A Homotopy Method for Motion Planning.

Shenyu Liu and Mohamed Ali Belabbas**

Abstract

We propose a novel method for motion planning and illustrate its implementation on several canonical examples. The core novel idea underlying the method is to define a metric for which a path of minimal length is an admissible path, that is path that respects the various constraints imposed by the environment and the physics of the system on its dynamics. To be more precise, our method takes as input a control system with holonomic and non-holonomic constraints, an initial and final point in configuration space, a description of obstacles to avoid, and an initial trajectory for the system, called a sketch. This initial trajectory does not need to meet the constraints, except for the obstacle avoidance constraints. The constraints are then encoded in an inner product, which is used to deform (via a homotopy) the initial sketch into an admissible trajectory from which controls realizing the transfer can be obtained. We illustrate the method on various examples, including vehicle motion with obstacles and a two-link manipulator problem.

1 Introduction

A fundamental problem in robotic motion planning is to find a trajectory which meets the various constraints stemming from the system's dynamics, which can be of holonomic or non-holonomic type, and obstacle avoidance constraints, which include constraints on the magnitude of some of the variables describing the system (e.g., a maximal turning radius), or obstacles present in physical space. We propose here a new method to find a trajectory which takes into account all the above constraints–we call such a trajectory *admissible*–and illustrate its performance on several examples. The method is a homotopy method: given an initial state and a final desired state, $\mathbf{x}_i$ and $\mathbf{x}_f$ respectively, and an arbitrary curve joining $\mathbf{x}_i$ to $\mathbf{x}_f$ in state-space, the method *deforms* the curve into an admissible curve joining $\mathbf{x}_i$ to $\mathbf{x}_f$. We presented a preliminary version of this method, with only non-holonomic constraints, in [1]. In this paper, we restrict the presentation to systems *affine in the control*, and leave the general case to subsequent work. We also refer the readers to the website[1] for slides, sample Matlab code and examples showcasing the method.

The problem of motion planning in robotics and control is a canonical problem, and many methods have been proposed over the years. For this reason, we can only give here a very partial overview of the current state of the field, and emphasize that the method we propose is built on a rather different set of ideas. A large subset of the methods is focused on non-holonomic dynamics, since this problem is by itself difficult and with a long history [13, 12, 3, 14]. Many of the proposed methods are based on the use of sinusoidal driving signals; the basic relation underlying these methods is the system approximation

$$\dot{x} = \lim_{\omega \to \infty} \left(\sqrt{\omega} \sin(\omega t) f_1(x) + \sqrt{\omega} \cos(\omega t) f_2(x) \right) \Leftrightarrow \dot{x} = [f_1, f_2](x),$$

**Shenyu Liu and Mohamed Ali Belabbas are with the department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana-Champaign. sliu113,belabbas@illinois.edu

[1] https://publish.illinois.edu/belabbas/motion-planning/

where $[f_1, f_2]$ is the Lie bracket [7] of the vector fields f_1, f_2. Indeed, this insight is at the basis of the work of Brockett [2], Murray et al. [16], Laferriere and Sussman [11]. Furthermore, interesting recent work shows that some special functions–which can be thought as generalizations of harmonic functions—play a distinguished role in solving under-actuated control problems [8].

For control and verification of hybrid systems in general, we refer to [19] and for a recent survey of motion planning for self-driving vehicles in urban environment, we refer to [18]. Other approaches of interest to obtain feasible trajectories for given problems and dynamics including random sampling-based [9] graph-based [10], and optimization-based approaches [6] and approaches based on solvers for nonlinear dynamics.

2 Background and problem set-up

We present some background and notation needed to explain the method. We refer to as vehicle/robot/plant whose motion we desire to plan as *the system*. The system is assumed to obey the controlled dynamics

$$\dot{\mathbf{x}} = \sum_{i=1}^{p} u_i f_i(\mathbf{x}), \tag{1}$$

where $\mathbf{x} \in M$ with M a (at least locally) differentiable manifold called the *configuration space*, $f_i(\mathbf{x})$ the *actuation vector fields* and $\mathbf{u} := (u_1, \ldots, u_p) \in \mathbb{R}^p$ the controls. We refer to as *workspace* the physical environment in which the system lives. We denote by $\text{span}_x\{g_i\}$ the (real) vector space spanned by the vectors $g_i(x)$.

We call a *curve* in configuration space a piecewise differentiable function $\mathbf{x}(t) : [0, T] \to M$, where $T > 0$, and refer to $\mathbf{x}(0)$ and $\mathbf{x}(T)$ as start-point and end-point, respectively, of $\mathbf{x}(t)$. We refer to them collectively as *end-points*. We call the *image* of a curve a *path*; a path is thus a geometric object (a collection of "contiguous states") and the times at which each point in a path is visited are not specified.

A *fixed end-points homotopy* between the two curves $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$ with the same end-points (i.e., $\mathbf{x}_1(0) = \mathbf{x}_2(0)$ and $\mathbf{x}_1(T) = \mathbf{x}_2(T)$) is a differentiable function $\mathbf{v}(s, t) : [0, \infty) \times [0, T] \to M$ with the properties:

$$\begin{aligned} \mathbf{v}(s, 0) &= \mathbf{x}_1(0) && \text{for all } s \geq 0 \\ \mathbf{v}(s, T) &= \mathbf{x}_1(T) && \text{for all } s \geq 0 \end{aligned}$$

The *length* of a curve $\mathbf{x}(t)$ is defined with respect to an norm on the tangent bundle TM of M. In the following, one can assume that $M = \mathbb{R}^n$ and the tangent space of M at $\mathbf{x} \in M$, denoted by $T_{\mathbf{x}}M$ is also $\mathbb{R}^n$. A *Riemannian inner product* on M is an given by piecewise differentiable symmetric positive definite bilinear form $G(\mathbf{x}) : T_{\mathbf{x}}M \times T_{\mathbf{x}}M \to \mathbb{R}$.. With a slight abuse of notation, we also denote by $G(\mathbf{x})$ its matrix representation in coordinates. Hence, we can think of $G(\mathbf{x})$ as an $\mathbf{x}$-dependent positive definite symmetric matrix.

The *length* of a curve $p(t)$ is then given by

$$L(\mathbf{x}) := \int_0^T \sqrt{\dot{\mathbf{x}}^\top(t) G(\mathbf{x}(t)) \dot{\mathbf{x}}(t)} dt. \tag{2}$$

Finally, we introduce the Christoffels' symbols associated to $G(\mathbf{x})$. To this end, denote by g_{ij} the ijth entry of the matrix representation $G(\mathbf{x})$, and by g^{ij} the ijth entry of the matrix $G^{-1}(\mathbf{x})$. The Christoffel's symbols are

$$\Gamma^i_{jk}(\mathbf{x}) := \frac{1}{2} \sum_l g^{il} \left(\frac{\partial g_{lj}}{\partial x_k} + \frac{\partial g_{lk}}{\partial x_j} - \frac{\partial g_{jk}}{\partial x_l} \right) \tag{3}$$

Problem definition. The problem that the method MotionSketch solves is the following: given a configuration space M, a set of holonomic, non-holonomic and obstacle avoidance constraints, an initial state $\mathbf{x}_i$ and a desired final state $\mathbf{x}_f$, provide a curve $\mathbf{x}(t) : [0, T] \to M$ which respects these constraints and so that $\mathbf{x}(0) = \mathbf{x}_i$, and $\mathbf{x}(T) = \mathbf{x}_f$, and provide the control $\mathbf{u}$ that drive a control system from $\mathbf{x}_i$ to $\mathbf{x}_f$. From now on, we normalize the time T to be equal to one; this is done for simplicity of exposition, and all the results below are easily extended to the case of arbitrary T. We recall that a curve that meets the constraints is an **admissible curve**.

