2018 IEEE Conference on Decision and Control (CDC)
Miami Beach, FL, USA, Dec. 17-19, 2018

Asymptotically Optimal Pruning for Nonholonomic Nearest-Neighbor Search

Valerio Varricchio and Emilio Frazzoli*

***Abstract*— Nearest-Neighbor Search (NNS) arises as a key component of sampling-based motion planning algorithms and it is known as their asymptotic computational bottleneck. Algorithms for exact Nearest-Neighbor Search rely on explicit distance comparisons to different extents. However, in motion planning, evaluating distances is generally a computationally demanding task, since the metric is induced by the minumum cost of *steering* a dynamical system between states. In the presence of driftless nonholonomic constraints, we propose efficient *pruning* techniques for the k-d tree algorithm that drastically reduce the number of distance evaluations performed during a query. These techniques exploit computationally convenient lower and upper bounds to the geodesic distance of the corresponding sub-Riemannian geometry. Based on asymptotic properties of the reachable sets, we show that the proposed pruning techniques are optimal, modulo a constant factor, and we provide experimental results with the Reeds-Shepp vehicle model.**

I. Introduction

In robotics, a great amount of research has been devoted to Motion Planning over the past decades. Sampling-based algorithms, such as PRMs [1] and RRTs [2] – as well as their asymptotically optimal counterparts, PRM* and RRT* [3] – represented major breakthroughs in the field and are widely used today. These algorithms share a common approach: they build a graph of motions by connecting randomly sampled configurations of the robot.

A crucial building block of sampling-based algorithms is Nearest-Neighbor Search, which is used – during the graph expansion – to direct the computational effort to the most relevant vertices in the graph. Noticeably, Nearest-Neighbor Search is the weakest asymptotic link of sampling-based planners, as the number of samples increases. While this bottleneck was believed purely theoretical as compared to collision-checking, recent work has shown that the latter can be made asymptotically irrelevant with tailored data structures [4], and that the *practical* computational role of Nearest-Neighbor Search is in fact far from negligible for a class of sensitive problems [5].

As compared to naive approaches, such as linear search, efficient Nearest-Neighbor algorithms have been identified as crucial to the performance of sampling-based planners [6]. In particular, k-d trees [7] are a popular choice in Motion Planning due to their remarkable effectiveness in relatively low-dimensional spaces. However, k-d trees are traditionally limited to the Euclidean metric, or equivalent ones (i.e., L_p metrics) [8]. Nevertheless, these metrics are arguably rarely appropriate in Motion Planning: most systems of interest are not able to follow straight lines in the Euclidean sense, since they are subject to *nonholonomic differential constraints*. These constraints locally limit the types of motion attainable, yet do not necessary compromise the system's controllability, i.e., the capability of moving between arbitrary configurations.

Previous work [9] has been focused on *generalizing* the k-d tree *construction* and *query* algorithms to correctly cope with the metrics induced by nonholonomic constraints.

In this paper, we instead focus on *improving* the performance of the k-d tree *query* algorithm. The enabling remark can be exposed as follows: in the presence of nonholonomic constraints, evaluating distances is generally a computationally demanding task. In fact, for the majority of cases, it entails solving a two-point boundary value problem. On the other hand, all the techniques for exact Nearest-Neighbor Search must rely at some point on explicit distance computations.

To address this, we propose pruning techniques aimed at *reducing the number of distance evaluations* necessary to discern the nearest neighbors, based on heuristics that approximate the true nonholonomic metric. From a control theory perspective, finding heuristics for a given control cost requires ingenuity and it is typically an ad-hoc process. General approaches to synthesize bounds for the cost of steering dynamical systems are object of recent research, ranging from learning-based approaches [10] to algebraic techniques such as sum-of-squares programming [11]. However, there are generally no theoretical guarantees as to the pruning effectiveness of the produced heuristics.

Focusing on driftless nonholonomic systems, in this work we address the questions: i) what are *good* heuristics for these systems? and ii) how can we generate them?

Along the lines of [9], we exploit results from sub-Riemannian geometry to identify a specific class of heuristics that are both computationally inexpensive and guarantee optimal asymptotic pruning effectiveness, up to a constant factor.

II. Preliminaries

A. Asymptotic properties of nonholonomic metrics

The basic concepts reviewed in this section are used to study nonholonomic differential constraints. For a formal introduction, we suggest reading [12], [13], while here we summarize the key notions required for our work and introduce our notations.

*The authors are with the Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, USA. Emails: valerio@mit.edu, frazzoli@mit.edu

1) Elements of sub-Riemannian geometry: Differential geometry is concerned with the study of *manifolds*, an abstract generalization of curves (1-manifolds) and surfaces (2-manifolds) to arbitrary dimensions. Informally, a *smooth k-dimensional manifold $\mathcal{M}$* is a set of points that "looks" like a Euclidean vector space $\mathbb{R}^k$, in the neighborhood of each point $p \in \mathcal{M}$. For instance, a spherical surface is two-dimensional manifold just like Earth's globe locally resembles a plane. This local vector space at a point $p \in \mathcal{M}$, denoted $T_p\mathcal{M}$, is named the *tangent space at p*.

A *geometry* is generated by endowing a manifold with a metric. A crucial distinction arises between *Riemannian* vs. *sub-Riemannian* geometries. If we temporarily think of the distance as the result of "counting the steps" taken while walking between two points on a manifold, then a *Riemannian* geometry allows steps in all directions around a given point, i.e., arbitrarily on the tangent space. In contrast, a *sub-Riemannian* geometry allows steps *only in a vector subspace* – called a *fiber* – of the tangent space around each point of the manifold. The collection of all "allowed subspaces" on a sub-Riemannian manifold is called the *horizontal distribution*, denoted $\mathcal{H}$ and the fiber of $\mathcal{H}$ at a given point p is denoted $\mathcal{H}_p$.

A Riemannian geometry can be seen as a special case of a sub-Riemannian geometry where the allowed subspace coincides with the tangent space at all points, i.e., $\mathcal{H} = T\mathcal{M}$.

