (SeDuMi in [7]), written in C, C++ or Fortran and precompiled as binary MEX files. Thus, the comparison with our library written in C++, although “informal”, is not completely unfounded.

Table III gives the convex optimization solver times (which thus do not include the robot dynamics computation times) reported by [7], [9] as well as the running times of our implementation after subtracting the robot dynamics computation times. Different grid sizes (N) and number of degrees of freedom to optimize (DOF) were reported or tested.

TABLE III
INFORMAL COMPARISON, TORQUE CONSTRAINTS

Source	Lang.	DOF	N	Exec.
Verschuer et al 2009 [7]	MEX	3	299	0.74 s
Verschuer et al 2009 [7]	MEX	3	1999	2.87 s
Debrouwere et al 2013 [9]	MEX	7	100	1.5 s
This article	C++	3	300	0.017 s
This article	C++	3	2000	0.10 s
This article	C++	7	100	0.0058 s

Next, in order to make a “formal” comparison – on the same computer and using the same programming language, we considered MINTOS (<http://www.iu.edu/~motion/mintos/>, last accessed December 2013), Hauser’s recent C++ implementation of the convex optimization approach [8], which is, to our knowledge, the fastest implementation currently available. To exclude the robot dynamics computations – which are independent of the TOPP problem and whose execution times depend largely on the robot simulation software used, we considered “pure” velocity and acceleration bounds. Note however that, from the general formulation of Section II-A, these pure kinematic constraints involve exactly the same difficulties as any other type of dynamic constraints.

We compiled and ran both implementations on our computer. We considered 1000 random trajectories of dimension 7 and velocity and acceleration bounds as in Section III-B. Table IV reports the average total execution times of the two implementations. It appears from this controlled comparison that our implementation is between 7 and 15 times faster than MINTOS⁴.

IV. CONCLUSION

We have established a rigorous characterization and treatment of dynamic singularities that arise in the numerical integration approach to the Time-Optimal Path Parameterization (TOPP) problem. This fully completes the celebrated line of research on TOPP, which started

⁴Note that MINTOS includes an innovative “constraints pruning” step that significantly speeds up the execution with respect to previous implementations of the convex optimization approach. We are investigating how this idea can be integrated in the numerical integration approach.

TABLE IV
FORMAL COMPARISON, KINEMATIC CONSTRAINTS

Source	Lang.	DOF	N	Exec.
Hauser 2013 [8]	C++	7	300	0.025 s
Hauser 2013 [8]	C++	7	1000	0.20 s
This article	C++	7	300	0.0038 s
This article	C++	7	1000	0.014 s

in the 1980's with the seminal papers of Bobrow et al. [13] and Shin and McKay [6] and which has since then received contributions from many prominent research groups (e.g. [17], [18], [10], [11], just to cite a few).

Based on that contribution and on a general formulation of the TOPP problem, we provided an open-source implementation of TOPP. We showed that this implementation is robust and fast: on typical test cases, it is about one order of magnitude faster than the fastest currently available implementation of the convex optimization approach. As our implementation is open-source and has been designed so as to facilitate the integration of new systems dynamics and constraints, we hope that it will be useful to robotics researchers interested in kinodynamic motion planning.

Our next immediate goal is to attack the *feasibility* problem (i.e. finding a collision-free, dynamically-balanced trajectory in a challenging context, in particular, where quasi-static trajectories are impossible) for humanoid robots using Admissible Velocity Propagation (AVP) [5], which itself is based on TOPP. We believe that the power of AVP combined with the speed of the present TOPP implementation can make possible the planning of unprecedentedly dynamic motions for humanoid robots in challenging environments.

Finally, from a theoretical perspective, we are investigating how higher-order constraints, such as jerk [24], [25] or other types of optimization objectives [7], [9], can be integrated in our TOPP framework.

Acknowledgments

We would like to thank Stéphane Caron, Zvi Shiller and Yoshihiko Nakamura for stimulating discussions regarding TOPP. We are also grateful to S. C. and Rosen Diankov for their helps with the implementation. This work was supported by "Grants-in-Aid for Scientific Research" for JSPS fellows and by a JSPS postdoctoral fellowship.