Length of a curve. In order to provide an intuitive justification of the method, we first revisit the definition of the generalized length of a curve given a Riemannian metric in 2. See also Fig. 1. Since $G(\mathbf{x})$ is positive definite for all $\mathbf{x} \in M$, we can factor it as $G(\mathbf{x}) = F(\mathbf{x})D(\mathbf{x})F^\top(\mathbf{x})$, where $D(\mathbf{x})$ is a positive definite diagonal matrix, and $F(\mathbf{x})^\top F(\mathbf{x}) = I$ (i.e., $F(\mathbf{x})$ is an orthogonal matrix.) Let $\mathbf{x}(t) : [0, 1] \to M$ be a differentiable curve and let $0 = t_0 < t_1 < \ldots < t_{l+1} = 1$ provide subdivisions of the unit interval. We can then approximate

$$\dot{\mathbf{x}}(t_i) \simeq \frac{1}{\Delta t_i}(\mathbf{x}(t_{i+1}) - \mathbf{x}(t_i)) = \frac{1}{\Delta t_i}(\Delta\mathbf{x}(t_i)),$$

where $\Delta t_i = t_{i+1} - t_i$, and the second equality defines $\Delta\mathbf{x}(t_i)$. Using these relations, we can approximate the length of $\mathbf{x}(t)$ as

$$\begin{aligned} L(\mathbf{x}) &\simeq \sum_{i=1}^{l} \sqrt{(\frac{\Delta\mathbf{x}(t_i)}{\Delta t_i})^\top F(t_i)D(t_i)F(t_i)\frac{\Delta\mathbf{x}(t_i)}{\Delta t_i}}\Delta t_i \\ &\simeq \sum_{i=1}^{l} \sqrt{(F(t_i)^\top\Delta\mathbf{x}(t_i))^\top D(t_i)(F(t_i)\Delta\mathbf{x}(t_i))}, \end{aligned}$$

where we set $D(t_i) := D(\mathbf{x}(t_i))$ and $F(t_i) := F(\mathbf{x}(t_i))$. Since F is an orthogonal matrix, we can think of $F^\top\Delta\mathbf{x}$ the vector of coordinates describing $\Delta\mathbf{x}$ in the basis spanned by the column vectors of F; more precisely, if we set f_k to be the kth column of F and set $\Delta\mathbf{x}_k(t_i) = f_k^\top\Delta\mathbf{x}(t_i)$, then we have $\Delta\mathbf{x}(t_i) = \sum_k f_k\Delta\mathbf{x}_k(t_i)$. Now denote by d_k^2 the kth diagonal entry of D (recall that D has positive diagonal entries). We obtain

$$L(\mathbf{x}) \simeq \sum_i \sum_{k=1}^{n} \Delta\mathbf{x}(t_i)_k d_k(t_i).$$

Hence, by adjusting the d_i and the f_k appropriately, we can *adjust which infinitesimal directions for a curve yield a larger length.* We show how this can be brought to bear on motion planning problems below.

3 The method MotionSketch

The method contains the three following steps:

1. Encode the constraints of the motion planning problem (obstacles, holonomic, nonholonomic and dynamical constraints) into a Riemannian inner product.

2. Provide a curve in configuration space between the initial and final desired states. This curve, which we call the *sketch*, does *not* need to meet the holonomic, non-holonomic and dynamical constraints, but is required to avoid obstacles. Numerically solve the geometric heat flow (GHF), defined below, equation with the sketch as initial condition.

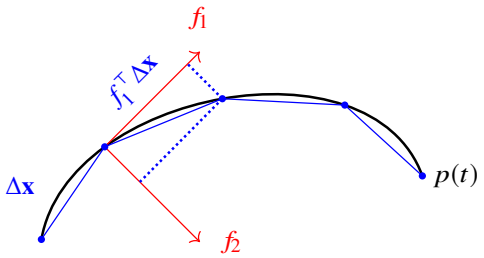


Figure 1: Length of a discretized curve.

3. Extract the controls from the solution of the GHF.

We now elaborate on the three items.

3.1 Step 1: Encoding the constraints in a Riemannian inner product

We start with holonomic/non-holonomic constraints.

3.1.1 Holonomic and non-holonomic constraints

Holonomic constraints can be formulated as a set of equations

$$q_i(\mathbf{x}) = 0, \quad i = 1, 2, \cdots, m_h$$

For each i and an infinitesimally small motion $\delta\mathbf{x}$, we have the approximation $q_i(\mathbf{x}_0 + \delta\mathbf{x}) \approx q_i(\mathbf{x}_0) + \frac{\partial q_i}{\partial \mathbf{x}}\delta\mathbf{x}$. In order to respect the constraint, $\delta\mathbf{x}$ needs to satisfy $q_i(\mathbf{x}_0 + \delta\mathbf{x}) = q_i(\mathbf{x}_0) = 0$, thus we have $\frac{\partial q_i}{\partial \mathbf{x}}\delta\mathbf{x} = 0$. This means that for $\mathbf{x}(t)$ to be an admissible curve, the direction of motion $\delta\mathbf{x}$ needs to be orthogonal to the vectors $\frac{\partial q_i}{\partial \mathbf{x}}$ for all i; in other words, it means the *undesirable* directions of motion are span $\left\{\frac{\partial q_i}{\partial \mathbf{x}}\right\}$.

We now turn our attention to non-holonomic constraints, which we assume are formulated as a set of constraints on the allowed velocities $\dot{\mathbf{x}}$ when at state x as follows:

$$\dot{\mathbf{x}}^\top f_{c,j}(\mathbf{x}) = 0, \quad j = 1, 2, \cdots, m_n.$$

The non-holonomic character of the constraints, which is reflected in the fact that they cannot be expressed as $\frac{d}{dt}q_n(\mathbf{x}) = 0$ for some function $q_n(\mathbf{x})$, does *not* play any particular function insofar our local encoding of the constraints is concerned; in fact, the undesirable directions of motion are easily seen to be in this case span $\{f_{c,j}(\mathbf{x})\}$.

Non-holonomic constraints can be presented as above, e.g. as non-slippage constraints, but they can also be encoded in the dynamics of the system, which is then called non-holononic. For this latter case, consider given the system of Eq. (1). We set $f_{f,i} = f_i$ and $f_{c,j}$ to be the m_n vectors orthogonal (for the Euclidean inner product) to $f_{f,i}$ for all $i = 1, \cdots, p$.

Encoding the constraints We set $\bar{p} := n - m_n - m_h$. We define the $n \times (n - \bar{p})$ matrix $\bar{F}_c$ as the matrix with first m_h columns given by $\frac{\partial q_i}{\partial \mathbf{x}}$ and the next m_n columns given by the $f_{c,j}$. We assume that $\bar{F}_c(\mathbf{x})$ is of constant rank almost everywhere in M, and we denote this rank by l, and set $p := n - l$. If $m_h + m_n = l$, it is of *full column rank*, and we set $F_c(\mathbf{x}) := \bar{F}_c(\mathbf{x})$. Otherwise $m_h + m_n > l$ and the constraints are not independent, in

the sense that satisfying a *subset* of the constraints insures that *all* constraints are met. We set $F_c(\mathbf{x})$ to be a $n \times l$ matrix whose column span equals the column span of $\bar{F}_c(\mathbf{x})$. Such matrix can be obtained, e.g., via the Gram-Schmidt process. Notice that F_c is of full column rank $l = n - p$ and the *column space of* F_c *contains all the undesirable directions of motion*.