Definition 1. *A sub-Riemannian geometry $\mathcal{G}$ on a smooth manifold $\mathcal{M}$ is a tuple $\mathcal{G} = (\mathcal{M}, \mathcal{H}, \langle\cdot,\cdot\rangle_{\mathcal{H}})$, where $\mathcal{H}$ is a distribution on $\mathcal{M}$ whose fibers at all points p are equipped with the inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}} : \mathcal{H}_p \times \mathcal{H}_p \to \mathbb{R}$.*

A smooth curve $\gamma : [0,1] \to \mathcal{M}$ is said *horizontal* if $\dot\gamma(t) \in \mathcal{H}_{\gamma(t)}$ for all $t \in [0,1]$. Through the inner product, the *length* of a smooth curve is defined as: $\ell_{\mathcal{G}}(\gamma) := \int_0^1 \sqrt{\langle\dot\gamma(t), \dot\gamma(t)\rangle_{\mathcal{H}}}\, dt$. Finally, a metric is defined with the length of horizontal geodesics, i.e., $d_{\mathcal{G}}(a,b) := \inf_{\gamma\in\Gamma_a^b} l(\gamma)$ is the *sub-Riemannian distance* between two points $a, b \in \mathcal{M}$, where Γ_a^b is the set of horizontal curves such that $\gamma(0) = a$ and $\gamma(1) = b$. As familiar, one can then define the balls associated to this notion of distance:

Definition 2. *The sub-Riemannian ball centered at p of radius r is defined as: $\mathcal{B}(p,r) := \{p' \in \mathcal{M} : d_{\mathcal{G}}(p,p') < r\}$.*

2) Connection with nonholonomic systems.: In this section, we clarify how the mathematical concepts reviewed above describe the class of dynamical systems of our interest.

Definition 3. *A driftless nonholonomic system is a dynamical system of the form $\dot x(t) = \sum_{i=1}^h g_i(x(t))\, u_i(t)$, where $x(t) \in \mathcal{X}$, $u(t) \in \mathcal{U}$. The sets $\mathcal{X}, \mathcal{U}$ – respectively of dimension k and h – are referred to as the configuration and input manifolds.*

As per Definition 3, the local mobility of these systems is limited to linear combinations of the *control vector fields* $g_1, \dots, g_h$. In other words, the dynamics resemble an horizontal distribution $\mathcal{H}$ on the configuration manifold, with fibers $\mathcal{H}_x := Span(\{g_i(x)\}_{i=1}^h)$.

However, while the velocities of the system are geometrically restricted to $\mathcal{H}_x$, a sub-Riemannian geometry *shall not prescribe restrictions on their sign*. Therefore, we will consider the following additional requirement:

Definition 4. *A system is* symmetric *iff its input manifold satisfies $\mathcal{U} = -\mathcal{U}$, where $-\mathcal{U} := \{-u : u \in \mathcal{U}\}$.*

The latter ensures that the lengths of geodesics induce a proper *metric*, since the trajectories of a symmetric system are time-revertible.

The techniques introduced in this paper are general enough to encompass any *driftless, controllable nonholonomic system* (see Theorem 1 below). However, for illustrative purposes, we use the the Reeds-Shepp vehicle [14] as an example throughout the paper.

Example 1 (Reeds-Shepp vehicle)**.** *This model describes the motion of a wheeled vehicle on a plane, allowed to move both forward and backward, with a minimum steering radius (assumed =1 for convenience). This system has a three-dimensional configuration manifold, $\mathcal{X} = SE(2)$, with coordinates $x = (x_1, x_2, x_3)$ and its dynamics is according to Definition 3, with $g_1(x) = \hat f(x) := (\cos(x_3), \sin(x_3), 0)$, $g_2(x) = \hat\theta(x) := (0,0,1)$, on the symmetric control domain $\mathcal{U} : \{(u_1,u_2) : |u_1| \le 1,\ |u_2| \le |u_1|\}$. Note that $\hat f(x)$ indicates the longitudinal axis of the vehicle, while $\hat\theta(x)$ represents the heading axis. For completeness, let $\hat l(x) = (-\sin(x_3), \cos(x_3), 0)$ denote the lateral axis. At each $x \in SE(2)$ the fiber of the Reeds-Shepp distribution is $\mathcal{H}_x^{RS} = Span\{g_1(x),\ g_2(x)\}$. The dynamically feasible paths – with bounded curvature ($\chi \le 1$) – are horizontal curves for the associated sub-Riemannian geometry. In [14], the geodesics – called Reeds-Shepp curves – are found in closed form and proved to be a sequence of straight lines and maximum-curvature arcs of circle that can be built through elementary geometric constructions.*

3) Controllability and shape of reachable sets: A system is said *controllable* if any pair of configurations can be connected with a feasible (horizontal) path. Determining the controllability of a nonholonomic system is nontrivial. For example, the Reeds-Shepp vehicle cannot move directly in the lateral direction, however intuition suggests that an appropriate sequence of motions can result in a lateral displacement (e.g., in parallel parking).

A crucial tool for the following analysis is the *Lie bracket operator*. The Lie bracket of two vector fields g_1 and g_2 – denoted $[g_1, g_2]$ – tells whether the result of consecutive motions along them depends on the order of execution. Loosely speaking, if the order of execution does not matter ($[g_1, g_2] = 0$) any permutation of controls yields the same displacement, so there is no reason to prefer a given sequence to another. If the order matters ($[g_1, g_2] \neq 0$), then one can wisely exploit different cyclic sequences to move in different directions across the manifold.

Increasingly complex cyclic sequences can be explored systematically by nesting the Lie bracket operator in an iterative fashion, e.g. $[g_1, [g_2, g_3]]$. In the sequel, we briefly review the formal presentation of this process and the most important results: Chow's Theorem (Theorem 1) and the Ball-box Theorem (Theorem 2), related to the controllability and the reachable sets of nonholonomic systems, respectively.

The *Lie derivative* of a vector field g at $p \in \mathcal{M}$ in the direction $v \in T_p\mathcal{M}$ is defined as $dg(p)v = \frac{d}{dt} g(\gamma(t))\big|_{t=0}$, where γ is a smooth curve starting in $p = \gamma(0)$ with velocity $v = \dot{\gamma}(0)$. Given two vector fields g_1, g_2 on $\mathcal{M}$, the *Lie bracket* $[g_1, g_2]$ is a vector field on $\mathcal{M}$ defined as $[g_1, g_2](p) = dg_2(p)g_1(p) - dg_1(p)g_2(p)$.

From the horizontal distribution $\mathcal{H}$, one can construct a sequence of distributions by iterating the Lie brackets of the generating vector fields, $g_1, g_2 \dots g_h$, with $h \leq k$. This sequence can be defined recursively as follows: $\mathcal{H}^1 = \mathcal{H}$ and $\mathcal{H}^{i+1} = \mathcal{H}^i \cup [\mathcal{H}, \mathcal{H}^i]$, where $[\mathcal{H}, \mathcal{H}^i]$ denotes the distribution given by the Lie brackets of each generating vector field of $\mathcal{H}$ with those of $\mathcal{H}^i$.

