Planning Truly Dynamic Motions:  
Path-Velocity Decomposition Revisited

Quang-Cuong Pham\textsuperscript{a}, Stéphane Caron\textsuperscript{b},  
Puttichai Lertkultanon\textsuperscript{a}, and Yoshihiko Nakamura\textsuperscript{b}

\textsuperscript{a}School of Mechanical and Aerospace Engineering, NTU, Singapore  
\textsuperscript{b}Department of Mechano-Informatics, University of Tokyo, Japan

November 18, 2014

Abstract

Path-velocity decomposition is an intuitive yet powerful approach to address the complexity of kinodynamic motion planning. The difficult trajectory planning problem is solved in two separate steps: first, find a path in the configuration space that satisfies the geometric constraints, and second, find a time-parameterization of that path satisfying the kinodynamic constraints. A fundamental requirement here is that the path found in the first step should be time-parameterizable. Most existing works fulfill this requirement by enforcing quasi-static constraints in the path planning step, resulting in an important loss in completeness. We propose a method that enables path-velocity decomposition to discover truly dynamic motions, i.e. motions that are not quasi-statically executable. At the heart of the proposed method is a new algorithm – Admissible Velocity Propagation – which, given a path and an interval of reachable velocities at the beginning of that path, computes exactly and efficiently the interval of all the velocities the system can reach after traversing the path while respecting the system kinodynamic constraints. Combining this algorithm with usual sampling-based planners then gives rise to a family of new trajectory planners that can appropriately handle kinodynamic constraints while retaining the advantages associated with path-velocity decomposition. We demonstrate the efficiency of the proposed method on some difficult kinodynamic planning problems, where, in particular, quasi-static methods are guaranteed to fail\footnote{This paper is a substantially revised and expanded version of Pham et al. [2013], which was presented at the conference Robotics: Science and Systems, 2013.}

1 Introduction

Planning motions for robots with many degrees of freedom and subject to kinodynamic constraints (i.e. constraints that involve higher-order time-derivatives of the robot configuration [Donald et al. 1993, LaValle and Kuffner 2001]) is one of the most important and challenging problems in robotics. Path-velocity decomposition is an intuitive yet powerful approach to address the complexity of kinodynamic motion planning: first,
find a path in the configuration space that satisfies the geometric constraints (such as obstacle avoidance, joint limits, kinematic closure, etc.), and second, find a time-parameterization of that path satisfying the kinodynamic constraints (such as torque limits for manipulators, dynamic balance for legged robots, etc.)

**Advantages of path-velocity decomposition** This approach was suggested as early as 1986 – only a few years after the birth of motion planning itself as a research field – by Kant and Zucker in the context of motion planning amongst movable obstacles. Since then, it has become a valuable tool to address many kinodynamic planning problems, from manipulators subject to torque limits [Bobrow et al., 1985, Shin and McKay, 1986, Bobrow, 1988], to coordination of teams of mobile robots [Siméon et al., 2002, Peng and Akella, 2005], to legged robots subject to balance constraints [Kuffner et al., 2002, Suleiman et al., 2010, Hauser et al., 2008, Pham and Nakamura, 2012, Escande et al., 2013, Hauser, 2014, etc.]

Path-velocity decomposition is appealing in that it exploits the natural decomposition of the constraints, in most systems, into two categories: those depending uniquely on the robot configuration, and those depending in particular on the velocity, which in turn is related to the energy of the system. Consider for instance a humanoid robot in a multi-contact task. Such a robot must (1) avoid collision with the environment, (2) avoid self-collisions, (3) respect kinematic closure for the parts in contact with the environment (e.g. the stance foot must be fixed with respect to the ground), (4) maintain balance. It can be noted that constraints (1 – 3) are exclusively related to the configuration of the robot, while constraint (4), once a path is given, depends mostly on the path velocity.

From a practical viewpoint, the two sub-problems – geometric path planning and kinodynamic time-parameterization – have received so much attention from the robotics community in the past three decades that a large body of theory and good practices exist and can be readily combined to yield efficient trajectory planners. Briefly, high-dimensional and cluttered geometric path planning problems can now be solved in seconds thanks to sampling-based planning algorithms such as PRM [Kavraki et al., 1996] or RRT [Kuffner and LaValle, 2000] and to the dozens of heuristics that have been developed for these algorithms. Regarding kinodynamic time-parameterization, two important discoveries about the structure of the problem have led to particularly efficient algorithmic solutions. First, the bang-bang nature of the optimal velocity profile was identified by [Bobrow et al., 1985, Shin and McKay, 1986], leading to fast numerical integration methods [see Pham, 2014 for extensive historical references]. Second, this problem was shown to be reducible to a convex optimization problem, leading to robust and versatile convex-optimization-based solutions [see e.g. Verscheure et al., 2009, Hauser, 2014].

**Problems with state-space planning and trajectory optimization approaches** Alternative approaches to path-velocity decomposition include planning directly in the state space and trajectory optimization. The first approach deploys traditional path planners such as RRT [LaValle and Kuffner, 2001] or PRM [Hsu et al., 2002] directly into the state space, that is, the configuration space augmented with velocity coordinates. Three main difficulties are associated with this approach. First, the dimension of the state space is twice that of the configuration space, resulting in higher algorith-
mic complexity. Second, while connecting two adjacent configurations under geometric constraints is trivial (using e.g. linear segments), connecting two adjacent states under kinodynamic constraints is considerably more challenging and time-consuming, requiring e.g. to solve a two-point boundary value problem [LaValle and Kuffner, 2001]. Third, especially for state-space RRTs, designing a reasonable metric is particularly difficult: Perez et al. [2012] showed that, even for the 1-dof pendulum subject to torque constraints, a state-space RRT with a simple Euclidean metric is doomed to failure. The authors then proposed to construct an efficient metric by locally solving an optimal control problem. While this method can address the case of the 1-dof pendulum, the necessity to solve an optimal control problem of the dimension of the system at each tree extension step makes it unlikely to scale to higher dimensions. For these reasons, in spite of appealing completeness guarantees [under some precise conditions, see Caron et al., 2014], there exists, to our knowledge, no example of successful application of state-space planning to high-dimensional systems with complex nonlinear dynamics and constraints.

The second approach, trajectory optimization, starts with an initial trajectory, which may not be valid (for example the trajectory may not reach the goal configuration, the robot may collide with the environment or may lose balance at some time instants, etc.) One then iteratively modifies the trajectory so as to decrease a cost – which encodes in particular how much the constraints are violated – until it falls below a certain threshold, implying in turn that the trajectory reaches the goal and all constraints are satisfied. Many interesting variations exist: the iterative modification step may be deterministic [Ratliff et al., 2009] or stochastic [Kalakrishnan et al., 2011], the optimization may be done through contact [Mordatch et al., 2012; Posa and Tedrake, 2013], etc. However, for long time-horizon and high-dimensional systems, this approach requires solving a large nonlinear optimization problem, which is computationally challenging because of the huge problem size and the existence of many local minima [see Hauser, 2014, for an extensive discussion of the advantages and limitations of trajectory optimization and comparison with path-velocity decomposition].

The quasi-static condition and its limitations Coming back to path-velocity decomposition, a fundamental requirement here is that the path found in the first step must be time-parameterizable. A commonly-used method to fulfill this requirement is to consider, in that step, the quasi-static constraints that are derived from the original kinodynamic constraints by assuming that the motion is executed at zero velocity. Indeed, the so-derived quasi-static constraints can be expressed using only configuration-space variables, in such a way that planning with quasi-static constraints is purely a geometric path planning problem. In the context of legged robots for example, the balance of the robot at zero velocity is guaranteed when the projection of the center of gravity lies in the support area – a purely geometric condition. This quasi-static condition is assumed in most works dedicated to the planning of complex humanoid motions [see e.g. Kuffner et al., 2002].

This workaround suffers however from a major limitation: the quasi-static condition may be too restrictive and one thus may overlook many possible solutions, i.e. incurring an important loss in completeness. For instance, legged robots walking with ZMP-based control [Vukobratovic et al., 2001] are dynamically balanced but almost never satisfy
Planning truly dynamic motions  Here we propose a method to overcome this limitation. At the heart of the proposed method is a new algorithm – Admissible Velocity Propagation (AVP) – which is based in turn on the classical Time-Optimal Path Parameterization (TOPP) algorithm first introduced by Bobrow et al. [1985, Shin and McKay [1986] and later perfected by many others [see Pham, 2014, and references therein]. In contrast with TOPP, which determines one optimal velocity profile along a given path, AVP addresses all valid velocity profiles along that path, requiring only slightly more computation time than TOPP itself. Combining AVP with usual sampling-based path planners, such as RRT, gives rise to a family of new trajectory planners that can appropriately handle kinodynamic constraints while retaining the advantages associated with path-velocity decomposition.

The remainder of this article is organized as follows. In Section 2, we briefly recall the fundamentals of TOPP before presenting AVP. In Section 3, we show how to combine AVP with usual sampling-based path planners such as RRT. In Section 4, we demonstrate the efficiency of the new AVP-based planners on some challenging kinodynamic planning problems – in particular, those where the quasi-static approach is guaranteed to fail. Finally, in Section 5, we discuss the advantages and limitations of the proposed approach and sketch some future research directions.

2  Propagating admissible velocities along a path

2.1  Background : Time-Optimal Path Parameterization (TOPP)

As mentioned in the Introduction, there are two main approaches to TOPP: “numerical integration” and “convex optimization”. We briefly recall the numerical integration approach [Bobrow et al., 1985, Shin and McKay, 1986], on which AVP is based. For more details about this approach, the reader is referred to Pham [2014].

Let \( \mathbf{q} \) be an \( n \)-dimensional vector representing the configuration of a robot system. Consider second-order inequality constraints of the form [Pham, 2014]

\[
A(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^\top B(\mathbf{q})\dot{\mathbf{q}} + f(\mathbf{q}) \leq 0, \tag{1}
\]

where \( A(\mathbf{q}), B(\mathbf{q}) \) and \( f(\mathbf{q}) \) are respectively an \( M \times n \) matrix, an \( n \times M \times n \) tensor and an \( M \)-dimensional vector. Inequality (1) is general and may represent a large variety of second-order systems and constraints, such as manipulators subject to velocity, acceleration or torque limits, mobile robots subject to friction constraints, legged robots subject to balance constraints in multi-contact, etc.