Next, find a rank p matrix $F_f(\mathbf{x}) \in \mathbb{R}^{n \times p}$ such that

$$F_f(\mathbf{x})^\top F_c(\mathbf{x}) = 0,$$

which again can be found using the Gram-Schmidt process. The column space of $F_f(\mathbf{x})$ contains all the directions in which the system can move when at state $\mathbf{x}$. Note that in the absence of holonomic constraints, we can start with defining F_f with columns f_i as in Eq. (1) and choose F_c the satisfy the above relation. Set

$$F(\mathbf{x}) = \begin{pmatrix} | & | \\ F_c(\mathbf{x}) & F_f(\mathbf{x}) \\ | & | \end{pmatrix} \tag{4}$$

Then $F(\mathbf{x}) \in \mathbb{R}^{n \times n}$ and we define

$$H(\mathbf{x}) = F(\mathbf{x}) D F^\top(\mathbf{x}) \tag{5}$$

where $D = \text{diag}([\underbrace{k \cdots k}_{n-p} \underbrace{1 \cdots 1}_{p}])$ is a constant matrix. Note that this k is exactly the d^2 discussed in the Section II.b. In practice, we take k to be of the order of $10 \sim 1000$.

Using the interpretation of the length functional given in the previous section, it is easy to see that if $\dot{\mathbf{x}}$ is a direction that respects the constraints, it is not multiplied by k in the inner product $\dot{\mathbf{x}}^\top H(\mathbf{x})\dot{\mathbf{x}}$ with H defined via (5), so $\dot{\mathbf{x}}^\top H(\mathbf{x})\dot{\mathbf{x}}$ will not be scaled by k. On the other hand, if $\dot{\mathbf{x}}$ is a direction that violates a constraint, it has some components lying in span $F_c(\mathbf{x})$, and consequently $\dot{\mathbf{x}}^\top H(\mathbf{x})\dot{x}$ is large.

Finally, we record here that the partial derivative of H is given by

$$\frac{\partial H}{\partial x_i}(\mathbf{x}) = 2FD\frac{\partial F^\top}{\partial x_i}(\mathbf{x}),$$

which is needed for the computation of the Christoffels symbols.

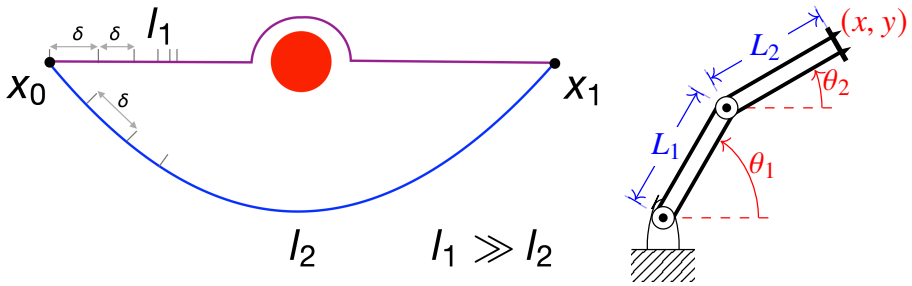
3.1.2 Obstacle constraints

We described obstacles $\Omega_i \subset \mathbb{R}^n$ in configuration space via functions $r_i : M \to \mathbb{R}$ according to

$$\Omega_i := \{\mathbf{x} \in \mathbb{R}^n : r_i(\mathbf{x}) \leq 0\}$$

The boundary of an obstacle is thus $\partial\Omega_i = \{\mathbf{x} \in \mathbb{R}^n : r_i(\mathbf{x}) = 0\}$. We incorporate obstacles in the Riemannian inner product via a barrier function $b(\mathbf{x}) = \sum_i b_i(\mathbf{x})$ with the following properties:

1. Each $b_i(\mathbf{x})$ is positive and differentiable for all $\mathbf{x} \in \mathbb{R}^n \backslash \Omega_i$
2. $b_i(\mathbf{x}) \to \infty$ as $\mathbf{x} \to \partial\Omega_i$,
3. $b(\mathbf{x}) = 1$ when $\mathbf{x}$ is far away from all Ω_i.



(a) (b)

Figure 2: (a).The length l_1 of the path passing near the obstacle is much larger that the length l_2 of the path staying far from the obstacles when the metric is scaled with $b(\mathbf{x})$. (b) Two-links articulated arm can be described as a system with 4 degrees of freedom and 2 holonomic constraints relating the position (x, y) of the tip to the joint angles θ_1, θ_2.

The idea is that we would like $b_i(\mathbf{x})$ to be large when $\mathbf{x}$ is in the vicinity of Ω_i, and becomes infinite if $\mathbf{x} \in \partial\Omega_i$. Thus if we multiply the metric tensor by $b(\mathbf{x})$, the length of a path that is in the vicinity of an obstacle is much larger than the length of a path that steer well-clear of the obstacle, where quantifying "well-clear" is of course dependent on the choice of $b_i(\mathbf{x})$ and how quickly it decays near the boundary of the obstacle. We illustrate this in Fig. 2a.

Such functions b_i are also known as *barrier functions* in the optimization literature [17]. In the case when obstacles are balls, that is, $\Omega = \cup_{i=1}^{l}\{\mathbf{x} \in \mathbb{R}^n : |\mathbf{x} - c_i| \le r_i\}$, one candidate of such $b(\mathbf{x})$ function will be a modification of penalty function from avoidance control [15]:

$$b(\mathbf{x}) = 1 + \sum_{i=1}^{l} \left(\min\left\{0, \frac{|\mathbf{x} - c_i|^2 - R_i^2}{|\mathbf{x} - c_i|^2 - r_i^2}\right\}\right)^2 \tag{6}$$

where R_i is such that $r_i < R_i$ for all $i = 1, 2, \cdots, l$, and R_i can be thought of as a *radius of detection* of the obstacle, in the sense that outside this radius, the obstacle does not affect the metric. Notice that $b(x)$ defined in (6) satisfies the 3 properties mentioned earlier. The derivative of b is also not hard to compute. Note that one can cover any obstacles with balls and use the above barrier function as a default approach.

3.1.3 simultaneous multi-vehicle path planning

Suppose there are l vehicles and each of them has its own state $\mathbf{x}_j = \left(x_{1j}, x_{2j}, \cdots x_{nj}\right)^\top \in \mathbb{R}^n$ and the dynamics is $\dot{\mathbf{x}}_j = F_j(\mathbf{x}_j)\mathbf{u}_j$. The j-th vehicle is supposed to drive from $\mathbf{x}_j(0) = \mathbf{a}_j$ to $\mathbf{x}_j(T) = \mathbf{b}_j$. Denote $\mathbf{x}^\top = \begin{pmatrix}\mathbf{x}_1^\top & \cdots \mathbf{x}_l^\top\end{pmatrix}$ and $\mathbf{u}^\top = \begin{pmatrix}\mathbf{u}_1^\top & \cdots \mathbf{u}_l^\top\end{pmatrix}$, then the system of multi-vehicle has total dimension of lm and initial and final states

$$\mathbf{x}_i = \begin{pmatrix}\mathbf{a}_1 \\ \vdots \\ \mathbf{a}_l\end{pmatrix}, \quad \mathbf{x}_f = \begin{pmatrix}\mathbf{b}_1 \\ \vdots \\ \mathbf{b}_l\end{pmatrix}.$$

and the overall dynamics is

$$\dot{\mathbf{x}} = \mathrm{diag}(F_1(\mathbf{x}_1), \cdots, F_l(\mathbf{x}_l))\mathbf{u} := F(\mathbf{x})\mathbf{u}. \tag{7}$$