Note that the Lie bracket can yield a vector field linearly independent from its operands, hence $\mathcal{H}^i \subseteq \mathcal{H}^{i+1}$. The *Lie hull* [13], denoted $Lie(\mathcal{H})$, is the limit of the sequence $\mathcal{H}^i$ as $i \to \infty$. A distribution $\mathcal{H}$ is said to be *involutive* or *bracket-generating* iff $Lie(\mathcal{H}) = T\mathcal{M}$.

Theorem 1. *(Chow's Theorem [13, p. 44]). Let $\mathcal{H}$ be a bracket-generating distribution on a connected manifold $\mathcal{M}$. Then any pair of points $a, b \in \mathcal{M}$ can be joined by a horizontal curve of $\mathcal{H}$.*

Example 2 (Lie hull of the Reeds-Shepp distribution)**.** *For every $x \in SE(2)$, the control vector fields $\hat{f}, \hat{\theta}$ span a two-dimensional subspace of $T_x(SE(2))$. Their Lie bracket is given by $[\hat{f}, \hat{\theta}](x) = \left(\frac{d}{dx}\hat{f}(x)\right)\hat{\theta}(x) - \left(\frac{d}{dx}\hat{\theta}(x)\right)\hat{f}(x) = (-\sin(x_3), \cos(x_3), 0)$, which in turn coincides with the body-frame lateral axis $\hat{l}(x)$ of the vehicle. Thus, the second-order distribution $\mathcal{H}^2_x = Span\{\hat{f}(x), \hat{\theta}(x), [\hat{f}, \hat{\theta}](x)\}$ coincides with the tangent bundle of $SE(2)$, and the Lie hull is obtained in the second step using the basis $y_1(x) = \hat{f}(x)$, $y_2(x) = \hat{\theta}(x)$, $y_3(x) = [y_1, y_2](x) = \hat{l}(x)$, referred to as the Philip-Hall basis [15, §15.4]. By Theorem 1 there is a feasible motion connecting any two configurations, which is consistent with one's intuition about the wheeled robot.*

Above, we have shown that from a basis $g_1, \dots, g_h$ of a bracket-generating distribution $\mathcal{H}$, one can define a basis $y_1, \dots, y_k$ of $T\mathcal{M}$ – the Philip-Hall basis – using the Lie bracket operator iteratively. Specifically $y_i = g_i$ for $i \leq h$, while the remaining $k - h$ fields are obtained with the appropriate Lie brackets that increment the dimensionality of $\mathcal{H}^i$ at each step.

The sub-Riemannian *weights* w_i are then defined as the smallest order of Lie brackets required to generate y_i from the original basis $g_1, \dots, g_h$. More formally, w_i is such that $y_i \notin \mathcal{H}^{w_i - 1}$ and $y_i \in \mathcal{H}^{w_i}$ for all i.

Under the conditions of Theorem 1, for each point $p \in \mathcal{M}$, one can identify a system of *privileged coordinates centered in* p, denoted $\{Z^p_i\}_{i=1}^k$, $Z_i : \mathcal{M} \to \mathbb{R}$, such that the mobility along each coordinate – in a neighborhood of p – has the order given by the corresponding sub-Riemannian weight. Formally, $Z^p_i(p') \in O(t^{w_i})$, as $t \to 0$, with $t := d(p, p')$ (see [12, §2.1.2]). This asymptotic behavior of privileged coordinates is related to the intuition that *arbitrarily small displacements in certain directions require increasingly longer paths than ones of the same magnitude along other directions*. Additionally, we call the set of vector fields $z_i(p) := dZ^p_i|_p$ for $i \in [1, ..., k]$ a system of *privileged axes at* p.

Although the above property can be used to check directly whether a coordinate system is privileged, techniques such as *Bellaiche's algorithm* [12, §2.1.2] are available to construct privileged coordinates from the Philip-Hall basis.

Once a system of privileged coordinates is identified, the *weighted box* at p of size ϵ and constants $\mu = [\mu_1, ...\mu_k]$ is defined as the set:

$$\mathrm{Box}^{w,\mu}(p, \epsilon) := \{p' \in \mathcal{M} : |Z^p_i(p')| < \mu_i \epsilon^{w_i}, \forall i \in [1, k]\}. \quad (1)$$

Theorem 2. *(The ball-box Theorem [13, p. 50]). Let $\mathcal{H}$ be an involutive distribution on a connected manifold $\mathcal{M}$. Then, there exist constants $\epsilon_0 \in \mathbb{R}_{>0}$ and $c, C \in \mathbb{R}^k_{>0}$ such that for all $\epsilon < \epsilon_0$ and for all $p \in \mathcal{M}$:*

$$\underbrace{\mathrm{Box}^{w,c}(p, \epsilon)}_{\mathrm{Box}^w_i(p,\epsilon)} \subset \mathcal{B}(p, \epsilon) \subset \underbrace{\mathrm{Box}^{w,C}(p, \epsilon)}_{\mathrm{Box}^w_o(p,\epsilon)}, \quad (2)$$

where Box^w_i and Box^w_o are referred to as the inner and outer sub-Riemannian boxes, respectively.

Example 3 (Ball-Box Theorem for a Reeds-Shepp vehicle)**.** *From the previous example, $\hat{f}(x), \hat{\theta}(x) \in \mathcal{H}^1$ so the corresponding weights are $w_{\hat{f}} = w_{\hat{\theta}} = 1$. Conversely, $\hat{l}(x)$ first appears in $\mathcal{H}^2$, so $w_{\hat{l}} = 2$. For this system, one can verify that $z_i(p) \equiv y_i(p)$ for all $i \in [1, ..., k]$ and for all $p \in \mathcal{M}$. Namely, the privileged axes coincide with the Philip-Hall basis. Theorem 2 states the existence of inner and outer boxes for the reachable sets as $t \to 0$ and predicts the infinitesimal order of each side of these boxes, as shown in Figure 1. Higher order Lie brackets correspond to sides that approach zero at a faster asymptotic rate. The longitudinal and angular sides — along $\hat{f}$ and $\hat{\theta}$ — scale with $\Theta(t)$, while the lateral one — along $\hat{l}$ — scales with $\Theta(t^2)$. Therefore, the boxes become increasingly flattened along $\hat{l}$ as $t \to 0$. Intuitively, this geometric feature of the sub-Riemannian boxes reflects the well known fact that a small lateral displacement of a car requires more time than an equivalent longitudinal one.*