Note that “direct” velocity bounds of the form

\[
\dot{\mathbf{q}}^\top B_v(\mathbf{q})\dot{\mathbf{q}} + f_v(\mathbf{q}) \leq 0, \tag{2}
\]
can also be taken into account Zlajpah [1996]. In the following discussion, we shall not, however, consider such “direct” velocity bounds for simplicity, although they are implemented and used in the applications.

Consider now a path $\mathcal{P}$ in the configuration space, represented as the underlying path of a trajectory $q(s)_{s \in [0,s_{\text{end}}]}$. Assume that $q(s)_{s \in [0,s_{\text{end}}]}$ is $C^1$- and piecewise $C^2$-continuous.

**Definition 1** A time-parameterization of $\mathcal{P}$—or time-reparameterization of $q(s)_{s \in [0,s_{\text{end}}]}$—is an increasing scalar function $s : [0,T'] \rightarrow [0,s_{\text{end}}]$. A time-parameterization can be seen alternatively as a velocity profile, which is the curve $\dot{s}(s)_{s \in [0,s_{\text{end}}]}$ in the $s$–$\dot{s}$ plane. We say that a time-parameterization or, equivalently, a velocity profile, is valid if $s(t)_{t \in [0,T']} \in C^1$ is continuous, $\dot{s}$ is always strictly positive, and the retimed trajectory $q(s(t))_{t \in [0,T']}$ satisfies the constraints of the system.

To check whether the retimed trajectory satisfies the system constraints, one may differentiate $q(s(t))$ with respect to $t$:

$$\dot{q} = q_s \dot{s}, \quad \ddot{q} = q_s \ddot{s} + q_{ss} \dot{s}^2,$$

where dots denote differentiations with respect to the time parameter $t$ and $q_s = \frac{dq}{ds}$ and $q_{ss} = \frac{d^2q}{ds^2}$. Substituting (3) into (1) then leads to

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

which can be rewritten as

$$\ddot{s}a(s) + \dot{s}^2 b(s) + c(s) \leq 0,$$

where

$$a(s) \overset{\text{def}}{=} A(q(s))q_s(s),$$
$$b(s) \overset{\text{def}}{=} A(q(s))q_{ss}(s) + q_s(s)^\top B(q(s))q_s(s),$$
$$c(s) \overset{\text{def}}{=} f(q(s)).$$

Each row $i$ of equation (4) is of the form

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

Next,

- 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 acceleration upper bound $\beta_i(s, \dot{s}) = \frac{c_i(s) - b_i(s)\dot{s}^2}{a_i(s)}$;
- 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 acceleration lower bound $\alpha_i(s, \dot{s}) = \frac{c_i(s) - b_i(s)\dot{s}^2}{a_i(s)}$.
One can then define for each \((s, \dot{s})\)

\[
\alpha(s, \dot{s}) \overset{\text{def}}{=} \max_i \alpha_i(s, \dot{s}), \quad \beta(s, \dot{s}) \overset{\text{def}}{=} \min_i \beta_i(s, \dot{s}).
\]

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

\[
\forall t \in [0, T'] \quad \alpha(s(t), \dot{s}(t)) \leq \dot{s}(t) \leq \beta(s(t), \dot{s}(t)). \tag{6}
\]

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, \(\alpha\) in short) to obtain minimum acceleration profiles (or \(\alpha\)-profiles), or following the field \(\beta\) to obtain maximum acceleration profiles (or \(\beta\)-profiles).

Next, observe that if \(\alpha(s, \dot{s}) > \beta(s, \dot{s})\) then, from \([6]\), there is no possible value for \(\ddot{s}\). Thus, to be valid, every velocity profile must stay below the maximum velocity curve (MVC in short) defined by\(^2\)

\[
\text{MVC}(s) \overset{\text{def}}{=} \begin{cases} 
\min \{ \ddot{s} \geq 0 : \alpha(s, \dot{s}) = \beta(s, \dot{s}) \} & \text{if } \alpha(s, 0) \leq \beta(s, 0), \\
0 & \text{if } \alpha(s, 0) > \beta(s, 0). 
\end{cases} \tag{7}
\]

It was shown \cite{Shiller and Lu 1992} that the time-minimal velocity profile is obtained by a bang-bang-type control, i.e., whereby the optimal profile follows alternatively the \(\beta\) and \(\alpha\) fields while always staying below the MVC. A method to find the optimal profile then consists in (see Fig.\([1A]\) for illustration):

- find all the possible \(\alpha \rightarrow \beta\) switch points. There are three types of such switch points: “discontinuous”, “singular” or “tangent” and they must all be on the MVC. The procedure to find these switch points is detailed in \cite{Pham 2014}:

- from each of these switch points, integrate backward following \(\alpha\) and forward following \(\beta\) to obtain the Limiting Curves (LC) \cite{Slotine and Yang 1989}:

- construct the Concatenated Limiting Curve (CLC) by considering, for each \(s\), the value of the lowest LC at \(s\);

- integrate forward from \((0, \dot{s}_{\text{beg}})\) following \(\beta\) and backward from \((s_{\text{end}}, \dot{s}_{\text{end}})\) following \(\alpha\), and consider the intersection of these profiles with each other or with the CLC. Note that the path velocities \(\dot{s}_{\text{beg}}\) and \(\dot{s}_{\text{end}}\) are computed from the desired initial and final velocities \(v_{\text{beg}}\) and \(v_{\text{end}}\) by

\[
\dot{s}_{\text{beg}} \overset{\text{def}}{=} v_{\text{beg}}/\|q_s(0)\|, \quad \dot{s}_{\text{end}} \overset{\text{def}}{=} v_{\text{end}}/\|q_s(s_{\text{end}})\|. \tag{8}
\]

We now prove two lemmata that will be important later on.

**Lemma 1 (Switch Point Lemma)** Assume that a forward \(\beta\)-profile hits the MVC at \(s = s_1\) and a backward \(\alpha\)-profile hits the MVC at \(s = s_2\), with \(s_1 < s_2\), then there exists at least one \(\alpha \rightarrow \beta\) switch point on the MVC at some position \(s_3 \in [s_1, s_2]\).

\(^2\)Setting \(\text{MVC}(s) = 0\) whenever \(\alpha(s, 0) > \beta(s, 0)\) as in \([7]\) precludes multiple-valued MVCs \cite{Shiller and Dubowsky 1985}. We made this choice throughout the paper for clarity of exposition. However, in the implementation, we did consider multiple-valued MVCs.
Proof: At \((s_1, \text{MVC}(s_1))\), the angle from the vector \(\beta\) to the tangent to the MVC is negative (see Fig. 1B). In addition, since we are on the MVC, we have \(\alpha = \beta\), thus the angle from \(\alpha\) to the tangent is negative too. Next, at \((s_2, \text{MVC}(s_2))\), the angle of \(\alpha\) to the tangent to the MVC is positive (see Fig. 1B). Thus, since the vector field \(\alpha\) is continuous, there exists, between \(s_1\) and \(s_2\)

(i) either a point where the angle between \(\alpha\) and the tangent to the MVC is 0 – in which case we have a tangent switch point;

(ii) or a point where the MVC is discontinuous – in which case we have a discontinuous switch point;

(iii) or a point where the MVC is continuous but non differentiable – in which case we have a singular switch point.

For more details, the reader is referred to [Pham 2014].

Lemma 2 (Continuity of the CLC) Either one of the LC’s reaches \(\dot{s} = 0\), or the CLC is continuous.

Proof: Assume by contradiction that no LC reaches \(\dot{s} = 0\) and that there exists a “hole” in the CLC. The left border \(s_1\) of the hole must then be defined by the intersection of the MVC with a forward \(\beta\)-LC (coming from the previous \(\alpha \rightarrow \beta\) switch point), and the right border \(s_2\) of the hole must be defined by the intersection of the MVC with a backward \(\alpha\)-LC (coming from the following \(\alpha \rightarrow \beta\) switch point). By Lemma 1 above, there must then exist a switch point between \(s_1\) and \(s_2\), which contradicts the definition of the hole.

2.2 Admissible Velocity Propagation (AVP)

This section presents the Admissible Velocity Propagation algorithm (AVP), which constitutes the heart of our approach. This algorithm takes as inputs:

- a path \(P\) in the configuration space, and
- an interval \([s_{\text{min}}^{\text{beg}}, s_{\text{max}}^{\text{beg}}]\) of initial path velocities;
and returns the interval (cf. Theorem 1) \([s_{\text{min}}^{\text{end}}, s_{\text{max}}^{\text{end}}]\) of all path velocities that the system can reach at the end of \(P\) after traversing \(P\) while respecting the system constraints. The algorithm comprises the following three steps:

A Compute the limiting curves;

B Determine the maximum final velocity \(s_{\text{max}}^{\text{end}}\) by integrating forward from \(s = 0\);

C Determine the minimum final velocity \(s_{\text{min}}^{\text{end}}\) by bisection search and by integrating backward from \(s = s_{\text{end}}\).

We now detail each of these steps.

A Computing the limiting curves

We first compute the Concatenated Limiting Curve (CLC) as shown in Section 2.1. From Lemma 2, either one of the LC’s reaches 0 or the CLC is continuous. The former case is covered by A1 below, while the latter is covered by A2–5.

A1 One of the LC’s hits the line \(\dot{s} = 0\). In this case, the path cannot be traversed by the system without violating the kinodynamic constraints: AVP returns Failure. Indeed, assume that a backward (\(\alpha\)) profile hits \(\dot{s} = 0\). Then any profile that goes from \(s = 0\) to \(s = s_{\text{end}}\) must cross that profile somewhere and from above, which violates the \(\alpha\) bound (see Figure 2A). Similarly, if a forward (\(\beta\)) profile hits \(\dot{s} = 0\), then that profile must be crossed somewhere and from below, which violates the \(\beta\) bound. Thus, no valid profile can go from \(s = 0\) to \(s = s_{\text{end}}\);

The CLC is now assumed to be continuous and strictly positive. Since it is bounded by \(s = 0\) from the left, \(s = s_{\text{end}}\) from the right, \(\dot{s} = 0\) from the bottom and the MVC from the top, there are only four exclusive and exhaustive cases, listed below.