While planning the path for all l vehicles, they are also supposed to avoid collision with each other. In case of planar vehicles where $(x_{1,j}, x_{2,j})$ represents the xy-coordinate of the j-th vehicle, collision between the j, k-th vehicles is avoided if

$$(x_{1j} - x_{1k})^2 + (x_{2j} - x_{2k})^2 \ge r_c^2, \tag{8}$$

where r_c is a safety radius guaranteeing collision-free between two vehicles. Thus the (6)-like barrier function induced from (8) is

$$b_c(\mathbf{x}) = \sum_{j \neq k} \left(\min \left\{ 0, \frac{(x_{1j} - x_{1k})^2 + (x_{2j} - x_{2k})^2 - R^2}{(x_{1j} - x_{1k})^2 + (x_{2j} - x_{2k})^2 - r_c^2} \right\} \right)^2$$

Thus, whenever two vehicles are too close ($(x_{1j} - x_{1k})^2 + (x_{2j} - x_{2k})^2 \leq R^2$), $b_c(\mathbf{x})$ becomes large and the metric at this state of vehicles is large. Notice that if we perform path planning for each individual vehicle first while treating the other vehicles as obstacles, the avoidance problem becomes dynamic in the sense that now the obstacles are moving with respect to time. Yet in our method avoidance of collision between vehicles and avoidance of static obstacles are processed in similar way and the result is promising as one can see later in our example.

In addition, Because $F(\mathbf{x})$ in (7) is block diagonal, H defined via (5) is also block diagonal and its j-th block only involves $\mathbf{x}_j$. As a result, inverse of H is in complexity of $O(lm^3)$ and computing $\frac{\partial H}{\partial x_i}$ for multi-vehicle has the same complexity as that for single vehicle. As a result, in each iteration of solving the numerical GHF equation, the complexity of computing all the Christoffel symbols is linear in l, the number of total vehicles.

3.1.4 The inner product with three type of constraints

We now formally define the inner product used in the method: given $H(\mathbf{x})$ as defined above from holonomic and non-holonomic constraints, and $b(\mathbf{x})$ a barrier function for the obstacles, we set

$$G(\mathbf{x}) := b(\mathbf{x})H(\mathbf{x})$$

With this construction, the partial derivatives of $G(\mathbf{x})$ can be computed using the chain rule: $\frac{\partial}{\partial x_i} G(\mathbf{x}) = \frac{\partial b}{\partial x_i}(\mathbf{x})H(\mathbf{x}) + b(\mathbf{x})\frac{\partial H}{\partial x_i}(\mathbf{x})$. Hence the Christoffel symbols in (3) can be computed solely based on the values H, $\frac{\partial H}{\partial x_i}$, b, $\frac{\partial b}{\partial x_i}$ at each state $\mathbf{x}$.

3.1.5 Examples

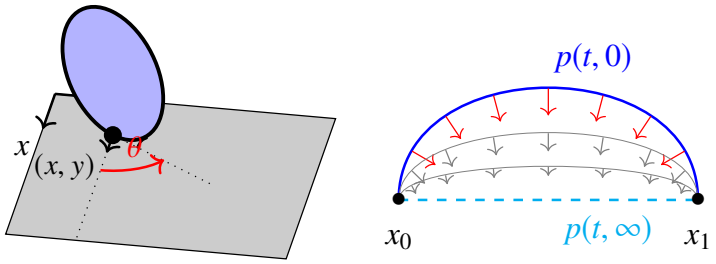
The two-links manipulator In this example we consider a two-links manipulator in the plane, see Fig. 2b. The working space, in terms of the position of the tool tip (x, y), is a subset of $\mathbb{R}^2$. The configuration space when the joint angles are also taken into account can be treated as a subset of $\mathbb{R}^4$. This system has 2 degrees of freedom and we can easily obtain the holonomic constraints:

$$\begin{cases} q_1(\mathbf{x}) = L_1 \cos(\theta_1) + L_2 \cos(\theta_2) - x = 0 \\ q_2(\mathbf{x}) = L_1 \sin(\theta_1) + L_2 \sin(\theta_2) - y = 0 \end{cases} \tag{9}$$

Taking differential of the two constraints, we find

$$\frac{\partial q_1}{\partial \mathbf{x}} = (-1,\ 0,\ -L_1 \sin\theta_1, L_2 \sin\theta_2)^\top, \frac{\partial q_2}{\partial \mathbf{x}} = (0,\ -1,\ L_1 \cos\theta_1,\ L_2 \cos\theta_2)^\top$$

Thus we set $F_c = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ \sin\theta_1 & -\cos\theta_1 \\ \sin\theta_2 & -\cos\theta_2 \end{pmatrix}$ and we find $F_f = \begin{pmatrix} -\sin\theta_1 & -\sin\theta_2 \\ \cos\theta_1 & \cos\theta_2 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$. We then set $F = (F_c \mid F_f)$.



(a) (b)

Figure 3: (a) A rolling coin or unicycle. is the side view. (b) In the mean-curvature flow, the curve $p(t, 0)$ is continuously deformed in the direction of its normal, depicted by the red arrows. The final curve is a straight line. In general, the final curve is a length minimizing curve. is the corresponding angles

We do not include obstacles and thus $b(\mathbf{x}) \equiv 1$ and

$$G = H = F\,\mathrm{diag}([k\ k\ 1\ 1])F^\top = \begin{pmatrix} \sin^2\theta_1+\sin^2\theta_2+k & -\frac{\sin 2\theta_1}{2}-\frac{\sin 2\theta_2}{2} & (k-1)\sin\theta_1 & (k-1)\sin\theta_2 \\ -\frac{\sin 2\theta_1}{2}-\frac{\sin 2\theta_2}{2} & \cos^2\theta_1+\cos^2\theta_2+k & -(k-1)\cos\theta_1 & -(k-1)\cos\theta_2 \\ (k-1)\sin\theta_1 & -(k-1)\cos\theta_1 & k+1 & k\cos(\theta_1-\theta_2) \\ (k-1)\sin\theta_2 & -\cos\theta_2\,k-1 & k\,\cos\theta_1-\theta_2 & k+1 \end{pmatrix}$$

The rolling coin or unicycle The kinematics of a unicycle can be modeled as

$$\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{pmatrix} = \begin{pmatrix}\cos\theta\\ \sin\theta\\ 0\end{pmatrix} u_1 + \begin{pmatrix}0\\ 0\\ 1\end{pmatrix} u_2 \tag{10}$$

where (x, y) is the position of the unicycle in the plane and θ is its orientation. Notice that there is only one non-holonomic constraints in this model and the constraint is the direction $\begin{pmatrix}-\sin\theta & \cos\theta & 0\end{pmatrix}^\top$ which prevents moving sideways and hence prevents slipping. Equivalently, because the model (10) is affine in control, the free directions F_f are simply the ones in (10). Hence

$$F(\mathbf{x}) = \begin{pmatrix}-\sin\theta & \cos\theta & 0\\ \cos\theta & \sin\theta & 0\\ 0 & 0 & 1\end{pmatrix},$$

from which we obtain

$$G(\mathbf{x}) = H(\mathbf{x}) = F\,\mathrm{diag}([k\ 1\ 1])F^\top = \begin{pmatrix}\cos^2\theta + k\sin^2\theta & (1-k)\cos\theta\sin\theta & 0\\ (1-k)\cos\theta\sin\theta & k\cos^2\theta+\sin^2\theta & 0\\ 0 & 0 & 1\end{pmatrix}.$$