Remark 1. *Theorem 2 is of crucial theoretical importance, since it establishes that sub-Riemannian metrics are fundamentally* not equivalent *to the Euclidean distance or to any L_p metric, in general. Specifically, for the latter metrics, balls shrink uniformly in all directions as their radius tends to zero. This behavior is typical of a Riemannian metric, i.e., one for which $h = k$ and $w_1 = w_2 = ... = w_k = 1$.*

For the Reeds-Shepp vehicle, the sides of both boxes can be computed explicitly. Their $\hat{\theta}$ extension is trivially

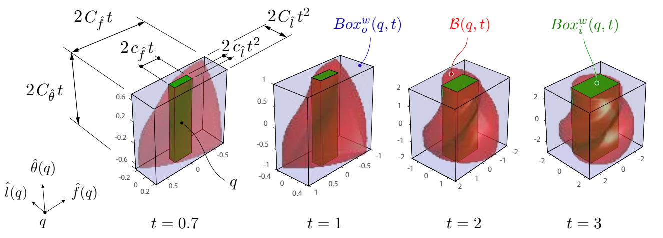


Fig. 1. Reachable sets and sub-Riemannian boxes for a Reeds-Shepp vehicle around a configuration q and for different values of t. The lengths of the box sides are highlighted. For both the inner and outer boxes, as $t \to 0$, the sides along the front $(\hat{f})$ and heading $(\hat{\theta})$ axes are linear in t, while the side along the lateral axis $(\hat{l})$ is quadratic in t.

determined by observing that one can keep a constant maximal angular velocity throughout a manoeuvre, i.e., with $|u_2(t)| \equiv 1$, and therefore $C_{\hat{\theta}} = c_{\hat{\theta}} = 1$, while the remaining sides can be found with elementary geometric considerations sketched out in Figure 2.

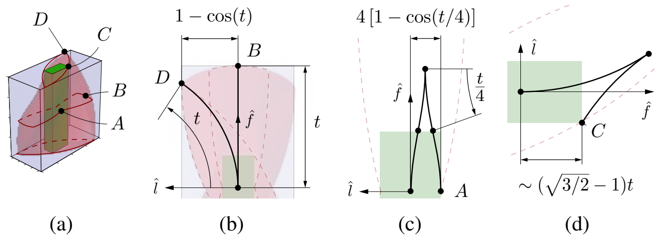


Fig. 2. Determining the constants for the weighted boxes of a Reeds-Shepp vehicle.

Specifically, in Figure 2(a), we highlight four extremal configurations – A, B, C and D – on the boundary of the reachable set in time t, which correspond to special manoeuvres of the vehicle. In Figure 2(b), we determine the half-sides of the outer box: B is reached through a pure forward motion and defines the maximum longitudinal displacement, yielding $C_{\hat{f}} = 1$, whereas D is reached through a pure turn, resulting in the maximal lateral displacement of $1 - \cos(t) \sim t^2/2$, i.e., $C_{\hat{l}} = 1/2$. In Figure 2(c), the lateral half-size of the inner box is determined through a parallel-parking manoeuvre towards A, composed of four arcs of circle of minimum-steering radius, each with length $t/4$, resulting in a lateral displacement of $4[1 - \cos(t/4)] \sim t^2/8$, yielding $c_{\hat{l}} = 1/8$. Last, in Figure 2(d) we show the manoeuvre that determines the front half-size of the inner box, i.e., a turn with gear switch yielding $c_{\hat{f}} = \sqrt{3/2} - 1$.

In addition to confirming the Ball-Box Theorem through elementary considerations for the Reeds-Shepp case, these results will later be used to construct computationally efficient heuristics that accurately approximate the Reeds-Shepp distance.

B. The k-d tree Algorithm

In this section we briefly review the k-d tree algorithm, a generalization of binary search trees to higher dimensions.

As typical of Nearest-Neighbor Search algorithms, a k-d tree operates by preprocessing a finite collection $X \in \mathbb{R}^k$ of data (build phase) in such a way that later, when a new "query" object q is given, one can efficiently identify the elements closer to to it (query phase). In the sequel, we review the fundamental properties of the k-d tree data structure and the m-nearest-neighbor query algorithm. For the build phase – beyond the scope of this work – we adopt an incremental construction algorithm, tailored to nonholonomic systems via a Lie splitting sequence according to previous work by the Authors (see [9, §5]).

1) Anatomy of a k-d tree: Each data point $x_i \in X$ is associated with a unitary normal vector $n_i \in \mathbb{R}^k$, $\|n_i\|_2 = 1$. Together, the pair $v_i = (x_i, n_i)$ defines a *vertex* of the binary tree and the set of all vertices is denoted with $\mathcal{V}$. Every vertex partitions $\mathbb{R}^k$ into two halfspaces, called the *positive* and *negative halfspace* and respectively described algebraically as:

$$\begin{aligned} \mathfrak{h}^+(x_i, n_i) &:= \left\{x \in \mathbb{R}^k : \langle n_i, x - x_i\rangle_{\mathbb{R}^k} > 0\right\}, \\ \mathfrak{h}^-(x_i, n_i) &:= \left\{x \in \mathbb{R}^k : \langle n_i, x - x_i\rangle_{\mathbb{R}^k} \leq 0\right\}, \end{aligned} \tag{3}$$

while the set $\mathfrak{h}(x_i, n_i) := \left\{x \in \mathbb{R}^k : \langle n_i, x - x_i\rangle_{\mathbb{R}^k} = 0\right\}$ is the *hyperplane* associated to v_i. An *edge* is an ordered pair of vertices $e = (v_i, v_j)$ and the k-d tree is defined as $\mathcal{T} := (\mathcal{V}, \mathcal{E}^-, \mathcal{E}^+)$, with $\mathcal{E}^-$ set of *left edges* and $\mathcal{E}^+$ set of *right edges*. Given an edge $e = (v_i, v_j)$, v_j is referred to as the *left child* of v_i if $e \in \mathcal{E}^-$, or the *right child* if $e \in \mathcal{E}^+$. Let $\texttt{parent}(v_i \in \mathcal{V}) = v_j \in \mathcal{V}$ s.t. $(v_i, v_j) \in \mathcal{E}^- \cup \mathcal{E}^+$. By convention, $\texttt{parent}(v_i) = \emptyset$ if such v_j does not exist and v_i is called the *root* of $\mathcal{T}$, denoted $v_i = \texttt{root}(\mathcal{T})$. Let $\texttt{child}(v_i \in \mathcal{V}, s \in \{-, +\}) = v_j \in \mathcal{V}$ s.t. $(v_i, v_j) \in \mathcal{E}^s$, or otherwise $\emptyset$ if such v_j does not exist.