A2 The CLC hits the MVC while integrating backward and while integrating forward.

In this case, let \(s_{\text{beg}}^{*} \triangleq \text{MVC}(0)\) and go to B. The situation where there is no switch point is assimilated to this case;

\[ \text{Johnson and Hauser [2012]} \] also introduced a velocity interval propagation algorithm along a path but for pure kinematic constraints and moving obstacles.

Figure 2: Illustration for step A (computation of the LC’s). A: illustration for case A1. A profile that crosses an \(\alpha\)-CLC violates the \(\alpha\) bound. B: illustration for case A3.

The CLC is now assumed to be continuous and strictly positive. Since it is bounded by \(s = 0\) from the left, \(s = s_{\text{end}}\) from the right, \(\dot{s} = 0\) from the bottom and the MVC from the top, there are only four exclusive and exhaustive cases, listed below.

A2 The CLC hits the MVC while integrating backward and while integrating forward.

In this case, let \(s_{\text{beg}}^{*} \triangleq \text{MVC}(0)\) and go to B. The situation where there is no switch point is assimilated to this case;
A3 The CLC hits $s = 0$ while integrating backward, and the MVC while integrating forward (see Figure 2B). In this case, let $\dot{s}_{\text{beg}} \overset{\text{def}}{=} \text{CLC}(0)$ and go to B;

A4 The CLC hits the MVC while integrating backward, and $s = s_{\text{end}}$ while integrating forward. In this case, let $\dot{s}_{\text{beg}} \overset{\text{def}}{=} \text{MVC}(0)$ and go to B;

A5 The CLC hits $s = 0$ while integrating backward, and $s = s_{\text{end}}$ while integrating forward. In this case, let $\dot{s}_{\text{beg}} \overset{\text{def}}{=} \text{CLC}(0)$ and go to B.

B Determining the maximum final velocity  Note that, in any of the cases A2–5, $\dot{s}_{\text{beg}}$ was defined so that no valid profile can start above it. Thus, if $\dot{s}_{\text{min}} > \dot{s}_{\text{beg}}$, the path is not traversable: AVP returns Failure. Otherwise, the interval of valid initial velocities is $[\dot{s}_{\text{min}}, \dot{s}_{\text{max}}]$ where $\dot{s}_{\text{max}} \overset{\text{def}}{=} \min(\dot{s}_{\text{beg}}, \dot{s}_{\text{beg}})$.

Definition 2 Under the nomenclature introduced in Definition 1, we say that a velocity $\dot{s}_{\text{end}}$ is a valid final velocity if there exists a valid profile that starts at $(0, \dot{s}_{0})$ for some $\dot{s}_{0} \in [\dot{s}_{\text{min}}, \dot{s}_{\text{max}}]$ and ends at $(s_{\text{end}}, \dot{s}_{\text{end}})$.

We argue that the maximum valid final velocity can be obtained by integrating forward from $\dot{s}_{\text{beg}}$ following $\beta$. Let’s call $\Phi$ the velocity profile obtained by doing so. Since $\Phi$ is continuous and bounded by $s = s_{\text{end}}$ from the right, $\dot{s} = 0$ from the bottom, and either the MVC or the CLC from the top, there are four exclusive and exhaustive cases, listed below (see Figure 3 for illustration).

Figure 3: Illustration for step B: one can determine the maximum final velocity by integrating forward from $(0, \dot{s}_{\text{beg}})$.

B1 $\Phi$ hits $\dot{s} = 0$ (cf. profile B1 in Fig. 3). Here, as in the case A1, the path is not traversable: AVP returns Failure. Indeed, any profile that starts below $\dot{s}_{\text{beg}}$ and tries to reach $s = s_{\text{end}}$ must cross $\Phi$ somewhere and from below, thus violating the $\beta$ bound;

B2 $\Phi$ hits $s = s_{\text{end}}$ (cf. profile B1 in Fig. 3). Then $\Phi(s_{\text{end}})$ corresponds to the $\dot{s}_{\text{end}}$ we are looking for. Indeed, $\Phi(s_{\text{end}})$ is reachable – precisely by $\Phi$ –, and to reach any value above $\Phi(s_{\text{end}})$, the corresponding profile would have to cross $\Phi$ somewhere and from below;
B3 Φ hits the CLC. There are two sub-cases:

B3a If we proceed from cases A4 or A5 (in which the CLC reaches \( s = s_{\text{end}} \), cf. profile B3 in Fig. 3), then \( \text{CLC}(s_{\text{end}}) \) corresponds to the \( \dot{s}_{\text{max}} \) we are looking for. Indeed, \( \text{CLC}(s_{\text{end}}) \) is reachable — precisely by the concatenation of Φ and the CLC —, and no value above \( \text{CLC}(s_{\text{end}}) \) can be valid by the definition of the CLC;

B3b If we proceed from cases A2 or A3, then the CLC hits the MVC while integrating forward, say at \( s = s_{1} \); we then proceed as in case B4 below;

B4 Φ hits the MVC, say at \( s = s_{1} \). It is clear that \( \text{MVC}(s_{\text{end}}) \) is an upper bound of the valid final velocities, but we have to ascertain whether this value is reachable. For this, we use the predicate IS_VALID defined in Box 1 of C:

- if \( \text{IS\_VALID}(\text{MVC}(s_{\text{end}})) \), then \( \text{MVC}(s_{\text{end}}) \) is the \( \dot{s}_{\text{max}} \) we are looking for;
- else, the path is not traversable: AVP returns Failure. Indeed, as we shall see, if for a certain \( \dot{s}_{\text{test}} \), the predicate IS\_VALID(\( \dot{s}_{\text{test}} \)) is \text{False}, then no velocity below \( \dot{s}_{\text{test}} \) can be valid either.

C Determining the minimum final velocity Assume that we proceed from the cases B2–4. Consider a final velocity \( \dot{s}_{\text{test}} \) where

- \( \dot{s}_{\text{test}} < \Phi(s_{\text{end}}) \) if we proceed from B2;
- \( \dot{s}_{\text{test}} < \text{CLC}(s_{\text{end}}) \) if we proceed from B3a;
- \( \dot{s}_{\text{test}} < \text{MVC}(s_{\text{end}}) \) if we proceed from B3b or B4.

Let us integrate backward from \((s_{\text{end}}, \dot{s}_{\text{test}})\) following \( \alpha \) and call the resulting profile \( \Psi \). We have the following lemma.

Lemma 3 \( \Psi \) cannot hit the MVC before hitting either Φ or the CLC.

Proof: If we proceed from B2 or B3a, then it is clear that \( \Psi \) must first hit Φ (case B2) or the CLC (case B3a) before hitting the MVC. If we proceed from B3b or B4, assume by contradiction that \( \Psi \) hits the MVC first at a position \( s = s_{2} \). Then by Lemma 1, there must exist a switch point between \( s_{2} \) and the end of the CLC (in case B3b) or the end of Φ (in case B4). In both cases, there is a contradiction with the fact that the CLC is continuous.

We can now detail in Box the predicate IS_VALID which assesses whether a final velocity \( \dot{s}_{\text{test}} \) is valid.

At this point, we have that, either the path is not traversable, or we have determined \( \dot{s}_{\text{max}} \) in B. Remark from C3–5 that, if some \( \dot{s}_{0} \) is a valid final velocity, then any \( \dot{s} \in [\dot{s}_{0}, \dot{s}_{\text{max}}] \) is also valid. Similarly, from C1 and C2, if some \( \dot{s}_{0} \) is not a valid final velocity, then no \( \dot{s} \leq s_{0} \) can be valid. We have thus established the following result:

Theorem 1: The set of valid final velocities is an interval.

This interval property enables one to efficiently search for the minimum final velocity as follows. First, test whether 0 is a valid final velocity: if IS\_VALID(0), then the
**Box 1: IS_VALID**

**Input:** candidate final velocity $\dot{s}_{\text{test}}$

**Output:** True if there exists a valid velocity profile with final velocity $\dot{s}_{\text{test}}$

Consider the profile $\Psi$ constructed above. Since it must hit $\Phi$ or the CLC before hitting the MVC, the following five cases are exclusive and exhaustive (see Fig. 4 for illustrations):

**C1** $\Psi$ hits $\dot{s} = 0$ (Fig. 4, profile C1). Then, as in cases A1 or B1, no velocity profile can reach $s_{\text{test}}$: return False;

**C2** $\Psi$ hits $s = 0$ for some $\dot{s}_0 < \dot{s}_{\text{min}}$ (see Figure 4, profile C2). Then any profile that ends at $\dot{s}_{\text{test}}$ would have to hit $\Psi$ from above, which is impossible: return False;

**C3** $\Psi$ hits $s = 0$ at a point $\dot{s}_0 \in [\dot{s}_{\text{min}}^{\text{beg}}, \dot{s}_{\text{max}}^{\text{beg}}]$ (Fig. 4, profile C3). Then $\dot{s}_{\text{test}}$ can be reached following the valid velocity profile $\Psi$: return True. (Note that, if $\dot{s}_0 > \dot{s}_{\text{max}}^{\text{beg}}$ then $\Psi$ must have crossed $\Phi$ somewhere before arriving at $s = 0$, which is covered by case C4 below);

**C4** $\Psi$ hits $\Phi$ (Fig. 4, profile C4). Then $\dot{s}_{\text{test}}$ can be reached, precisely by the concatenation of a part of $\Phi$ and $\Psi$: return True;

**C5** $\Psi$ hits the CLC (Fig. 4, profile C5). Then $\dot{s}_{\text{test}}$ can be reached, precisely by the concatenation of $\Phi$, a part of the CLC and $\Psi$: return True.

Figure 4: Illustration for the predicate IS_VALID: one can assess whether a final velocity $\dot{s}_{\text{test}}$ is valid by integrating backward from $(s_{\text{end}}, \dot{s}_{\text{test}})$. 
sought-after $\dot{s}_{\text{end}}^{\text{min}}$ is 0. Else, run a standard bisection search with initial bounds $(0, \dot{s}_{\text{end}}^{\text{max}})$ where 0 is not valid and $\dot{s}_{\text{end}}^{\text{max}}$ is valid. Thus, after executing $\log_2(1/\epsilon)$ times the routine ISVALID, one can determine $\dot{s}_{\text{end}}^{\text{min}}$ with an error smaller than $\epsilon$.