3.2 Step 2: Initial sketch and solving the Geometric Heat Flow equation

Our method proceeds with solving the following GHF equation:

$$\frac{\partial}{\partial s}v_i(s,t) = \frac{\partial^2}{\partial t^2}v_i(s,t) + \sum_{j,k}\Gamma^i_{jk}\frac{\partial v_j}{\partial t}\frac{\partial v_k}{\partial t} \quad i = 1, 2, \ldots, n \tag{11}$$

where Γ^i_{jk} are the Christoffel symbols introduced in (3) for the inner product defined in the previous subsection. We impose the boundary conditions

$$v(s,0) = \mathbf{x}_i, v(s,1) = \mathbf{x}_f$$

and a user defined initial condition,

$$v(0,t) = \mathbf{x}(t)$$

in order to find the solution. The initial curve $\mathbf{x}(t)$ is an arbitrary curves satisfying the following 2 conditions:

1. It satisfies the boundary conditions: $\mathbf{x}(0) = \mathbf{x}_i$ and $\mathbf{x}(1) = \mathbf{x}_f$;
2. It does not pass though any obstacles: $r(\mathbf{x}(t)) > 0$ for all $t \in [0,1]$.

An important point here is that $\mathbf{x}(t)$ does not need to satisfy any holonomic or non-holonomic constraints; it can be simply a curve drawn from $\mathbf{x}_i$ to $\mathbf{x}_f$ without touching Ω.

Notice that for each $s \geq 0$ fixed, the solution $v(s,\cdot)$ represent a curve connecting $\mathbf{x}_i$ to $\mathbf{x}_f$. As we explain below, as s increases, $v(s,\cdot)$ is a curve that uses "less and less of the constrained directions", said precisely, $F_c^\top \frac{\partial}{\partial t} v(s,t)$ tends to zero. We set $s_{\max}$ to be the simulation time for the PDE (in our examples, between 1 and 20) and

$$\mathbf{x}_{sol}(\cdot) = v(s_{\max}, \cdot).$$

Mean-curvature flows We now elaborate on the origin of Eq. (11): it is a type of curve-shortening flow [4], called a *mean-curvature flow* for a 1-dimensional manifold (i.e. a curve) or *geometric heat flow*. For an introduction to mean-curvature flows in arbitrary dimensions, see [5]. For clarity of exposition, we present first the flow in two dimensional plane with the Euclidean inner product. We briefly mention steps that need to be taken for the general flow below.

Consider a curve $p(t) : [0,1] \to \mathbb{R}^2 = (p_1(t), p_2(t))$, as depicted in Fig. 3b. The scalar curvature [7] of p at $p(t)$ is defined as $\kappa(p(t)) = \|\ddot{p}\|$. Denote by $N_{p(t)}$ the unit normal vector pointing "inward". The curvature of p at $p(t)$ is then $\kappa(p(t))N(p(t))$.

The mean-curvature flow for this curve is defined as follows: consider a *family* of curves $p(t,s)$, $s \geq 0$, where for each s_0 fixed, $p(t,s_0) : [0,1] \to \mathbb{R}^2$ is a curve joining x_0 to x_1, and $p(t,0)$ is the original curve. Then the mean-curvature flow is the partial differential equation

$$\frac{\partial p}{\partial s} = \kappa(p(t,s))N(p(t,s)).$$

Note that it is in fact a system of two PDEs. Looking at Fig. 3b, it is easy to conclude intuitively that $\lim_{s\to\infty} p(t,s)$ converges to a straight line between x_0 and x_1. This is also the *shortest path* between x_0 and x_1 for the usual Euclidean metric. This is no accident, and we can show that in general the solution of this PDE converges to a curve of minimal length. For our purpose, we need to extend this idea in *two* directions: to (*i*) curves in higher dimensions and (*ii*) to a general Riemannian metric (or more precisely, inner product). One can show, after some extensive algebraic manipulations which we omit here, that the equivalent of the flow for a general curve in a Riemannian manifold is exactly the geometric heat flow presented in Eq. (11).

3.3 Step 3: Extracting the controls

The control can be directly computed:

$$\mathbf{u}(t) = F_f^\dagger(\mathbf{x}_{sol}(t))\dot{\mathbf{x}}_{sol}(t) \tag{12}$$

where $F_f^\dagger = (F_f^\top F_f)^{-1} F_f^\top$ is the pseudo-inverse of F_f. Notice that in the case $\mathbf{x}_{sol}$ is admissible, that is, if $\dot{\mathbf{x}}_{sol}(t) = F_f v(t)$ for some control $\mathbf{v}$,

$$\mathbf{u} = F_f^\dagger \dot{x}_{sol} = (F_f^\top F_f)^{-1} F_f^\top F_f \mathbf{v} = \mathbf{v}$$

Thus we have recovered the control and ideally the system should exactly follow the path $\mathbf{x}_{sol}$. Notice that $F_f F_F^\dagger$ is a minimal square error projection onto the column space of F_f, the control extracted from (12) will drive the system along a path that is close to $\mathbf{x}_{sol}$, even if $\dot{\mathbf{x}}_{sol}$ has small components in the constrained direction.

3.4 On the implementation

As mentioned earlier, the key of our method is to find an inner product matrix G and then solve the GHF equation (11). In our case, this is processed in MATLAB. To be explicit, once we have obtained F_c from the constraints, we implement them as symbolic vectors in MATLAB and thus find $F_f(\mathbf{x})$. Subsequently, both $G(\mathbf{x})$ and $\frac{\partial G}{\partial \mathbf{x}}$ can be derived symbolically and the symbolics are then replaced by state values and then stored in an $n \times n$ array `G` and an $n \times n \times n$ array `pG`, respectively. `pdepe` is then called with the boundary conditions and customized initial condition. In each iteration of solving the PDEs, the Christoffel symbols are computed from `G` and `pG` according to (3) and then stored in an $n \times n \times n$ array `Chris`. Notice that the `pdepe` solves PDEs of the general form

$$c(s,t,x,\frac{\partial x}{\partial t})\frac{\partial x}{\partial s} = x^{-m}\frac{\partial}{\partial t}\left(t^m f(s,t,x,\frac{\partial x}{\partial t})\right) + s(s,t,x,\frac{\partial x}{\partial t})$$

Compare it to (11) we see that in our case we need to set `c=ones(4,1);m=0,f=DxDt` and `s(i)=DxDt'*Chris(i,:,:)*DxDt`. Eventually the numerical solution of `pdepe` will be in the form of `sol(t,s,i)`,

3.5 Theoretical guarantee

Set $\Delta(x) = \text{span}\,\frac{\partial q_i}{\partial x} \cap \text{span}\, f_{c,j}$.