APPENDIX

A. Characterizing dynamic singularities

In line with Section II-C, consider a zero-inertia point s^* and assume that $a_k(s^*) = 0$ and that $a_k(s) < 0$ in a neighborhood to the left of s^* and $a_k(s) > 0$ in a neighborhood to the right of s^* . We have the following definition and proposition.

Proposition 1. *Define, for all (s, \dot{s}) ,*

$$\tilde{\alpha}(s, \dot{s}) = \max_{i \neq k} \alpha_i(s, \dot{s}) ; \quad \tilde{\beta}(s, \dot{s}) = \min_{i \neq k} \beta_i(s, \dot{s}).$$

There exists a neighborhood $]s^ - \epsilon, s^*[$ to the left of s^* such that*

$$\forall (s, \dot{s}) \in]s^* - \epsilon, s^*[\times[0, \infty[, \quad \beta(s, \dot{s}) = \tilde{\beta}(s, \dot{s}).$$

and a neighborhood $]s^, s^* + \epsilon[$ to the right of s^* such that*

$$\forall (s, \dot{s}) \in]s^*, s^* + \epsilon[\times[0, \infty[, \quad \alpha(s, \dot{s}) = \tilde{\alpha}(s, \dot{s}),$$

Note also that $\tilde{\alpha}$ and $\tilde{\beta}$ are continuous and differentiable in a neighborhood around s^ .*

Proof: From our assumption that $a_k(s) < 0$ on the left of s^* and $a_k(s) > 0$ on the right of s^* , constraint k gives rise to an α_k on the left of s^* and to a β_k on the right of s^* . It thus does not contribute to the value of β on the left of s^* or to that of α on the right of s^* \square

Proposition 2. *If $c_k(s^*) > 0$ then there exists a neighborhood $]s^* - \epsilon, s^*]$ such that $MVC(s) = 0$ for all $s \in]s^* - \epsilon, s^*]$ (which in turn implies that the path is not traversable).*

Proof: Suppose that $c_k(s^*) = \eta > 0$. By continuity of c_k , there exists a neighborhood $]s^* - \epsilon_1, s^*]$ such that

$$\forall s \in]s^* - \epsilon_1, s^*], \quad -c_k(s) < -\eta/2.$$

On the other hand, one has $a_k(s) \uparrow 0$ when $s \uparrow s^*$. Thus, $\alpha_k(s, 0) = \frac{-c_k(s)}{a_k(s)} \rightarrow \infty$ when $s \uparrow s^*$. Since $\alpha = \max_i \alpha_i$, we have that $\alpha(s, 0) \rightarrow +\infty$ when $s \uparrow s^*$. Next, from Proposition 1, β is continuous, hence *upper-bounded*, in a neighborhood to the left of s^* . Thus, there exists a neighborhood to the left of s^* in which $\alpha(s, 0) > \beta(s, 0)$, which in turn implies that $MVC(s) = 0$ in that neighborhood \square

In light of Proposition 2, we assume from now on that $c_k(s^*) < 0$. We next distinguish two cases according to the value of $b_k(s^*)$.

Case $b_k(s^*) > 0$: Define

$$\dot{s}^* = \sqrt{\frac{-c_k(s^*)}{b_k(s^*)}}. \quad (11)$$

Note that, since $c_k(s^*) < 0$ and $b_k(s^*) > 0$, the expression under the radical sign is indeed positive. Next, let \dot{s}^\dagger be the smallest velocity \dot{s} that satisfies $\tilde{\alpha}(s^*, \dot{s}) = \tilde{\beta}(s^*, \dot{s})$ ($\dot{s}^\dagger = +\infty$ if no such \dot{s} exists).

We now distinguish two sub-cases.