2.3 Remarks

Implementation and complexity of AVP  As clear from the previous section, AVP can be readily adapted from the numerical integration approach to TOPP. As a matter of fact, we implemented AVP in about 100 lines of C++ code based on the TOPP library we developed previously (see https://github.com/quangounet/TOPP).

In terms of complexity, the main difference between AVP and TOPP lies in the bisection search of step C, which requires $\log(1/\epsilon)$ backward integrations. However, in practice, these integrations terminate quickly, either by hitting the MVC or the line $\dot{s} = 0$. Thus, the actual running time of AVP is only slightly larger than that of TOPP. As illustration, in the bottle experiment of Section 4.2 we considered 100 random paths, discretized with grid size $N = 1000$. TOPP and AVP under velocity, acceleration and balance constraints took the same amount of computation time $0.033 \pm 0.003$ s per path.

AVP-backward  Consider the “AVP-backward” problem: given an interval of final velocities $[\dot{s}_{\text{end}}^{\text{min}}, \dot{s}_{\text{end}}^{\text{max}}]$, compute the interval $[\dot{s}_{\text{beg}}^{\text{min}}, \dot{s}_{\text{beg}}^{\text{max}}]$ of all possible initial velocities. As we shall see in Section 3.2, AVP-backward is essential for the bi-directional version of AVP-RRT.

It turns out that AVP-backward can be easily obtained by modifying AVP as follows [Lertkultanon and Pham, 2014]:

- step A of AVP-backward is the same as in AVP;
- in step B of AVP-backward, one integrates backward from a suitably defined $\dot{s}_{\text{beg}}^{\text{min}}$ instead of integrating forward from $\dot{s}_{\text{beg}}^{\text{max}}$;
- in the bisection search of step C of AVP-backward, one integrates forward from $(0, \dot{s}_{\text{test}})$ instead of integrating backward from $(\dot{s}_{\text{end}}, \dot{s}_{\text{test}})$.

Convex optimization approach  As mentioned in the Introduction, “convex optimization” is another possible approach to TOPP [Verscheure et al., 2009, Hauser, 2014]. It is however unclear to us whether one can modify that approach to yield a “convex-optimization-based AVP” other than sampling a large number of $(\dot{s}_{\text{start}}, \dot{s}_{\text{end}})$ pairs and running the “convex-optimization-based TOPP” between $(0, \dot{s}_{\text{start}})$ and $(\dot{s}_{\text{end}}, \dot{s}_{\text{end}})$, which would arguably be very slow.

3 Kinodynamic trajectory planning using AVP

3.1 Combining AVP with sampling-based planners

The AVP algorithm presented in Section 2.2 is general and can be combined with various iterative path planners. As an example, we detail in Box 2 and illustrate in Figure 5 a
planner we call AVP-RRT, which results from the combination of AVP with the standard RRT path planner [Kuffner and LaValle, 2000].

As in the standard RRT, AVP-RRT iteratively constructs a tree \( \mathcal{T} \) in the configuration space. However, in contrast with the standard RRT, a vertex \( V \) here consists of a triple \((V\.\text{config}, V\.\text{inpath}, V\.\text{interval})\) where \( V\.\text{config} \) is an element of the configuration space \( \mathcal{C} \), \( V\.\text{inpath} \) is a path \( P \subset \mathcal{C} \) that connects the configuration of \( V \)’s parent to \( V\.\text{config} \), and \( V\.\text{interval} \) is the interval of reachable velocities at \( V\.\text{config} \), that is, at the end of \( V\.\text{inpath} \).

At each iteration, a random configuration \( q_{\text{rand}} \) is generated. The EXTEND routine (see Box 3) then tries to extend the tree \( \mathcal{T} \) towards \( q_{\text{rand}} \) from the closest – in a certain metric \( d \) – vertex in \( \mathcal{T} \). The algorithm terminates when either

- A newly-found vertex can be connected to the goal configuration (line 10 of Box 2). In this case, AVP guarantees by recursion that there exists a path from \( q_{\text{start}} \) to \( q_{\text{goal}} \) and that this path is time-parameterizable;
- After \( N_{\text{maxrep}} \) repetitions, no vertex could be connected to \( q_{\text{goal}} \). In this case, the algorithm returns Failure.

**Box 2: AVP-RRT**

**Input**: \( q_{\text{start}}, q_{\text{goal}} \)

**Output**: A valid trajectory connecting \( q_{\text{start}} \) to \( q_{\text{goal}} \) or Failure

1: \( \mathcal{T} \leftarrow \text{NEW\_TREE()} \)
2: \( V_{\text{start}} \leftarrow \text{NEW\_VERTEX()} \)
3: \( V_{\text{start}}\.\text{config} \leftarrow q_{\text{start}}; V_{\text{start}}\.\text{inpath} \leftarrow \text{Null}; V_{\text{start}}\.\text{interval} \leftarrow [0,0] \)
4: \( \text{INITIALIZE}(\mathcal{T}, V_{\text{start}}) \)
5: \( \text{for rep = 1 to } N_{\text{maxrep}} \text{ do} \)
6: \( q_{\text{rand}} \leftarrow \text{RANDOM\_CONFIG()} \)
7: \( V_{\text{new}} \leftarrow \text{EXTEND}(\mathcal{T}, q_{\text{rand}}) \)
8: \( \text{if EXTEND succeeds then} \)
9: \( \text{ADD\_VERTEX}(\mathcal{T}, V_{\text{new}}) \)
10: \( \text{if CONNECT}(V_{\text{new}}, q_{\text{goal}}) \text{ succeeds then} \)
11: \( \text{return COMPUTE\_TRAJECTORY}(\mathcal{T}, q_{\text{goal}}) \)
12: \( \text{end if} \)
13: \( \text{end if} \)
14: \( \text{end for} \)
15: \( \text{return Failure} \)

The other routines are defined as follows:

- \( \text{CONNECT}(V, q_{\text{goal}}) \) attempts at connecting directly \( V \) to the goal configuration \( q_{\text{goal}} \), using the same algorithm as in lines 2 to 10 of Box 3, but with the further requirement that the goal velocity is included in the final velocity interval;

- \( \text{COMPUTE\_TRAJECTORY}(\mathcal{T}, q_{\text{goal}}) \) reconstructs the entire path \( P_{\text{total}} \) from \( q_{\text{start}} \) to \( q_{\text{goal}} \) by recursively concatenating the \( V\.\text{inpath} \). Next, \( P_{\text{total}} \) is time-parameterized.
Figure 5: Illustration for AVP-RRT. The horizontal plane represents the configuration space while the vertical axis represents the path velocity space. Black areas represent configuration space obstacles. A vertex in the tree is composed of a configuration (blue disks), the incoming path from the parent (blue curve), and the interval of admissible velocities (magenta segment). At each tree extension step, one interpolates a smooth, collision-free path in the configuration space and propagates the interval of admissible velocities along that path using AVP.

Box 3: EXTEND

Input: $\mathcal{T}$, $q_{\text{rand}}$

Output: A new vertex $V_{\text{new}}$ or Failure

1: $V_{\text{near}} \leftarrow \text{NEAREST\_NEIGHBOR}(\mathcal{T}, q_{\text{rand}})$
2: $(P_{\text{new}}, q_{\text{new}}) \leftarrow \text{INTERPOLATE}(V_{\text{near}}, q_{\text{rand}})$
3: if $P$ is collision-free then
4: $[\dot{s}_{\text{min}}, \dot{s}_{\text{max}}] \leftarrow \text{AVP}(P_{\text{new}}, V_{\text{near}}.\text{interval})$
5: if AVP succeeds then
6: $V_{\text{new}} \leftarrow \text{NEW\_VERTEX}()$
7: $V_{\text{new}}.\text{config} \leftarrow q_{\text{new}}; V_{\text{new}}.\text{inpath} \leftarrow P_{\text{new}}; V_{\text{new}}.\text{interval} \leftarrow [\dot{s}_{\text{min}}, \dot{s}_{\text{max}}]$
8: return $V_{\text{new}}$
9: end if
10: end if
11: return Failure
by applying TOPP. The existence of a valid time-parameterization is guaranteed by recursion by AVP.

- **NEAREST\_NEIGHBOR(T, q)** returns the vertex of \( T \) whose configuration is closest to configuration \( q \) in the metric \( d \), see Section 3.2 for a more detailed discussion.

- **INTERPOLATE(V, q)** returns a pair \((P_{\text{new}}, q_{\text{new}})\) where \( q_{\text{new}} \) is defined as follows
  
  - if \( q \) is close enough to \( V\.config \):
    \[ q_{\text{new}} \leftarrow q; \]
  
  - otherwise:
    \[ q_{\text{new}} \text{ is a configuration situated somewhere “between” } V\.config \text{ and } q. \]

  The path \( P_{\text{new}} \) is a smooth path connecting \( V\.config \) and \( q_{\text{new}} \), and such that the concatenation of \( V\.inpath \) and \( P_{\text{new}} \) is \( C^1 \) at \( V\.config \), see Section 3.2 for a more detailed discussion.

### 3.2 Implementation and variations

As in the standard RRT [Kuffner and LaValle, 2000], some implementation choices influence substantially the performance of the algorithm.

**Metric** In state-space RRTs, the most critical choice is that of the metric \( d \), in particular, the relative weighting between configuration-space coordinates and velocity coordinates. In our approach, since the whole interval of valid path velocities is considered, the relative weighting does not come into play. In practice, a simple Euclidean metric on the configuration space is often sufficient. However, in some applications, one may also include the final orientation of \( V\.inpath \) in the metric.

**Interpolation** In geometric path planners, the interpolation between two configurations is usually done using a straight segment. Here, since one needs to propagate velocities, it is necessary to enforce \( C^1 \)-continuity at the junction point. In the examples of Section 4 we used third-degree polynomials to do so. Other interpolation methods are possible: higher-order polynomials, splines, etc. The choice of the appropriate method depends on the application and plays an important role in the performance of the algorithm.