The fundamental property of k-d trees is that *left children belong to the negative halfspace defined by their parent and right children to the positive halfspace*. Recursively, a vertex belongs to the halfspaces of all its ancestors. As a result, a k-d tree defines a *partition* of $\mathcal{M}$ into non-overlapping polyhedra, called *buckets* [8], that cover the entire manifold.

C. m-nearest-neighbor query algorithm

The following primitives will be used as building blocks of the query procedure, presented in Algorithm 1.

a) Side of hyperplane: Let $\texttt{sideOf}(x, p, n) \to \{-, +\}$, with $x, p \in \mathcal{M}$, $n \in \mathbb{R}^k$ be a procedure that returns $+$ iff $x \in \mathfrak{h}^+(p, n)$ and $-$ otherwise. For convenience, define $\texttt{opposite}(+) = -$ and $\texttt{opposite}(-) = +$.

b) Queue: The algorithm maintains a *Bounded Priority Queue*, Q of size m to collect the results. The queue can be thought of as a sequence $Q = [q_1, q_2, \ldots q_m]$, where each element q_i is defined as a distance-vertex pair $q_i : (d_i, v_i)$, $d_i \in \mathbb{R}_{\geq 0}$, $v_i \in \mathcal{V}$. The property $d_1 \leq d_2 \leq \cdots \leq d_m$ is an invariant of the data structure. When an element (d_{new}, v_{new}) is inserted in the queue, if $d_{new} < d_m$, then q_m is discarded and the indices of the remaining elements are rearranged to maintain the order. In the sequel, the $\cup$ operator denotes the insertion of new elements, with the internal discard / rearrange operations implicit in the notation.

c) Ball-Hyperplane Intersection: This is a procedure $\texttt{ballHyperplane}(x, R, p, n)$, with $x \in \mathcal{M}$ and $R \geq 0$ that determines whether a ball and a hyperplane definitely intersect. In the sequel, we adopt an implementation of this primitive presented in [9, §3.1], tailored to nonholonomic systems through the use of the outer sub-Riemannian box and shown to have optimal asymptotic behavior. Namely, we let the procedure return true iff $\text{Box}_o(x, R) \cap \mathfrak{h}(p, n) \neq \emptyset$.

Algorithm 1: The k-d tree query algorithm

```
define query ($q \in \mathcal{M}$, $\mathcal{T} = (\mathcal{V}, \mathcal{E}^-, \mathcal{E}^+)$):
    $Q \longleftarrow \{q_{1:m} = (\emptyset, \infty)\}$;
    define queryNode ($q \in \mathcal{M}$, $v_i = (x_i, n_i) \in \mathcal{V}$):
        if $v_i = \emptyset$ then return;
        $s \longleftarrow$ sideOf ($q$, $x_i$, $n_i$);
        queryNode ($q$, child($v_i$, $s$))
        Q $\longleftarrow Q \cup \{(\text{dist}(q, x_i), v_i)\}$
        if ballHyperplane ($q$, $d_m$, $x_i$, $n_i$) then
            queryNode ($q$, child($v_i$, opposite($s$)))
    queryNode ($q$, root ($\mathcal{T}$)) return $Q$;
```

III. Pruned k-d tree query algorithms

In most practical Motion Planning applications, once a k-d tree is built, the main computational bottleneck during query is the distance evaluation on line 7 in Algorithm 1. In fact, distances are defined as the optimal cost of steering the system between two configurations (see [16, §III]). In the most general setup, this requires solving a two-point boundary value problem numerically. In a few specific cases, the solution is known in closed form, however its computation usually entails a relatively high number of operations, as compared to the remaining components of Algorithm 1. For instance, to compute a distance in the Reeds-Shepp case, one has to construct 48 trajectories and compare their lengths [14].

In this section, we address the mentioned bottleneck by replacing calls to the distance function with calls to computationally efficient heuristics, H_l and H_u, whenever possible. These heuristics shall provide lower and upper bounds to the distance, respectively, i.e., $H_l(p, q) \leq d(p, q) \leq H_u(p, q)$ for all $p, q \in \mathcal{M}$. Although they are only approximations of the metric, if chosen carefully, H_l and H_u can render a significant number of distance evaluations unnecessary (pruning) at negligible computation cost.

Heuristics are generally found by exploiting peculiar properties of a given dynamical system. For instance, below we suggest an arguably simple choice of heuristics for the Reeds-Shepp vehicle.

Example 4 (Euclidean heuristics for the Reeds-Shepp vehicle)**.** *Given two configurations $p = (p_x, p_y, p_\theta), q = (q_x, q_y, q_\theta) \in SE(2)$, then:*

$$\begin{aligned} H_l^{\text{Eucl}}(p, q) &= \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}, \\ H_u^{\text{Eucl}}(p, q) &= H_l^{\text{Eucl}}(p, q) + \pi \end{aligned} \tag{4}$$

are respectively lower and upper bounds for the Reeds-Shepp distance between p and q. The lower bound can be easily explained, since a straight line connecting p, q is not dynamically feasible under general positions of p, q. The upper bound instead derives from the fact that a Reeds-Shepp curve never entails arcs of circle whose absolute-value sum exceeds an angle of π.

Clearly, not all heuristics are equal, and the speedup attained through pruning depends on the specific choice of heuristics. This shall effectively shift one's focus on finding "good heuristics". In the next section, we show that – for nonholonomic systems – on can use the mathematical tools reviewed in Section II-A.3 to produce heuristics with optimal pruning effectiveness at all the cardinalities of a k-d tree, as opposed to generic heuristics like the one introduced in Example 4. The optimality of such heuristic derives from the fact that, as the distance tends to zero, the estimates produced induce a metric equivalent to the geodesic distance for the sub-Riemannian geometry.

A. Ball-inclusion vs. bounds.

In this section, we clarify the relationship between the sub-Riemannian boxes and lower/upper bounds to the sub-Riemannian metric. Intuitively, we use the following fairly general argument: if the balls of a certain metric d' contain those of another metric d with the same radius, then d' is a lower bound to d. Simple examples of this fact are shown in Figure 3, while in Lemma 1 below, we formalize the argument in relation to sub-Riemannian geometries, using the results of the Ball-Box Theorem.