Sub-case $\dot{s}^\dagger < \dot{s}^*$: Let $\dot{s}^\ddagger = (\dot{s}^\dagger + \dot{s}^*)/2$. By the definition of \dot{s}^* and the assumption that $b_k(s^*) > 0$, there exists $\eta > 0$ such that, in a neighborhood to the left of s^*

$$\forall \dot{s} \leq \dot{s}^\ddagger, \quad -b_k(s)\dot{s}^2 - c_k(s) > \eta.$$

On the other hand, one has $a_k(s) \uparrow 0$ when $s \uparrow s^*$. Thus, $\alpha_k(s, \dot{s}) = \frac{-b_k(s)\dot{s}^2 - c_k(s)}{a_k(s)} \rightarrow -\infty$ when $s \uparrow s^*$ and $\dot{s} \leq \dot{s}^\ddagger$. As a consequence, constraint k does *not* contribute to α in a neighborhood to the left of s^* and for $\dot{s} \leq \dot{s}^\ddagger$. Thus, one has $\alpha = \tilde{\alpha}$ in a neighborhood to the left of s^* and for $\dot{s} \leq \dot{s}^\ddagger$.

By the same argument, one can show that $\beta = \tilde{\beta}$ in a neighborhood to the right of s^* and for $\dot{s} \leq \dot{s}^\ddagger$. Combined with Proposition 1, one has thus obtained that $\alpha = \tilde{\alpha}$ and $\beta = \tilde{\beta}$ in a neighborhood around s^* and for $\dot{s} \leq \dot{s}^\ddagger$. This shows that $MVC(s^*) = \dot{s}^\ddagger$, and that the MVC is entirely determined by $\tilde{\alpha}$ and $\tilde{\beta}$ around (s^*, \dot{s}^\ddagger) . One can thus conclude that the MVC is continuous and differentiable at s^* , which in turn implies that constraint k does *not* trigger a singularity at s^* .

Sub-case $\dot{s}^\dagger > \dot{s}^*$: Remark first that, excepting degenerate cases, one can find a neighborhood $]s^* - \epsilon, s^* + \epsilon[\times[\dot{s}^* - \eta, \dot{s}^* + \eta]$ in which $\tilde{\alpha}$ is given by a unique α_q and $\tilde{\beta}$ is given by a unique β_p . Note that, by definition of $\tilde{\alpha}$ and $\tilde{\beta}$, one has $p \neq k$ and $q \neq k$.

In the neighborhood just defined, let

$$u(s) = \frac{-a_k(s)c_q(s) + a_q(s)c_k(s)}{a_k(s)b_q(s) - a_q(s)b_k(s)};$$

$$v(s) = \frac{-a_k(s)c_p(s) + a_p(s)c_k(s)}{a_k(s)b_p(s) - a_p(s)b_k(s)}.$$

From the assumption that $a_k(s^*) = 0$, one has

$$\lim_{s \uparrow s^*} u(s) = \frac{-c_k(s^*)}{b_k(s^*)} = \dot{s}^{*2}.$$

Thus, in a neighborhood to the left of s^* , one has $0 < u(s) < \dot{s}^{\dagger 2}$. Next, remark that by definition $\dot{s} = \sqrt{u(s)}$ satisfies $\alpha_k(s, \dot{s}) = \beta_q(s, \dot{s})$. The above two statements together imply that $MVC(s) = \sqrt{u(s)}$.

One can show similarly that there exists a neighborhood to the right of s^* in which $MVC(s) = \sqrt{v(s)}$. Combining the results concerning the left and the right of s^* , one obtains that the MVC is *continuous* at s^* , since

$$\lim_{s \uparrow s^*} MVC(s) = \lim_{s \uparrow s^*} \sqrt{u(s)} = \dot{s}^* = \lim_{s \downarrow s^*} \sqrt{v(s)} = \lim_{s \downarrow s^*} MVC(s).$$

However, the MVC is *undifferentiable* at s^* since, in general,

$$\lim_{s \uparrow s^*} MVC'(s) = \lim_{s \uparrow s^*} (\sqrt{u(s)})' \neq \lim_{s \downarrow s^*} (\sqrt{v(s)})' = \lim_{s \downarrow s^*} MVC'(s).$$

Thus, in this sub-case, s^* is indeed a dynamic singularity.