**K-nearest-neighbors** Attempting connection from \( K \) nearest neighbors, where \( K > 1 \) is a judiciously chosen parameter, has been found to improve the performance of RRT. To implement this, it suffices to replace line 2 of Box 3 with a FOR loop that enumerates the \( K \) nearest neighbors.

Another significant benefit of AVP is that one can readily adapt heuristics that have been developed for geometric path planners. We discuss two such heuristics below.

**Bi-directional RRT** [Kuffner and LaValle, 2000] remarked that growing simultaneously two trees, one rooted at the initial configuration and one rooted at the goal configuration yielded significant improvement over the classical uni-directional RRT. This idea [see also Nakamura and Mukherjee, 1991] can be easily implemented in the context of AVP-RRT as follows [Lertkultanon and Pham, 2014]:
• The start tree is grown normally as in Section 3.1.
• The goal tree is grown similarly, but using AVP-backward (see Section 2.3) for the velocity propagation step;
• Assume that one finds a configuration where the two trees are geometrically connected. If the forward velocity interval of the start tree and the backward velocity interval of the goal tree have a non-empty intersection at this configuration, then the two trees can be connected dynamically.

Bridge test If two nearby configurations are in the obstacle space but their midpoint $q$ is in the free space, then most probably $q$ is in a narrow passage. This idea enables one to find a large number of such configurations $q$, which is essential in problems involving narrow passages [Hsu et al., 2003]. This idea can be easily implemented in AVP-RRT by simply modifying RANDOM_CONFIG in line 6 of Box 2 to include the bridge test.

One can observe from the above discussion that powerful heuristics developed for geometric path planning can be readily used in AVP-RRT, precisely because the latter is built on the idea of path-velocity decomposition. It is unclear how such heuristics can be integrated in other approaches to kinodynamic motion planning such as the trajectory optimization approach discussed in the Introduction.

4 Examples of application

As AVP-RRT is based on the classical Time-Optimal Path Parameterization (TOPP) algorithm, it can be applied to any type of systems and constraints TOPP can handle, from double-integrators subject to velocity and acceleration bounds, to manipulator subject to torque limits [Bobrow et al., 1985; Shin and McKay, 1986], to wheeled vehicles subject to balance constraints [Shiller and Gwo, 1991], to humanoid robots in multi-contact tasks [Pham and Stasse, 2014], etc. Furthermore, the overhead for addressing a new problem is minimal: it suffices to reduce the system constraints to the form of inequality (1), and le tour est joué!

In this section, we present two examples where AVP-RRT was used to address planning problems in which no quasi-static solution exists. In the first example, the task consisted in swinging a double pendulum into the upright configuration under severe torque bounds. While this example does not fully exploit the advantages associated with path-velocity decomposition (no configuration-space obstacle nor kinematic closure constraint was considered), we chose it since it was simple enough to enable a careful comparison with the usual state-space planning approach [LaValle and Kuffner, 2001]. In the second example, the task consisted in transporting a bottle placed on a tray through a small opening. This example demonstrates the full power of path-velocity decomposition: configuration-space constraints (going through the small opening) and dynamics constraints (the bottle must remain on the tray) could be addressed separately. To the best of our knowledge, this is the first successful demonstration on a physical robot with dof $\geq 6$ that kinodynamic planning can succeed where quasi-static planning is guaranteed to fail.
4.1 Double pendulum with severe torque bounds

We first consider a fully-actuated double pendulum (see Figure 6B), subject to torque limits
\[ |\tau_1| \leq \tau_{1\text{max}}, \quad |\tau_2| \leq \tau_{2\text{max}}. \]

Such a pendulum can be seen as a 2-link manipulator, so that the reduction to the form of (1) is straightforward, see [Pham] 2014.

4.1.1 Obstruction to quasi-static planning

The task consisted in bringing the pendulum from its initial state \((\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, 0, 0)\) towards the upright state \((\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (\pi, 0, 0, 0)\), while respecting the torque bounds. For simplicity, we did not consider self-collision issues.

Any trajectory that achieves the task must pass through a configuration where \(\theta_1 = \pi/2\). Note that the configuration with \(\theta_1 = \pi/2\) that requires the smallest torque at the first joint to stay still is \((\theta_1, \theta_2) = (\pi/2, \pi)\). Let then \(\tau_{1\text{qs}}\) be this smallest torque. It is clear that, if \(\tau_{1\text{max}} < \tau_{1\text{qs}}\), then no quasi-static trajectory can achieve the task.

In our simulations, we used the following lengths and masses for the links: \(l = 0.2\) m and \(m = 8\) kg, yielding \(\tau_{1\text{qs}} = 15.68\) N-m. For information, the smallest torque at the second joint to keep the configuration \((\theta_1, \theta_2) = (0, \pi/2)\) stationary was 7.84 N-m. We carried experiments in the following scenarios: \((\tau_{1\text{max}}, \tau_{2\text{max}}) \in \{(11, 7), (13, 5), (11, 5)\}\) (N-m).

4.1.2 Solution using AVP-RRT

For simplicity we used the uni-directional version of AVP-RRT as described in Section 3, without any heuristics. Furthermore, for fair comparison with state-space RRT in Python (see Section 4.1.3), we used a Python implementation of AVP rather than the C++ implementation contained in the TOPP library [Pham] 2014.

Regarding the number of nearest neighbors to consider, we chose \(K = 10\). The maximum number of repetitions was set to \(N_{\text{maxrep}} = 2000\). Random configurations were sampled uniformly in \([-\pi, \pi]^2\). A simple Euclidean metric in the configuration space was used. Inverse Dynamics computations (required by the TOPP algorithm) were performed using OpenRAVE [Diankov] 2010. We ran 40 simulations for each value of \((\tau_{1\text{max}}, \tau_{2\text{max}})\) on a 2 GHz Intel Core Duo computer with 2 GB RAM. The results are given in Table 1 and Figure 6. A video of some successful trajectories are shown at [http://youtu.be/oFyPhI3JW00](http://youtu.be/oFyPhI3JW00).

| \(\tau_{\text{max}}\) (N-m) | Success rate | Configs tested | Vertices added | Search time (min) |
|--------------------------|-------------|----------------|----------------|------------------|
| (11,7)                   | 100%        | 64±44          | 31±23          | 4.2±2.7          |
| (13,5)                   | 100%        | 92±106         | 29±30          | 5.9±6.3          |
| (11,5)                   | 92.5%       | 212±327        | 56±81          | 12.1±15.0        |
Figure 6: Swinging up a fully-actuated double pendulum. A typical solution for the case $(\tau_1^{\text{max}}, \tau_2^{\text{max}}) = (11, 5) \, \text{N} \cdot \text{m}$, with trajectory duration $1.88 \, \text{s}$ (see also the attached video). A: The tree in the $(\theta_1, \theta_2)$ space. The final path is highlighted in magenta. B: Snapshots of the trajectory, taken every $0.1 \, \text{s}$. Snapshots taken near the beginning of the trajectory are lighter. A video of the movement is available at http://youtu.be/oFyPhI3JN00. C: Velocity profiles in the $(s, \dot{s})$ space. The MVC is in cyan. The various velocity profiles (CLC, $\Phi$, $\Psi$, cf. Section 2.2) are in black. The final, optimal, velocity profile is in dashed blue. The vertical dashed red lines correspond to vertices where $0$ is a valid velocity, which allowed a discontinuity of the path tangent at that vertex. D: Torques profiles. The torques for joint 1 and 2 are respectively in red and in blue. The torque limits are in dotted line. Note that, in agreement with time-optimal control theory, at each time instant, at least one torque limit was saturated (the small overshoots were caused by discretization errors).
4.1.3 Comparison with state-space RRT

We compared our implementation of AVP-RRT with the standard state-space RRT [LaValle and Kuffner, 2001] including the $K$-nearest-neighbors heuristic ($K$NN-RRT). We do not provide a comparison with more complex RRT-based kinodynamic planners [such as LQR-RRT, Perez et al., 2012; Tedrake, 2009], which would be out of the scope of the present work. However, we made a special effort to fine-tune the state-space RRT we considered, see Appendix A. In particular, we carefully determined the optimal number of neighbors $K$ for $K \in \{1, 10, 40, 100\}$. Figure 7 and Table 2 summarize the results.

Figure 7: Comparison of AVP-RRT and $K$NN-RRT. A: Percentage of trials that have reached the goal area at given time instants for $\tau_{\text{max}} = (11, 7)$. B: Individual plots for each trial. Each curve shows the distance to the goal as a function of time for a given instance (red: AVP-RRT, blue: RRT-40). Dots indicate the time instants when a trial successfully terminated. Stars show the mean values of termination times. C and D: same legends as A and B but for $\tau_{\text{max}} = (11, 5)$.

In the two problem instances, AVP-RRT was respectively 13.4 and 5.6 times faster than the best $K$NN-RRT in terms of search time. We noted however that the search time of AVP-RRT increased significantly from instance $(\tau_{1}^{\text{max}}, \tau_{2}^{\text{max}}) = (11, 7)$ to instance $(\tau_{1}^{\text{max}}, \tau_{2}^{\text{max}}) = (11, 5)$, while that of RRT only marginally increased. This may be caused by the “superposition” phenomenon: as torque constraints become tighter, more “pumping” swings are necessary to reach the upright configuration. However, since our metric was only on the configuration-space variables, configurations with different speeds
Table 2: Comparison of AVP-RRT and KNN-RRT

| Planner   | τ_m = (11, 7) | γ_m = (11, 5) |
|-----------|---------------|---------------|
|           | Success rate | Search time (min) | Success rate | Search time (min) |
| AVP-RRT   | 100%         | 3.3±2.6        | 100%         | 9.8±12.1         |
| RRT-1     | 40%          | 70.0±34.1      | 47.5%        | 63.8±36.6        |
| RRT-10    | 82.5%        | 53.1±59.5      | 85%          | 56.3±60.1        |
| RRT-40    | 92.5%        | 44.6±42.6      | 87.5%        | 54.6±52.2        |
| RRT-100   | 82.5%        | 88.4±54.0      | 92.5%        | 81.2±46.7        |

(corresponding to different pumping cycles) may become indistinguishable. While this problem could be addressed by including a measure of reachable velocity intervals into the metric, we chose not to do so in the present paper in order to avoid over-fitting our implementation of AVP-RRT to the problem at hand. Nevertheless, AVP-RRT still significantly over-performed the best KNN-RRT.