We call the constraints **satisfiable** if the distribution Δ satisfies the Lie algebraic rank condition (LARC). It is easy to see that it is a necessary condition for the existence of a trajectory joining arbitrary $\mathbf{x}_i$ and $\mathbf{x}_f$ while respecting the holonomic and non-holonomic constraints on the system. Under mild assumptions our method provides controls $\bar{\mathbf{u}}(t)$ so that the solution $\mathbf{x}^*(t)$ of $\dot{\mathbf{x}} = \sum_i \bar{\mathbf{u}}_i f_i$ by construction satisfies both the holonomic and non-holonomic constraints. In addition,

Theorem 3.1 *Suppose $F(\mathbf{x})$ defined in (4) is globally Lipschitz with constant L and $\|F_c(\mathbf{x})\| = 1$ for all $x \in \mathbb{R}^d$. Let $\bar{E}$ be the infimum of the energy functional*

$$E(\mathbf{u}) = \int_0^1 |\mathbf{u}(t)|^2 dt$$

over the space of controls that the corresponding state trajectory satisfies both the holonomic and non-holonomic constraints. For any arbitrary $k \in \mathbb{N}$, $s > 0$, define $\mathbf{x}$ to be the part $v(\cdot, s)$ of the solution of (11), $\mathbf{u}$ to be the control derived via (12) and $\tilde{\mathbf{x}}$ to be the solution of (1) generated by $\mathbf{u}$ from $\tilde{\mathbf{x}}(0) = \mathbf{x}_i$. Then for any $\epsilon > 0$, there exists $T = T(\epsilon, k)$ such that for all $s \geq T$,

1. *$E(\mathbf{u}) \leq \bar{E} + \epsilon$;*
2. *$|\tilde{\mathbf{x}}(t) - \mathbf{x}(t)| \leq \left(\sqrt{\frac{2t}{k}(\bar{E} + \epsilon)}\right) e^{L^2(\bar{E}+\epsilon)}$ for all $t \in [0,1]$. In particular, $|\tilde{\mathbf{x}}(1) - \mathbf{x}_f| \leq \left(\sqrt{\frac{2}{k}(\bar{E} + \epsilon)}\right) e^{L^2(\bar{E}+\epsilon)}$.*

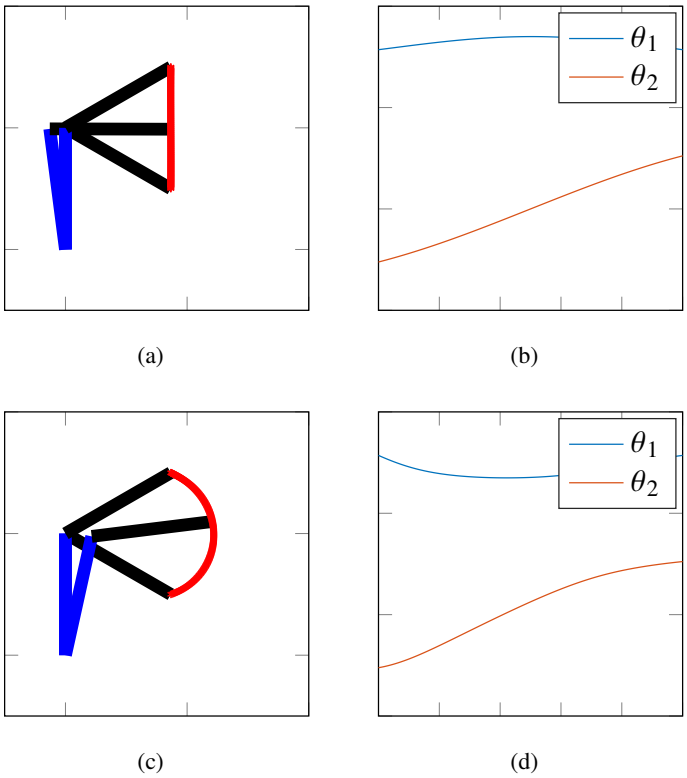


Figure 4: Vertical motion (a) and circular motion (c) of the two-links articulated arm. The links are in blue and black. The trajectory of the tip is marked in red. We draw the initial and final state and an intermediate state. The joint angles are given in (b) and (d) respectively.

4 Case study

Articulated arm We first study the 2R robot introduced earlier. Our goal is to plan the motion of the tip of the arm, from an initial state $\mathbf{x}(0) = \mathbf{x}_i = (\sqrt{2}/2, 1 - \sqrt{2}/2, \pi/2, -\pi/4)$, where we recall that the coordinates are $(x, y, \theta_1, \theta_2)$, to a final state $\mathbf{x}(1) = \mathbf{x}_f = (\sqrt{2}/2, 1 + \sqrt{2}/2, \pi/2, \pi/4)$. We furthermore require the motion to follow a *straight line* given by $x = constant$. The resulting motion planning problem thus contains, in addition to the two holonomic constraints relating the tip of the arm to the angles given in Eq. (9), the constraint $q_3(\mathbf{x}) = x - x_i = 0$ and the corresponding constrained direction is $\frac{\partial q_3}{\partial \mathbf{x}} = (1, 0, 0, 0)^\top$. Given these constraints, we implement the three steps of the method outlined above show the results in Fig. 4. We then replaced the constraint of vertical motion by asking that the tip follows an arc of a circle. The corresponding holonomic constraint is $q_4(\mathbf{x}) = (x - x_c)^2 + (y - y_c)^2 - r = 0$ for some constants x_c, y_c, r. The differential of this constraint is easily evaluated. We show in Fig. **??** the result obtained. Note that this illustrate the use of our method to solve *inverse kinematic problems* numerically.

Unicycle Consider the unicyle described above with coordinates (x, y, θ). We desire to transfer the unicycle from $(x(0), y(0), \theta(0)) = (-1, 0, 0)$ to $(x(1), y(1), \theta(1)) = (1, 0, 0)$ without slip (a non-holonomic constraint). In addition, there are two point obstacles located at $(-0.7, 0), (0.7, 0)$ which the unicycle should avoid in the xy-plane. Provided these constraints, we first build an inner product $G(x)$ as described earlier. We then provide an arbitrary curve connecting $(-1, 0, 0)$ and $(1, 0, 0)$ and avoiding the obstacles–we called this curve

the initial sketch. We opted simply for a sinusoidal curve in xy-plane and kept $\theta \equiv 0$, as shown in Fig. 5a. As observed in Fig. 5b, the unicycle certainly cannot follow this curve, as the motion direction is not aligned with the unicycle orientation or, in other words, the non-slip constraint is not met.

Recall that the solution of GHF equations (11) is a curve connecting the initial and final states when s fixed. In Figs. 5c to 5g, we show the gradual deformation of the curve in configuration space as s increases. In the final step $s = 4$, the curve becomes almost admissible and we see that the unicycle can basically follow such trajectory to reach its final state. It is worth noticing that because the obstacles are very close to the initial and final states, the unicycle has to move backward first in order to have more room to maneuver around said obstacles. Similarly, it overshoots the second obstacles before backing up and parking at its final destination.

Car We now illustrate our method for planning the motion of a car with position $(x, y) \in \mathbb{R}^2$, body orientation ϕ and wheel angle θ. A top view of car is illustrated in Fig. 6a and the equations of motion equation are:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix} = u_1 \begin{pmatrix} \cos\phi \\ \sin\phi \\ 0 \\ \frac{1}{d}\sin\theta \end{pmatrix} + u_2 \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \tag{13}$$

where u_1 is the throttle input, u_2 is the steering input and d is the distance between front wheels axis and rear wheels axis. We have studied this example in our paper [1], and we refer the reader to this paper for an explicit derivation of the corresponding $G(\mathbf{x})$.

Our first experiment is a 180° turn. Our initial sketch for this motion is illustrated in Fig. 6b. It is clear that, unless equipped with omniwheels or $d = 0$, the car cannot perform the motion illustrated. Interestingly, Motionsketch deforms this curve into the well-known 3-points turn path illustrated in Fig. 6c. This corresponds to the most efficient way of 180° turning of a car in practice, assuming there are no any other spatial obstacles.