Case $b_k(s^*) < 0$: From the assumptions that $c_k(s^*) < 0$ and $b_k(s^*) < 0$, one has that $-b_k(s^*)\dot{s}^2 - c_k(s^*) > 0$ for all \dot{s} . Thus, by the same argument as in sub-case $\dot{s}^\dagger < \dot{s}^*$, there exists a neighborhood $]s^* - \epsilon, s^*[$ where $\alpha = \tilde{\alpha}$ and a neighborhood $]s^*, s^* + \epsilon[$ where $\beta = \tilde{\beta}$. One can thus conclude that constraint k does *not* trigger a singularity at s^* .

REFERENCES

- [1] S. LaValle, *Planning algorithms*. Cambridge Univ Press, 2006.
- [2] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.
- [3] J. Bobrow, "Optimal robot plant planning using the minimum-time criterion," *IEEE Journal of Robotics and Automation*, vol. 4, no. 4, pp. 443–450, 1988.
- [4] Z. Shiller and S. Dubowsky, "On computing the global time-optimal motions of robotic manipulators in the presence of obstacles," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 785–797, 1991.
- [5] Q.-C. Pham, S. Caron, and Y. Nakamura, "Kinodynamic planning in the configuration space via velocity interval propagation," in *Robotics: Science and System*, 2013.
- [6] K. Shin and N. McKay, "Selection of near-minimum time geometric paths for robotic manipulators," *IEEE Transactions on Automatic Control*, vol. 31, no. 6, pp. 501–511, 1986.
- [7] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009.
- [8] K. Hauser, "Fast interpolation and time-optimization on implicit contact submanifolds," in *Robotics: Science and Systems*, 2013.
- [9] F. Debrouwere, W. Van Loock, G. Pipeleers, Q. Tran Dinh, M. Diehl, J. De Schutter, and J. Swevers, "Time-optimal path following for robots with convex-concave constraints using sequential convex programming," *IEEE Transactions on Robotics*, 2013.
- [10] Z. Shiller and H. Lu, "Computation of path constrained time optimal motions with dynamic singularities," *Journal of dynamic systems, measurement, and control*, vol. 114, p. 34, 1992.
- [11] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Robotics: Science and Systems*, vol. 8, 2012, pp. 09–13.
- [12] Q.-C. Pham, "Characterizing and addressing dynamic singularities in the time-optimal path parameterization algorithm," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.
- [13] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.
- [14] Q.-C. Pham and Y. Nakamura, "Time-optimal path parameterization for critically dynamic motions of humanoid robots," in *IEEE-RAS International Conference on Humanoid Robots*, 2012.
- [15] K. Hauser, "Fast dynamic optimization of robot paths under actuator limits and frictional contact," in *Robotics and Automation, IEEE International Conference on*, 2014.
- [16] F. Bullo and K. M. Lynch, "Kinematic controllability for decoupled trajectory planning in underactuated mechanical systems," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 402–412, 2001.
- [17] F. Pfeiffer and R. Johanni, "A concept for manipulator trajectory planning," *IEEE Journal of Robotics and Automation*, vol. 3, no. 2, pp. 115–123, 1987.
- [18] J. Slotine and H. Yang, "Improving the efficiency of time-optimal path-following algorithms," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 118–124, 1989.
- [19] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.
- [20] L. Zlajpah, "On time optimal path control of manipulators with bounded joint velocities and torques," in *IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 1996, pp. 1572–1577.
- [21] F. Lamiroux and J.-P. Laumond, "From paths to trajectories for multi-body mobile robots," in *Experimental Robotics V*. Springer, 1998, pp. 301–309.
- [22] M. Walker and D. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *Journal of Dynamic Systems, Measurement, and Control*, vol. 104, p. 205, 1982.
- [23] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010. [Online]. Available: http://www.programmingvision.com/rosen_diankov_thesis.pdf
- [24] M. Tarkainen and Z. Shiller, "Time optimal motions of manipulators with actuator dynamics," in *IEEE International Conference on Robotics and Automation*. IEEE, 1993, pp. 725–730.
- [25] D. Constantinescu and E. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of Robotic Systems*, vol. 17, no. 5, pp. 233–249, 2000.