4.2 Non-prehensile object transportation

Here we consider the non-prehensile (i.e. without grasping) transportation of a bottle, or “waiter motion”. Non-prehensile transportation can be faster and more efficient than prehensile transportation since the time-consuming grasping and un-grasping stages are entirely skipped. Moreover, in many applications, the objects to be carried are too soft, fragile or small to be adequately grasped (e.g. food, electronic components, etc.)

4.2.1 Obstruction to quasi-static planning

A plastic milk bottle partially filled with sand was placed (without any fixation device) on a tray. The mass of the bottle was 2.5 kg, its height was 24 cm (the sand was filled up to 16 cm) and its base was a square of size 8 cm × 8 cm. The tray was mounted as the end-effector of a 6-dof serial manipulator (Denso VS-060). The task consisted in bringing the bottle from an initial configuration towards a goal configuration, these two configurations being separated by a small opening (see Fig. 8A).

For the bottle to remain stationary with respect to the tray, the following three conditions must be satisfied:

- (Unilaterality) The normal component $f_n$ of the reaction force must be non-negative;
- (Non-slippage) The tangential component $f_t$ of the reaction force must satisfy $\|f_t\| \leq \mu f_n$, where $\mu$ is the static friction coefficient between the bottle and the tray. In our experimental set-up, the friction coefficient was set to a high value ($\mu = 1.7$), such that the non-slippage condition was never violated before the ZMP condition;
- (ZMP) The ZMP of the bottle must lie inside the bottle base [Vukobratovic et al., 2001].

The height of the opening was designed so that, for the bottle to go through the opening, it must be tilted by at least an angle $\theta_{qs}$. However, when the bottle is tilted by that
angle, the center of mass (COM) of the bottle projects outside of the bottle base. As the projection of the COM coincides with the ZMP in the quasi-static condition, tilting the bottle by the angle $\theta_{qs}$ thus violates the ZMP condition and as a result, the bottle will tip over. One can therefore conclude that no quasi-static motion can bring the bottle through the opening without tipping it over.

4.2.2 Solution using AVP-RRT

We first reduced the three aforementioned conditions to the form of (1). Details of this reduction can be found in [Lertkultanon and Pham, 2014]. We next used the bi-directional version of AVP-RRT presented in Section 3.2. All vertices in the tree were considered for possible connection from a new random configuration, but they were sorted by increasing distance from the new configuration (a simple Euclidean metric in the configuration space was used for the distance computation). As the opening was very small (narrow passage), we made use of the bridge test [Hsu et al., 2003] in order to automatically sample a sizable number of configurations inside or close to the opening. Note that the use of the bridge test was natural thanks to path-velocity decomposition.

Because of the discrepancy between the planned motion and the motion actually executed on the robot (in particular, actual acceleration switches cannot be infinitely fast), we set the safety boundaries to be a square of size 5.5 cm $\times$ 5.5 cm (the actual base size was 8 cm $\times$ 8 cm), which makes the planning problem even harder. Nevertheless, our algorithm was able to find a feasible movement in about 3 hours on a 3.2 GHz Intel Core computer with 3.8 GB RAM, and this movement could be executed successfully on the actual robot, see Fig. 8 and the video of the movement executed on the actual robot at [http://youtu.be/LdZSjNwpJs0](http://youtu.be/LdZSjNwpJs0). Note that the computation time of 3 hours was for a particularly difficult problem instance: if the opening was only 5 cm higher, computation time would be around 1 or 2 minutes, see Fig. 8F.

5 Conclusion

We have presented a new algorithm, Admissible Velocity Propagation (AVP) which, given a path and an interval of reachable velocities at the beginning of that path, computes exactly and efficiently the interval of valid final velocities. We have shown how to combine AVP with well-known sampling-based geometric planners to give rise to a family of new efficient kinodynamic planners, which we have evaluated on two difficult kinodynamic problems. In particular, we believe the bottle transportation presented in Section 4.2 is the first successful demonstration of kinodynamic planning in a complex setting (dof $\geq 6$), on a physical robot, in an environment where quasi-static motions are guaranteed to fail.

Comparison to existing approaches to kinodynamic planning  Compared to traditional planners based on path-velocity decomposition, our planners remove the limitation of quasi-static feasibility, precisely by propagating admissible velocity intervals at each step of the tree extension. This enables our planner to find solutions when quasi-static trajectories are guaranteed to fail, as illustrated by the two examples of Section 4.
Figure 8: Non-prehensile transportation of a bottle. **A**: Simulation environment. The robot must bring the bottle to the other side of the opening while keeping it balanced on the tray. **B**: Bi-RRT tree in the workspace: the start tree had 125 vertices and the goal tree had 116 vertices. Red boxes represent the obstacles. Red stars represent the initial and goal positions of the bottle COM. Green lines represent the paths of the bottle COM in the tree. The successful path is highlighted in blue: it had 6 vertices. **C**: MVC and velocity profiles in the \((s, \dot{s})\) space. Same legend as in Fig. 6C. **D**: ZMP of the bottle in the tray reference frame (RF) for the successful trajectory. Note that the ZMP always stayed within the imposed safety borders \(\pm 2.75\) cm (the actual borders were \(\pm 4\) cm). **E**: COM of the bottle in the tray RF for the successful trajectory. Note that the X-coordinate of the COM reached the maximum value of 4.03 cm, around the moment when the bottle went through the opening, indicating that the successful trajectory would not be quasi-statically feasible. **F**: Here we varied the opening height (X-axis, from left to right: higher opening to lower opening) and determine the average and standard deviation (Y-axis, logarithmic scale) of computation time required to find a solution. We carried out 30 runs for opening heights from 0.4m to 0.365m, 10 runs for 0.36m, 3 runs for 0.355m and 0.35m and 2 runs for 0.345m. The red dashed vertical line indicates the critical height below which no quasi-static trajectory was possible. Here, we used \(\pm 4\) cm as boundaries for the ZMP, so that the computed motions, while theoretically feasible, might not be actually feasible. **Bottom**: snapshots, taken every 0.5 s, of the trajectory in B–E executed on the actual robot. A video of the movement is available at [http://youtu.be/LdZSjNwpJs0](http://youtu.be/LdZSjNwpJs0).
Compared to other approaches to kinodynamic planning, our approach enjoys the advantages associated with path-velocity decomposition, namely, the separation of the complex planning problem into two simpler sub-problems: geometric and dynamic, for both of which powerful methods and heuristics have been developed.

The bottle transportation example in Section $4.2$ illustrates clearly this advantage. To address the problem of the narrow passage constituted by the small opening, we made use of the bridge test heuristics – initially developed for geometric path planners \cite{Hsu03} – which provides a large number of samples inside the narrow passage. It is unclear how such a method could be integrated into the “trajectory optimization” approach for example. Next, to steer between two configurations, we simply interpolated a geometric path – and can check for collision at this stage – and then found possible trajectories by running AVP. By contrast, in a “state-space planning” approach, it would be difficult – if not impossible – to steer exactly between two states of the system, which requires for instance solving a two-point boundary value problem. To avoid solving such difficult problems, \cite{LaValle01, Hsu02} propose to sample a large number of time-series of random control inputs and to choose the time-series that steers the system the closest to the target state. However, such shooting methods are usually considerably slower than “exact” methods – which is the case of AVP –, as also illustrated in our simulation study (see Sections $4.1$ and Appendix).

As mentioned earlier, AVP-based planners can handle all systems and constraints that TOPP can handle. We may add here: “and only those systems and constraints”. Consequently, AVP-based planners cannot be used for most non-holonomic robots, except for a class of kinematically-controllable robots, which include space robots and some under-actuated planar manipulators \cite{Bullo01}.

Further remarks on completeness and complexity  As presented in Section $3$, AVP-RRT is likely not probabilistically complete. While we recognize the importance of completeness \cite{Caron14} and while it should be possible to make AVP-RRT probably complete by modifying the sampling and the interpolation routines, we refrain to do so in this article to keep it simple and to focus on the advantages associated with path-velocity decomposition.

We now discuss another feature of AVP-based planners that makes them interesting from a complexity viewpoint. Consider a trajectory or a trajectory segment that is “explored” by a state-space planning or a trajectory optimization method – either in one extension step for the former, or in an iterative optimization step for the latter. If one considers the underlying path of this trajectory, one may argue that these methods are exploring only one time-parameterization of that path, namely, that corresponding to the trajectory at hand. By contrast, for a given path that is “explored” by AVP, AVP precisely explores all time-parameterizations of that path, or in other words, the whole “fiber bundle” of path velocities above the path at hand – at a computation cost only slightly higher than that of checking one time-parameterization (see Section $2.3$). Granted that path velocity encodes important information about possible violations of the dynamics constraints as argued in the Introduction, this full and free (as in free beer) exploration enables significant performance gains.
Future works  We have recently extended TOPP to redundantly-actuated systems, which include in particular humanoid robots in multi-contact tasks [Pham and Stasse, 2014]. This enables AVP-based planners to be applied to multi-contact planning for humanoid robots. In this application, the existence of kinematic closure constraints (the parts of the robot in contact with the environment should remain fixed) makes path-velocity decomposition highly appealing since these constraints can be handled by a kinematic planner independently from dynamic constraints (torque limits, balance, etc.) In a preliminary experiment, we have planned a non-quasi-statically-feasible but dynamically-feasible motion for a humanoid robot (see video at [http://youtu.be/PkDSHodmvxY](http://youtu.be/PkDSHodmvxY)). Going further, we are currently investigating how AVP-based planners can enable existing quasi-static multi-contact planning methods [Hauser et al., 2008; Escande et al., 2013] to discover truly dynamic motions for humanoid robots with multiple contact changes.

Acknowledgments

We are grateful to Prof. Zvi Shiller for inspiring discussions about the TOPP algorithm and kinodynamic planning. This work was supported by a JSPS postdoctoral fellowship, by a Start-Up Grant from NTU, Singapore, and by a Tier 1 grant from MOE, Singapore.