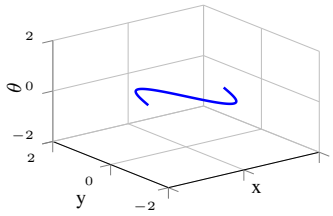
If in addition, we impose add parallel curbs, which are encoded in the barrier function $b(\mathbf{x})$ as described earlier, the constrained space the car can move in results in additional back-and-forth. The narrower the street, the more back-and-forth are needed. We provide additional examples in the webpage[2].

We conclude with the case of a car turning in a narrow street. The initial curve is simply an L-shaped curve in xy-plane with ϕ linear with respect to t and $\theta \equiv 0$, as illustrated in Fig. 7a. With the curbs modeled as obstacles, our method generates the relatively "optimal" path for this corner turn. Interestingly enough, the car is able to perform the turn in one shot if the street is relatively wide as shown in Fig. 7b, or may need extra maneuvering if the street is narrow, as shown in Fig. 7b. We emphasize that both simulation are performed with the same initial curve provided in Fig. 7a. The only difference is the street width. Whether one shot or two is automatically determined by our method without any further specification.

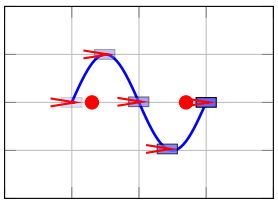
Finally, we note that in addition to the curb of the streets which are modeled as obstacles in the xy-plane, we also put limits on the steering angle θ as an obstacle for the θ variable.

Multi-vehicle path planning We show that multiple vehicles can be path planned simultaneously using our methods. In the first simulation two unicycles are initially at states $(0, 1, 0), (0, -1, 0)$; that is, parked at xy-coordinate $(0, 1), (0, -1)$ while both facing east. The task is to swap the position of the two unicycles. The initial sketch is a circle passing through the two unicycles – clearly these two paths are infeasible since the orientation vectors of the unicycles are not tangent to the paths. After running our algorithm, the two initial sketch of paths deform into the two V-shaped paths and now the two unicycles are able to perform the swap of

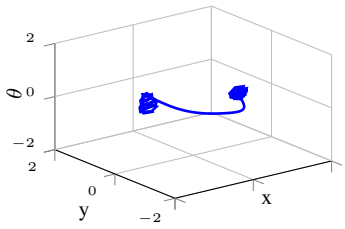
[2] https://publish.illinois.edu/belabbas/motion-planning/



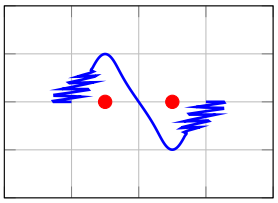
(a) Initial sketch in (x, y, θ)-space.



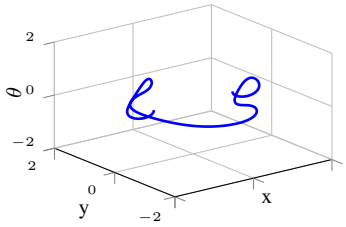
(b) Projection of the initial sketch in (x, y)- plane, with snapshots of the corresponding position of the unicycle. Note that $\theta = 0$ for each snapshot.



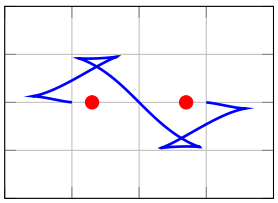
(c) Solution $\mathbf{x}(s, t)$ for $s = 0.0001$



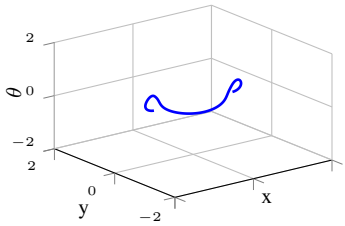
(d) Plane view at $s = 0.0001$



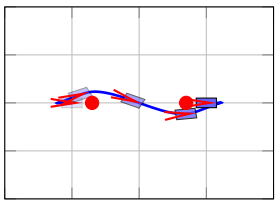
(e) Solution $\mathbf{x}(s, t)$ for $s = 0.01$.



(f) Solution $\mathbf{x}(s, t)$ for $s = 0.01$



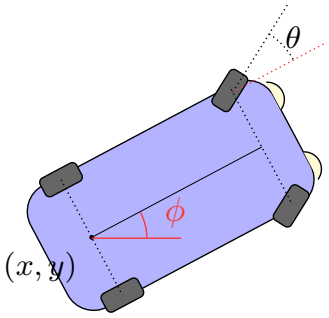
(g) 3D view of the final curve



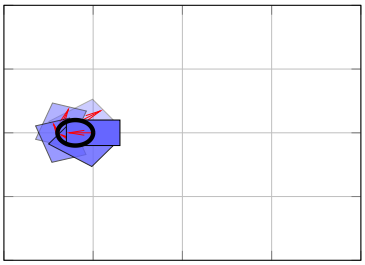
(h) Plan view of the final curve

Figure 5: Path planning for a unicycle avoiding two point obstacles. The red dots are the two obstacles, the blue curves are the solution of GHF equations at different s. In the plan views of initial curve and final curve, unicycle positions are marked along the curve, with its orientation labeled with red arrows

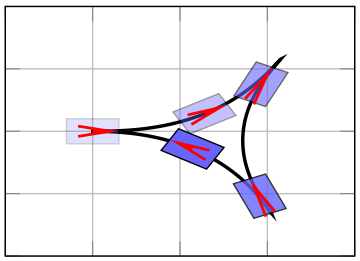
positions along such paths *while avoiding collisions*. While readers might think the previous example has no major difference compared with path planning for single vehicle and hence less challenging, the next example is more interesting and shows the power of our algorithm in multi-vehicle path planning. In this case one unicycle is supposed to move from $(-1, 0, \pi/2)$ to $(1, 0, \pi/2)$ while the other one is supposed to move from $(0, -1, 0)$ to $(0, 1, 0)$.



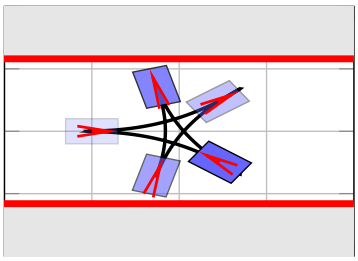
(a) Car modelled by Eq. (13). The red arrow is used to indicate the front of the car.



(b) Initial sketch. The car rotates 180 degrees with its center of mass following the black curve with slipping.

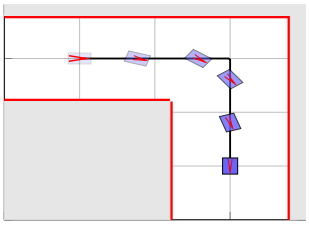


(c) 3 points turning when no spatial constraints

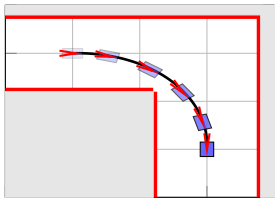


(d) 5 points turning between walls

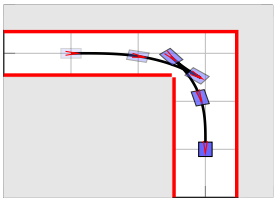
Figure 6: Car 180° turn experiment.



(a) Initial sketch. Note that the constraints are not met.