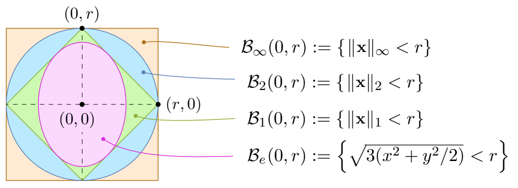

$(0, r)$
$(r, 0)$
$(0, 0)$
$\mathcal{B}_\infty(0, r) := \{\|\mathbf{x}\|_\infty < r\}$
$\mathcal{B}_2(0, r) := \{\|\mathbf{x}\|_2 < r\}$
$\mathcal{B}_1(0, r) := \{\|\mathbf{x}\|_1 < r\}$
$\mathcal{B}_e(0, r) := \left\{\sqrt{3(x^2 + y^2/2)} < r\right\}$

Fig. 3. Visualizing the relationship between metric inequalities and ball-inclusions. Since $\mathcal{B}_e(0, r) \subseteq \mathcal{B}_1(0, r) \subseteq \mathcal{B}_2(0, r) \subseteq \mathcal{B}_\infty(0, r)$ for all $r \in \mathbb{R}_{>0}$, then $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \leq \sqrt{3(x^2 + y^2/2)}$ for all $\mathbf{x} = (x, y)^T \in \mathbb{R}^2$.

By Definition 2, sub-Riemannian balls centered in p are essentially *sublevel sets* of the sub-Riemannian distance from p. Similarly, we can regard outer boxes as the sublevel sets of a suitable function, $\tau_o(p, x) := \max_{i=[1,k]} |(Z_i^p(x)/C_i)^{-w_i}|$, such that, by Equation (1)

$$\text{Box}_o(p, r) \equiv \{p' \in \mathcal{M} : \tau_o(p, p') < r\}. \tag{5}$$

Lemma 1. *There exists a constant $\varepsilon_0 > 0$ such that, for all $\varepsilon < \varepsilon_0$ and for all $p, q \in \mathcal{M}$, $\tau_o(p, q)$ is a lower bound to the sub-Riemannian distance $d(p, q)$.*

Proof. Consider two points $p, q \in \mathcal{M}$ and call $r_l = \tau_o(p, q)$ s.t. $r_l \leq \varepsilon_0$, then clearly $q \notin \text{Box}_o(p, r_l)$ from Equation (5). Since $\mathcal{B}(p, r_l) \subset \text{Box}_o(p, r_l)$ by Equation (2), then also $q \notin \mathcal{B}(p, r_l)$. In turn, this implies that $d(p, q) \geq r_l = \tau_o(p, q)$

by Definition 2, i.e., the outer box is in fact associated to a lower bound of the sub-Riemannian distance. □

With an analogous argument, one can show the correspondence of inner boxes with an upper bound τ_i, such that $\tau_i(p,q) \geq d(p,q)$ for all $p,q \in \mathcal{M}$.

Although τ_0 and τ_i are acceptable lower and an upper bounds for $d(p,q)$, they only represent *quasi-metrics*, i.e., they lack symmetry, due to the local validity of the privileged coordinates around a given point. To overcome this, we define:

$$\begin{aligned} H_l^{\mathrm{Box}}(p,q) &:= \max[\tau_o(p,q), \tau_o(q,p)] \\ H_u^{\mathrm{Box}}(p,q) &:= \min[\tau_i(p,q), \tau_i(q,p)]. \end{aligned} \tag{6}$$

Remark 2. *As opposed to the euclidean heuristics from Equation (4), the box-heuristics defined above are known to estimate the sub-Riemannian metric up to a constant factor, for arbitrarily small distances. We refer to this property as* asymptotic equivalence.

Based on these results, we will present two variations of the k-d tree algorithm with a reduced number of distance evaluations: one using *Lower-Bound Pruning* (LBP) and a second one using both lower and upper bounds, which we refer to as *Interval Pruning* (IP).

B. Lower-bound pruning (LBP).

In this variation of the algorithm, the lower-bound distance to the query point q can be exploited as a quick way of discarding a newly visited data point x_i. In fact, if $H_l(q,x_i)$ exceeds the highest distance in the queue, the evaluation of $d(q,x_i)$ is unnecessary. This behavior is achieved with a straightforward modification of Algorithm 1, reported below in Algorithm 2.

Algorithm 2: k-d tree query with Lower-Bound Pruning (LBP)

```
1 define query_lbp ($q \in \mathcal{M}$, $\mathcal{T} = (\mathcal{V}, \mathcal{E}^-, \mathcal{E}^+)$):
      [...]    // Lines 2-6 from Algorithm 1
7     if $d_l(q,x_i) < d_m$ then Q $\longleftarrow Q \cup \{(\mathrm{dist}(q,\ x_i), v_i)\}$ ;
      [...]    // Lines 8-11 from Algorithm 1
```

Despite very significant performance improvements – which will be confirmed with experimental results in Section IV – Algorithm 2 suffers from a worst-case linear query complexity. Per se, this is a known fact about k-d trees in general, which also affects the unpruned query technique (Algorithm 1). However, in both algorithms, the worst case also entails a *linear number of distance evaluations*, which is undesireable when computing distances is an especially demanding task.

C. Interval pruning (IP).

The worst-case linear complexity of Algorithms 1 and 2 is caused by the fact that binary decisions are taken – at each visited node – solely on the information available from previously explored branches of the k-d tree, thus making the algorithm sensitive to adversarial traversal orders.

While an overall linear complexity of k-d tree queries is generally unavoidable in the worst case, Interval Pruning prevents that a linear number of distance evaluations occur.

Loosely speaking, the technique is designed to postpone all the distance evaluations (*evaluation phase*), so that one can first collect and exploit as much information as possible from the heuristics (*traversal phase*). In doing so, we are *trading a reduction in distance evaluations with an increased amount of visited vertices*.

To take full advantage of both the lower and the upper bounds, we design a specialized data structure, which we refer to as a *Bounded Interval Priority Queue*, Q_I. In principle, this data structure behaves similarly to the bounded priority queue described in Section II-C.

However, instead of known quantities that can be directly compared, Q_I accepts *intervals*, i.e., compact subsets of $\mathbb{R}$, for which only a partial order is defined. Internally, Q_I maintains only the intervals that might contain the m lowest values and updates its *supremum* $\sup(Q_I)$ i.e., the worst-case higher value of the m-th biggest element.

Below, we detail an efficient implementation of the Q_I data structure, while the pseudocode of the Interval Pruning query algorithm is reported in Algorithm 3.