A  Comparison of AVP-RRT with $K$NN-RRT on the double pendulum with severe torque limits

In this Appendix, we detail the implementation of the standard state-space planner $K$NN-RRT and the comparison of this planner with AVP-RRT.

A.1  $K$NN-RRT

A.1.1  Overall Algorithm

Our implementation of RRT in the state-space [LaValle and Kuffner, 2001] is detailed in Boxes 4 and 5.

**Box 4: $K$NN-RRT($x_{init}$, $x_{goal}$)**

1: $T$.INITIALIZE($x_{init}$)
2: for rep = 1 to $N_{max,rep}$ do
3:     $x_{rand} \leftarrow$ RANDOM_STATE() if mod(rep,5) $\neq$ 0 else $x_{goal}$
4:     $x_{new} \leftarrow$ EXTEND($T$, $x_{rand}$)
5:     $T$.ADD_VERTEX($x_{new}$)
6:     $x_{new2} \leftarrow$ EXTEND($x_{new}$, $x_{goal}$)
7:     if $d(x_{new}, x_{goal}) \leq \epsilon$ or $d(x_{new2}, x_{goal}) \leq \epsilon$ then
8:         return Success
9:     end if
10:    end for
11:   return Failure
Steer-to-goal frequency: we asserted the efficiency of the following strategy: every five extension attempts, try to steer directly to $x_{\text{goal}}$ (by setting $x_{\text{rand}} = x_{\text{goal}}$ on line 3 of Box 4). See also the discussion in [LaValle and Kuffner 2001], p. 387, about the use of uni-directional and bi-directional RRTs. We observed that the choice of the steer-to-goal frequency (every 5, 10, etc., extension attempts) did not significantly alter the performance of the algorithm, except when it is too large, e.g. once every two extension attempts.

Metric: the metric for the neighbors search in EXTEND (Box 5) and to assess whether the goal has been reached (line 7 of Box 4) was defined as:

$$d(x_a, x_b) = d((q_a, v_a), (q_b, v_b)) = \frac{\sum_{j=1,2} \sqrt{1 - \cos(q_{aj} - q_{bj})}}{4} + \frac{\sum_{j=1,2} |v_{aj} - v_{bj}|}{4V_{\text{max}}},$$

where $V_{\text{max}}$ denotes the maximum velocity bound set in the random sampler (function RANDOM_STATE() in Box 4). This simple metric is similar to an Euclidean metric but takes into account the periodicity of the joint values.

Termination condition: we defined the goal area as a ball of radius $\epsilon = 10^{-2}$ for the metric (9) around the goal state $x_{\text{goal}}$. As an example, $d(x_a, x_b) = \epsilon$ corresponds to a maximum angular difference of $\Delta q_1 \approx 0.057$ rad $\approx 3.24$ degrees in the first joint.

This choice is connected to that of the integration time step (used e.g. in Forward Dynamics computations in section A.1.2), which we set to $\delta t = 0.01$ s. Indeed, the average angular velocities we observed in our benchmark was around $\bar{V} = 5$ rad.s$^{-1}$ for the first joint, which corresponds to an average instantaneous displacement $\bar{V} \cdot \delta t \approx 5.10^{-2}$ rad of the same order as $\Delta q_1$ above.

Nearest-neighbor heuristic: instead of considering only extensions from the nearest neighbor, as has commonly been done, we considered the “best” extension from the $K$ nearest neighbors (line 5 in Box 5), i.e. the extension yielding the state closest to $x_{\text{rand}}$ for the metric $d$ (cf. Equation (9)).

A.1.2 Local steering

Regarding the local steering scheme (STEER on line 3 of Box 5), there are two main approaches corresponding to the complementary sides of the equations of motion: state-based and control-based steering [Caron et al. 2014].
Control-based Steering: in this approach, a control input $\tau(t)$ is computed first. It generates a given trajectory computable by forward dynamics. Because $\tau(t)$ is computed beforehand, there is no direct control on the end-state of the trajectory. To palliate this, the function $\tau(t)$ is then updated, with or without feedback on the end-state, until some satisfactory result is obtained or a computation budget is exhausted. For example, in works such as LaValle and Kuffner [2001], Hsu et al. [2002], random functions $u$ are sampled from the set of piecewise-constant functions. A number of them are tried and only the one bringing the system closest to the target is retained. Linear-Quadratic Regulation [Perez et al., 2012, Tedrake, 2009] is another example of control-based steering where the function $u$ is computed as the optimal policy for a linear approximation of the system dynamics (given a quadratic cost function).

In the present work, we followed the control-based approach from LaValle and Kuffner [2001], Hsu et al. [2002], as described by Box 6. The random control is a stationary $(\tau_1, \tau_2)$ sampled as:

$$(\tau_1, \tau_2) \sim \mathcal{U}([\tau_{1 \text{max}}, \tau_{1 \text{max}}] \times [-\tau_{2 \text{max}}, \tau_{2 \text{max}}]).$$

where $\mathcal{U}$ denotes uniform sampling from a set. The random time duration $\Delta t$ is sampled uniformly in $[\delta t, \Delta t_{\text{max}}]$ where $\Delta t_{\text{max}}$ is the maximum duration of local trajectories (parameter to be tuned), and $\delta t$ is the time step for the forward dynamics integration (set to $\delta t = 0.01 \text{ s}$ as discussed in Section A.1.1). The number of local trajectories to be tested, $N_{\text{local_trajs}}$, is also a parameter to be tuned.

**Box 6: STEER($x_{\text{near}}, x_{\text{rand}}$)**

```plaintext
1: for $p = 1$ to $N_{\text{local_trajs}}$ do
2: $u \leftarrow$ RANDOM_CONTROL($\tau_{1 \text{max}}, \tau_{2 \text{max}}$)
3: $\Delta t \leftarrow$ RANDOM_DURATION($\Delta t_{\text{max}}$)
4: $x^p \leftarrow$ FORWARD_DYNAMICS($x_{\text{near}}, u, \Delta t$)
5: end for
6: return $\arg\min_p d(x^p, x_{\text{rand}})$
```

State-based Steering: in this approach, a trajectory $\tilde{q}(t)$ is computed first. For instance, $\tilde{q}$ can be a Bezier curve matching the initial and target configurations and velocities. The next step is then to compute a control that makes the system track it. For fully- or over-actuated system, this can done using inverse dynamics. If no suitable controls exist, the trajectory is rejected. Note that both the space $\mathcal{I}(\tilde{q})$ and timing $t$ impact the dynamics of the system, and therefore the existence of admissible controls. Bezier curves or B-splines will conveniently solve the spatial part of the problem, but their timing is arbitrary, which tends to result in invalid controls and needs to be properly cared for.

To allow meaningful comparisons with AVP-RRT, we considered the simple state-based steering described in Box 7. Trying to design the best possible nonlinear controller for the double pendulum would be out of the scope of this work, as it would imply either problem-specific tunings or substantial modifications to the core RRT algorithm (as done e.g. in Perez et al. [2012]).
Here, INTERPOLATE\((T, x_{\text{near}}, x_{\text{rand}})\) returns a third-order polynomial \(P_i(t)\) such that \(P_i(0) = q_{ai}\), \(P'_i(0) = v_{ai}\), \(P_i(T) = q_{bi}\), \(P'_i(T) = v_{bi}\), and our local planner tries 10 different values of \(T\) between 0.01 s and 2 s. We use inverse dynamics at each time step of the trajectory to check if a control \(\tilde{\tau}(t)\) is within torque limits. The trajectory is cut at the first inadmissible control.

Comparing the two approaches: on the pendulum, state-based steering yielded RRTs with much slower exploration speeds (and, as a consequence, much higher search time) compared with the control-based steering, as illustrated in Figure 9. This slowness is likely due to the uniform sampling in a wide velocity range \([-V_{\text{max}}, V_{\text{max}}]\) which resulted in trajectories requiring high torques, most of which were consequently discarded.

Let us remark here that, although AVP-RRT follows the state-based paradigm (it indeed interpolates paths in configuration space and then computes feasible velocities along the path using Bobrow-like approach, which includes inverse dynamics computations), it is much more successful. The reason for this lies precisely in AVP: when the interval of feasible velocities is small, a randomized approach will have a high probability of sampling unreachable velocities. Therefore, it will fail most of the time. Using AVP, the set of reachable velocities is exactly computed and this failure factor disappears. With AVP-RRT, failures only occur from “unlucky” sampling in the configuration space. Note however that the algorithm only saves and propagates the norm of the velocity vectors, not their directions, which may make the algorithm probabilistically incomplete (cf. discussion in Section 5).

A.1.3 Fine-tuning of KNN-RRT

Based on the above results, we now focus on KNN-RRTs with “sampling-based” steering for the remainder of this section. The parameters to be tuned are:

- \(N_{\text{local,trajs}}\): number of local trajectories tested in each call to STEER;
- \(\Delta t_{\text{max}}\): maximum duration of each local trajectory.

The values we tested for these two parameters are summed up in Table 3. The parameters we do not tune are:
Figure 9: Comparison of control-based and state-based steering for $K = 1$ (left-top), $K = 10$ (right-top), $K = 40$ (left-bottom) and $K = 100$ (right-bottom). The X-axis represents the angle of the first joint and the Y-axis its velocity. The trees grown by the state-based and control-based methods are in red and blue, respectively. The goal area is depicted by the red ellipse on the left side.

| Number of trials | $N_{\text{local,trajs}}$ | $\Delta t_{\text{max}}$ |
|------------------|--------------------------|-------------------------|
| 10               | 1                        | 0.2                     |
| 10               | 30                       | 0.2                     |
| 10               | 80                       | 0.2                     |
| 20               | 20                       | 0.5                     |
| 20               | 20                       | 1.0                     |
| 20               | 20                       | 2.0                     |

Table 3: Parameter sets for each test.
• Maximum velocity $V_{\text{max}}$ for sampling velocities. We set $V_{\text{max}} = 50 \text{ rad.s}^{-1}$, which is about twice the maximum velocity observed in the successful trials of AVP-RRT;

• Number of neighbors $K$. In this tuning phase, we set $K = 10$. Other values of $K$ will be tested in the final comparison with AVP in section A.2;

• Space-time precision ($\epsilon, \delta t$): as discussed in Section A.1.1 we chose $\epsilon = 0.01$ and $\delta t = 0.01 \text{ s}$.

Finally, in this tuning phase, we set the torque limit as $(\tau_{1}^{\text{max}}, \tau_{2}^{\text{max}}) = (13, 7) \text{ N.m}$, which are relatively “slack” values, in order to obtain faster termination times for RRT. Tighter values such as $(\tau_{1}^{\text{max}}, \tau_{2}^{\text{max}}) = (11, 5) \text{ N.m}$ will be tested in our final comparison with AVP-RRT in section A.2.

![Figure 10](image)

Figure 10: Minimum distance to the goal as a function of time for different values of $N_{\text{local traj}}$, $\Delta t_{\text{max}}$. At each instant, the minimum distance of the tree to the goal is computed. The average of this value across the 10 trials of each set is drawn in bold, while shaded areas indicate standard deviations. A: tuning of $N_{\text{local traj}}$. B: tuning of $\Delta t_{\text{max}}$.

Fig. 10A shows the result of simulations for different values of $N_{\text{local traj}}$. One can note that the performance of RRT is similar for values 10 and 30, but gets worse for 80. Based on this observation, we chose $N_{\text{local traj}} = 20$ for the final comparison in section A.2.

Fig. 10B shows the simulation results for various values of $\Delta t_{\text{max}}$. Observe that the performance of RRT is similar for the three tested values, with smaller values (e.g. 0.5 s) performing better earlier in the trial and larger values (e.g. 2.0 s) performing better later on. We also noted that smaller values of $\Delta t_{\text{max}}$ such as 0.1 s or 0.2 s tended to yield poorer results (not shown here). Our choice for the final comparison was thus $\Delta t_{\text{max}} = 1.0 \text{ s}$.

A.2 Comparing $K$NN-RRT and AVP-RRT

In this section, we compare the performance of $K$NN-RRT (for $K \in \{1, 10, 40, 100\}$, the other parameters being set to the values discussed in the previous section) against
AVP-RRT with 10 neighbors. For practical reasons, we further limited the execution time of every trial to $10^4$ s, which had no impact in most cases or otherwise induced a slight bias in favor of RRT (since we took $10^4$ s as our estimate of the “search time” when RRT does not terminate within this time limit).

We ran the simulations for two instances of the problem, namely

- $(\tau_{1\text{max}}, \tau_{2\text{max}}) = (11, 7)$ N.m;
- $(\tau_{1\text{max}}, \tau_{2\text{max}}) = (11, 5)$ N.m.

For each problem instance, we ran 40 trials for each planner AVP-RRT, state-space RRT with 1 nearest neighbor (RRT-1), RRT-10, RRT-40 and RRT-100. Note that for each trial $i$, all the planners received the same sequence of random states

$$X_i = \left\{ x_{\text{rand}}^{(i)}(t) \in \mathbb{R}^4 \mid t \in \mathbb{N} \right\} \sim \mathcal{U}\left((-\pi, \pi)^2 \times [-V_{\text{max}}, +V_{\text{max}}]^2 \mathbb{N}\right),$$

although AVP-RRT only used the first two coordinates of each sample since it plans in the configuration space. The results of this benchmark were already illustrated in Fig. 7. Additional details are provided in Tables 4 and 5. All trials of AVP successfully terminated within the time limit.

For $(\tau_{1\text{max}}, \tau_{2\text{max}}) = (11, 7)$, the average search time was 3.3 min. Among the $K$NN-RRT, RRT-40 performed best with a success rate of 92.5% and an average computation time ca. 45 min, which is however 13.4 times slower than AVP-RRT.

For $(\tau_{1\text{max}}, \tau_{2\text{max}}) = (11, 7)$, the average search time was 9.8 min. Among the $K$NN-RRT, again RRT-40 performed best in terms of search time (54.6 min on average, which was 5.6 times slower than AVP-RRT), but RRT-100 performed best in terms of success rate within the $10^4$s time limit (92.5%).

| Planner  | Success rate | Search time (min) |
|----------|--------------|-------------------|
| AVP-RRT  | 100%         | 3.3±2.6           |
| RRT-1    | 40%          | 70.0±34.1         |
| RRT-10   | 82.5%        | 53.1±59.5         |
| RRT-40   | 92.5%        | 44.6±42.6         |
| RRT-100  | 82.5%        | 88.4±54.0         |

Table 4: Comparison of AVP-RRT and $K$NN-RRT for $(\tau_{1\text{max}}, \tau_{2\text{max}}) = (11, 7)$.

| Planner  | Success rate | Search time (min) |
|----------|--------------|-------------------|
| AVP-RRT  | 100%         | 9.8±12.1          |
| RRT-1    | 47.5%        | 63.8±36.6         |
| RRT-10   | 85%          | 56.3±60.1         |
| RRT-40   | 87.5%        | 54.6±52.2         |
| RRT-100  | 92.5%        | 81.2±46.7         |

Table 5: Comparison of AVP-RRT and $K$NN-RRT for $(\tau_{1\text{max}}, \tau_{2\text{max}}) = (11, 5)$. 
References

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

J.E. Bobrow, S. Dubowsky, and JS Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.

Francesco Bullo and Kevin M Lynch. Kinematic controllability for decoupled trajectory planning in underactuated mechanical systems. *IEEE Transactions on Robotics and Automation*, 17(4):402–412, 2001.

Stéphane Caron, Quang-Cuong Pham, and Yoshihiko Nakamura. Completeness of randomized kinodynamic planners with state-based steering. In *Proceedings of the International Conference on Robotics and Automation*, pages 5818–5823, 2014.

R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. URL [http://www.programmingvision.com/rosen_diankov_thesis.pdf](http://www.programmingvision.com/rosen_diankov_thesis.pdf).

B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.

Adrien Escande, Abderrahmane Kheddar, and Sylvain Miossec. Planning contact points for humanoid robots. *Robotics and Autonomous Systems*, 61(5):428–442, 2013.

Kris Hauser. Fast interpolation and time-optimization with contact. *The International Journal of Robotics Research*, 33(9):1231–1250, 2014.

Kris Hauser, Timothy Bretl, Jean-Claude Latombe, Kensuke Harada, and Brian Wilcox. Motion planning for legged robots on varied terrain. *The International Journal of Robotics Research*, 27(11-12):1325–1349, 2008.

David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

David Hsu, Tingting Jiang, John Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Robotics and Automation, 2003. Proceedings. ICRA’03. IEEE International Conference on*, volume 3, pages 4420–4426. IEEE, 2003.

J. Johnson and K. Hauser. Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path. In *IEEE International Conference on Robotics and Automation*, pages 2035–2041, 2012.

M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.
Kamal Kant and Steven W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, 1986.

L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, 2000.

J.J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, 12(1):105–118, 2002.

S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

Puttichai Lertkultanon and Quang-Cuong Pham. Dynamic non-prehensile object transportation. In *IJCAI*, 2014. URL [http://www.ntu.edu.sg/home/cuong/docs/nonprehensile.pdf](http://www.ntu.edu.sg/home/cuong/docs/nonprehensile.pdf).

Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):43, 2012.

Yoshihiko Nakamura and Ranjan Mukherjee. Nonholonomic path planning of space robots via a bidirectional approach. *Robotics and Automation, IEEE Transactions on*, 7(4):500–514, 1991.

Jufeng Peng and Srinivas Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *The International Journal of Robotics Research*, 24(4):295–310, 2005.

A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *IEEE International Conference on Robotics and Automation*, 2012.

Q.-C. Pham. A general, fast, and robust implementation of the time-optimal path parameterization algorithm. *IEEE Transactions on Robotics (to appear)*, 2014. doi: 10.1109/TRO.2014.2351113. URL [http://arxiv.org/abs/1312.6533](http://arxiv.org/abs/1312.6533).

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*, pages 165–170, 2012.

Quang-Cuong Pham and Olivier Stasse. Time-optimal path parameterization for overactuated robots (numerical integration approach), 2014. URL [http://www.ntu.edu.sg/home/cuong/docs/overactuated.pdf](http://www.ntu.edu.sg/home/cuong/docs/overactuated.pdf).
Quang-Cuong Pham, Stéphane Caron, and Yoshihiko Nakamura. Kinodynamic planning in the configuration space via velocity interval propagation. In *Robotics: Science and System*, 2013.

Michael Posa and Russ Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Algorithmic Foundations of Robotics X*, pages 527–542. Springer, 2013.

N. Ratliff, M. Zucker, J.A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation*, pages 489–494, 2009.

Z. Shiller and S. Dubowsky. On the optimal control of robotic manipulators with actuator and end-effector constraints. In *IEEE International Conference on Robotics and Automation*, pages 614–620, 1985.

Z. Shiller and Y.R. Gwo. Dynamic motion planning of autonomous vehicles. *IEEE Transactions on Robotics and Automation*, 7(2):241–249, 1991.

Z. Shiller and H.H. Lu. Computation of path constrained time optimal motions with dynamic singularities. *Journal of dynamic systems, measurement, and control*, 114:34, 1992.

K. Shin and N. McKay. Selection of near-minimum time geometric paths for robotic manipulators. *IEEE Transactions on Automatic Control*, 31(6):501–511, 1986.

Thierry Siméon, Stéphane Leroy, and J-P Laumond. Path coordination for multiple mobile robots: a resolution-complete algorithm. *Robotics and Automation, IEEE Transactions on*, 18(1):42–49, 2002.

J.J.E. Slotine and H.S. Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Transactions on Robotics and Automation*, 5(1):118–124, 1989.

W. Suleiman, F. Kanehiro, E. Yoshida, J.P. Laumond, and A. Monin. Time parameterization of humanoid-robot paths. *IEEE Transactions on Robotics*, 26(3):458–468, 2010.

R. Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, 2009.

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*, 54(10):2318–2327, 2009.

M. Vukobratovic, B. Borovac, and D. Sordilovic. Zero-moment point–proper interpretation and new applications. In *IEEE/RAS International Conference on Humanoid Robots*, 2001.

L. Zlajpah. On time optimal path control of manipulators with bounded joint velocities and torques. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1572–1577. IEEE, 1996.