The Q_I data structure can be described as a pair (Q_s, Q_c), with the following components:

a) Sup-Queue Q_s: A bounded priority queue of size m, with elements $Q_s = [q_1, q_2, ..., q_m]$, where each q_i is a real-vertex pair $q_i = (u_i \in \mathbb{R}_{\geq 0}, v_i \in \mathcal{V})$. The property $u_1 \leq u_2 \leq ... \leq u_m$ is an invariant of this data structure, that is, when a new element (u_{new}, v_{new}) s.t. $u_{new} < u_m$ is inserted, q_m is discarded and the indices of the remaining elements are updated accordingly. Since the queue is kept ordered, these updates are implementable with $O(\log|Q_s|)$ operations. The number u_m is the worst-case distance of the m-th nearest neighbor from q, which we refer to as the supremum of the interval queue, $\sup(Q_I) := u_m$. Note that the quantity $\sup(Q_I)$ is monotonically non-increasing during a query.

b) Candidate Queue Q_c: This collects the candidate nearest neighbors ordered by incresing lower bounds. Specifically, $Q_c = \{c_i\}_{i=1}^r$, where each element is a real-vertex pair $c_i = (l_i \in \mathbb{R}_{\geq 0}, v_i \in \mathcal{V})$ such that $l_1 \leq l_2 \leq ... \leq l_r \leq \sup(Q_I)$. When $\sup(Q_I)$ is decreased, Q_c is minimally truncated to maintain the invariant, i.e., the least number of tail elements are discarded. Similarly, these updates are implementable with $O(\log|Q_c|)$ operations. Note that all the vertices in Q_s also appear in Q_c, hence $|Q_c| \geq |Q_s|$ and the number of candidates shall not be limited to a maximum cardinality.

c) Interval insertion: Let $\texttt{insert}(Q_I, [L,U], v)$ denote the insertion into Q_I of the interval $[L,U]$ associated to vertex v. This operation can be implemented as: $Q_s \leftarrow Q_s \cup \{(U,v)\}$, $\quad Q_c \leftarrow Q_c \cup \{(L,v)\}$, where $\cup$ expresses the addition of new elements into the queues with implicit internal updates to preserve their invariants.

d) Interval retrieval: Let $\texttt{popFront}(Q_I)$ be a primitive that removes from Q_I and returns the first vertex in Q_c associated to a non-zero measure interval. When no such element exists, the procedure returns $\emptyset$.

The IP k-d tree query procedure is then split into two phases:

- in the *traversal* phase, the k-d tree is explored without distance evaluations. At each visited data point x_i, we insert into Q_I the lower and upper bounds to its distance from q, while $\sup(Q_I)$ is used to discard halfspaces. In Figure 4 we show the behavior of Q_I during this phase.

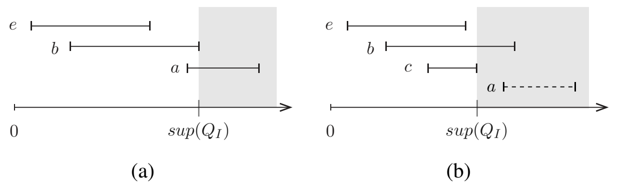


Fig. 4. Behavior of the Interval Prority Queue during the traversal phase of a query for $m = 2$ neighbors. (a) At an intermediate stage, Q_I holds 3 candidate intervals, referring to the nodes $\{a, b, e\}$ of the k-d tree. (b) After visiting node c, the supremum is lowered to $H_u(q, c)$ and a is discarded. Note that the candidates are kept in ascending ordered of lower bounds.

- in the *evaluation* phase, we walk through the ordered candidates left in Q_I after the traversal and we evaluate their exact distances from the query point q. At each evaluation, Q_I updates its supremum, thus dynamically discarding additional candidates. In Figure 5, we show the behavior of the interval priority queue during this phase.

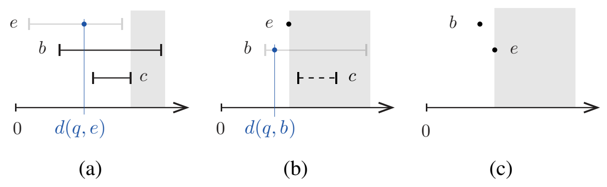


Fig. 5. Behavior of the Interval Prority Queue during the evaluation phase. (a) Evaluation of the distance from e to the query point q. This leaves the supremum unchanged. (b) When evaluating candidate b, the supremum drops to $d(q, e)$, thus causing c to be discarded, (c) Final result with $\{b, e\}$ left in the queue, as the 2 nearest neighbors to q.

IV. Experimental results

In this section, we test the performance of the proposed k-d tree query algorithms, with the metric generated by the geodesics of a Reeds-Shepp vehicle. Specifically, we test the Lower-Bound Pruning (LBP) and Interval Pruning (IP), coupled with both the Euclidean and Box Heuristics introduced in Equations (4) and (6), respectively. For comparison, we also include in our tests the unpruned k-d tree query algorithm and a Geometric Near-Neighbor Access Tree (GNAT) [17]. To date, the latter is designated in

Algorithm 3: k-d tree query with Interval Pruning (IP)

```
1  define query_ip ($q \in \mathcal{M}$, $\mathcal{T} = (\mathcal{V}, \mathcal{E}^-, \mathcal{E}^+)$):
2      $Q_s, Q_c \longleftarrow \{q_{1:m} = (\emptyset, \infty)\}$, $Q_I \longleftarrow (Q_s, Q_c)$;
3      define queryNode ($q \in \mathcal{M}$, $v_i = (x_i, n_i) \in \mathcal{V}$):
4          if $v_i = \emptyset$ then return;
5          $s \longleftarrow$ sideOf ($q$, $x_i$, $n_i$);
6          queryNode ($q$, child($v_i, s$))
7          insert ($Q_I$, $[d_l(x_i, q), d_u(x_i, q)]$, $v_i$)
8          if ballHyperplane ($q$, $\sup(Q_I)$, $x_i$, $n_i$) then
9              queryNode ($q$, child($v_i$, opposite($s$)))
       // Traversal phase
10     queryNode ($q$, root ($\mathcal{T}$)) ;
       // Evaluation phase
11     while $v = (x, n) \longleftarrow$ popFront($Q_I$) and $v \neq \emptyset$ do
12         insert ($Q_I$, $[d(q, x), d(q, x)]$);
13     return $Q_I$;
```

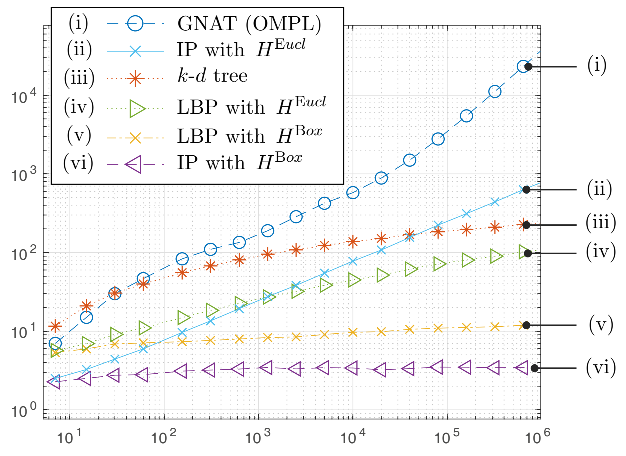
OMPL [18] as the default algorithm of choice to address Nearest-Neighbor Search within sampling-based planners.

To produce the performance plots of Figure 6, we build a k-d tree up to certain test cardinalities. Once a test cardinality is reached, for each algorithm we perform 1000 queries, in the search for $m = 2$ nearest neighbors and evaluate the average performances. Specifically, we monitor the number of distance evaluations and the number of visited nodes during queries. We then repeat until a maximum test cardinality of $N_{\max} = 10^6$ is reached. For a fair comparison, the query points are drawn in advance at random from a uniform distribution in the region $[-10, 10]^2 \times (-\pi, \pi]$ and the same query sequence is used across all the analyzed algorithms. Up to cardinality $N = 10^4$, the correctness of the algorithm is verified by comparison with the results of linear search. At higher cardinalities, results are cross-verified for practical reasons.

As compared to the unpruned k-d tree (curve (iii), Algorithm 1), in Figure 6(a) we observe that both the LBP and the IP with the Box heuristics (v)-(vi) cause a significant reduction of distance evaluations, respectively by observed factors of ~20 and ~65 at N_{max}. Noticeably, the IP variant computes a near-minimal[1] number of distances (~3.44 at $N_{\max}$), which also appears asymptotically constant, for practical purposes. As expected, it also visits more nodes than its LBP counterpart, due to delaying the distance computations.

Additionally – as anticipated in Section III-C – we observe significantly worse results when using the Euclidean heuristics. Specifically, the LBP algorithm with H^{Eucl}(curve (iv)) fails to maintain its pruning effectiveness at high cardinalities, while the corresponding IP variant (ii) exhibits a remarkably suboptimal behavior. The latter is precisely due to the fact that H^{Eucl} does not define an asymptotically-equivalent metric. As the cardinality of the k-d tree increases, the sample dispersion tends to zero, while the measure of the intervals inserted in Q_I stays constant, π. In turn, the ratio $\sup(Q_I)/d_m$ grows unbounded and causes the algorithm to visit an asymptotically unbounded number of unnecessary

[1]the least no. of evaluations is the no. of neighbors seeked, $m = 2$.



(a) Average number of distance evaluations

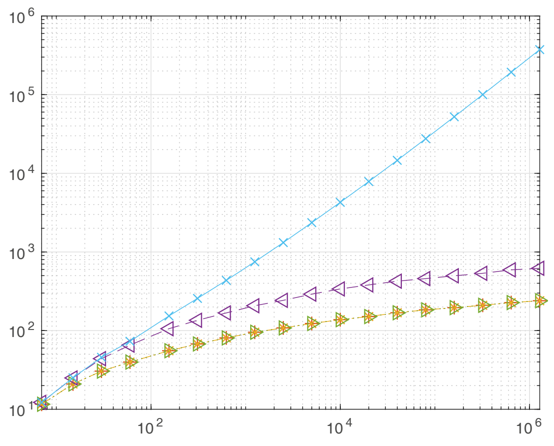

(b) Average number of visited nodes

Fig. 6. Experimental results comparing GNAT (i) and the unpruned k-d tree algorithm (iii) with different pruned query algorithms (ii, iv, v, vi). The latter are obtained from all the combinations of pruning (LBP / IP) and Heuristics (H^{Eucl} / H^{Box}) presented in Section III.

sibling branches. This undesired behavior is also confirmed by the significantly different asymptotic slopes of curves (ii) and (vi) in all the performance plots, corroborating the importance of using pruning heuristics *asymptotically equivalent* to the underlying metric.

V. Conclusion

Motivated by applications in sampling-based motion planning, in this work we have investigated techniques to reduce the runtime of k-d tree queries for dynamical systems subject to driftless nonholonomic differential constraints. The efficiency afforded by our techniques is based on *reducing the number of distance evaluations* necessary to discern the nearest neighbors, with the aid of computationally convenient heuristics that provide lower and upper bounds to the geodesic distance. We have introduced two pruned variations of the k-d tree query algorithm: the first one (LBP) based solely on the lower-bound heuristics and the second one (IP) based on both the lower and the upper bounds.

As to the heuristics, we have identified *asymptotic equivalence* to the true metric – i.e., the ability to estimate arbitrarily small distances up to a constant factor – as a crucial property for the introduced pruning algorithms to offer optimal behavior. Additionally, by means of tools from sub-Riemannian geometry, we have described a methodology to generate asymptotically equivalent heuristic in the case of driftless nonholonomic systems.

In experimental tests carried out with a Reeds-Shepp vehicle model, the proposed pruning techniques – coupled with asymptotic equivalent heuristics – have shown to reduce the number of distance evaluations to a near-minimal level.

References

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Citeseer, 1998.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[4] J. Bialkowski, S. Karaman, M. Otte, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 365–380.

[5] M. Kleinbort, O. Salzman, and D. Halperin, "Collision detection or nearest-neighbor search? on the computational bottleneck in sampling-based motion planning," *CoRR*, vol. abs/1607.04800, 2016. [Online]. Available: http://arxiv.org/abs/1607.04800

[6] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[7] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[8] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[9] V. Varricchio, B. Paden, D. S. Yershov, and E. Frazzoli, "Efficient nearest-neighbor search for dynamical systems with nonholonomic constraints," *CoRR*, vol. abs/1709.07610, 2017. [Online]. Available: http://arxiv.org/abs/1709.07610

[10] P. Gomoluch, D. Alrajeh, A. Russo, and A. Bucchiarone, "Towards learning domain-independent planning heuristics," *CoRR*, vol. abs/1707.06895, 2017. [Online]. Available: http://arxiv.org/abs/1707.06895

[11] B. Paden, V. Varricchio, and E. Frazzoli, "Verification and synthesis of admissible heuristics for kinodynamic motion planning," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 648–655, 2017.

[12] F. Jean, *Control of nonholonomic systems: from sub-Riemannian geometry to motion planning*. Springer, 2014.

[13] R. Montgomery, *A tour of subriemannian geometries, their geodesics and applications*. American Mathematical Soc., 2006, no. 91.

[14] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[15] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[16] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE, 2010, pp. 7681–7687.

[17] S. Brin, "Near neighbor search in large metric spaces," 1995.

[18] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, no. 4, pp. 72–82.