(b) Turn in a wide street corner



(c) Back-forth behavior at narrow street corner

Figure 7: Car street corner turn experiment

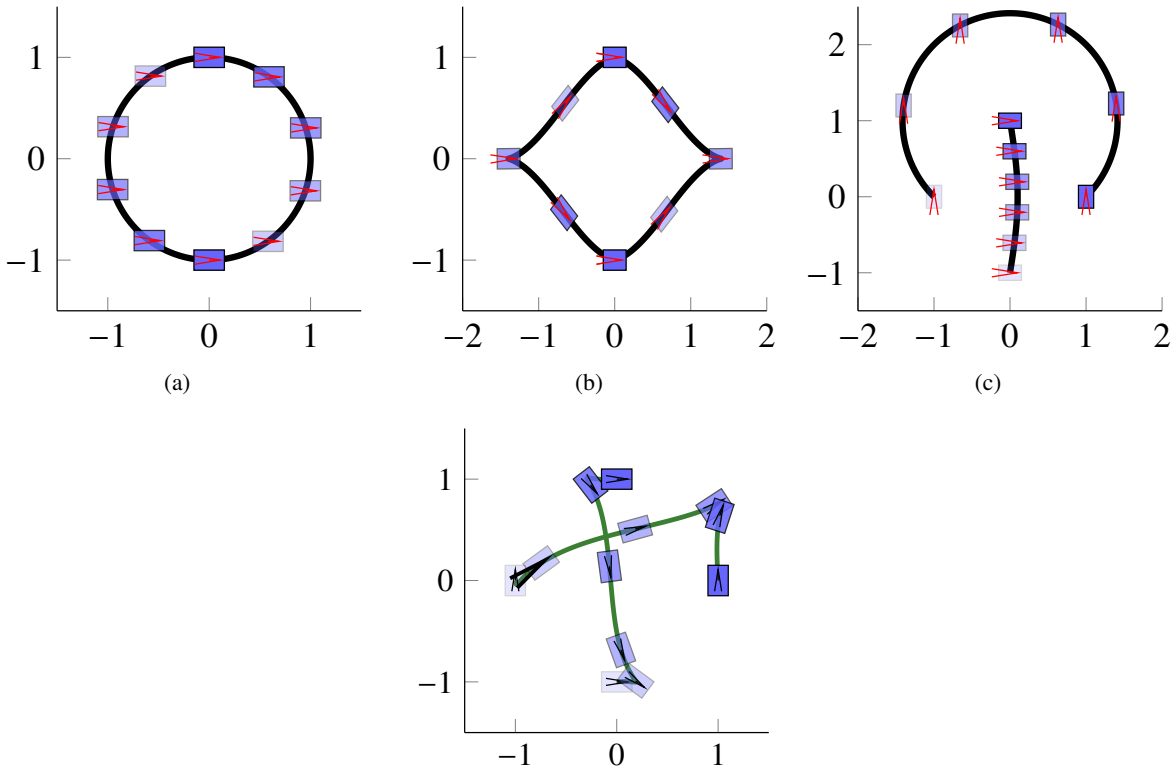


Figure 8: Multi-vehicles motion planning with collision avoidance.

5 Summary and discussion

We have provided in this paper a guide to the implementation of the method we termed MotionSketch for solving motion planning problems. We have illustrated the use of the method on examples with holonomic, non-holonomic and obstacle constraints, and have demonstrated that the method yields good practical results.

The salient points of the method were that it encodes all the constraints into a Riemannian inner product, and that it requires an initial sketch of the curve joining a desired final state to an initial state. This curve however does not need to meet the holonomic and non-holonomic constraints and is thus often easily obtained. In fact, if the space is convex, a straight line joining the two states most often meets the constraints.

Amongst the problems that are also readily solved using MotionSketch, but that we did not show here, we mention multi-vehicle motion planning with collision avoidance. For example, think of having to plan the trajectory of two non-holonomic cars with the constraints that they should avoid each other. This can be done using our method as follows: denote by $(x_i, y_i, \theta_i, \phi_i)$ the coordinates describing the state of car i, and by $G_i \in \mathbb{R}^{4\times4}$ the corresponding Riemannian inner products modeling the constraints for each car (e.g. max turning angle as am obstacle in θ, curbs, etc.). In order to model the two vehicles scenario, we first consider the cartesian product of the coordinates with metric $\bar{G} \in \mathbb{R}^{8\times8}$ a block diagonal matrix with blocks G_i. In order to avoid collisions between the cars, it suffices to place an obstacle around the "diagonal" subspace $x_1 = x_2$ and $y_1 = y_2$. As we have seen earlier, adding obstacles to a metric only requires multiplying by a barrier function, hence we can set $G(x) = b(x)\bar{G}(x)$. This procedure generalizes in a straightforward way to the case of more than two vehicles.

On the computational complexity of solving the GHF The numerically intensive part of the method lies in solving the geometric heat flow, which is a system of parabolic partial differential equations. We point out that solving such a PDE can be done rather efficiently, owing to the fact that the complexity scales polynomially with the dimension, and not exponentially, and the fact that there exists parallel algorithms to do so.

To elaborate on the first point, the main reason why the PDE we use scales well is that the *domain* of its solution has a *constant* dimension of two. For most PDEs encountered in engineering, such as the heat equation, or the Hamilton-Jacobi-Bellman equation, the dimension of the problem affects the dimension of the *domain* of the solution seeked, whereas is our case, it affects the dimension of the image of the solution. A linear increase in the dimension of the domain yields what is often referred to as the *curse of dimensionality*, as the number of interpolation points needed to represent a function on a domain of dimension n grows exponentially with n. Note however that the domain of our PDE is *always* two-dimensional, but the dimension of the image increases linearly, the number of interpolation points grows *linearly* with the dimension. Hence our PDE does not suffer from the curse of dimensionality and thus scales well to higher-dimensional problems. We refer to, e.g., [] for a more detailed discussion on the complexity of solving such PDEs. In practice, using MATLAB on a common laptop computer with non-optimized code (in particular, MATLAB does not solve such PDEs using multiple cores), the computation time was of the order of seconds to minutes, depending on the complexity of the problem. Per our discussion above, we believe however that there is ample room for improvement on this front.

References

[1] Mohamed-Ali Belabbas and Shenyu Liu. New method for motion planning for non-holonomic systems using partial differential equations. *2017 American Control Conference (ACC)*, pages 4189–4194, 2017.

[2] Roger W Brockett. On the rectification of vibratory motion. *Sensors and actuators*, 20(1-2):91–96, 1989.

[3] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. A Bradford book. Prentice Hall of India, 2005.

[4] Kai-Seng Chou and Xi-Ping Zhu. *The curve shortening problem*. CRC Press book, 2001.

[5] Tobias Colding, William Minicozzi, Erik Pedersen, et al. Mean curvature flow. *Bulletin of the American Mathematical Society*, 52(2):297–333, 2015.

[6] H. Dai, A. Valenzuela, and R. Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 295–302, Nov 2014.

[7] M.P. do Carmo. *Riemannian Geometry*. Mathematics (Boston, Mass.). Birkhäuser, 1992.

[8] Jean-Paul Gauthier and Matthias Kawskiz. Minimal complexity sinusoidal controls for path planning. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 3731–3736. IEEE, 2014.

[9] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[10] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Online footstep planning for humanoid robots. In *2003 IEEE International Conference on Robotics and Automation*, volume 1, pages 932–937 vol.1, Sept 2003.

[11] Gerardo Lafferriere and Hector J Sussmann. A differential geometric approach to motion planning. In *Nonholonomic motion planning*, pages 235–270. Springer, 1993.

[12] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[13] J.P. Laumond. *Robot motion planning and control*. Lecture notes in control and information sciences. Springer, 1998.

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

[15] G. Leitmann. Guaranteed avoidance strategies. *Journal of Optimization Theory and Applications*, 32(4):569–576, Dec 1980.

[16] Richard M Murray, Zexiang Li, S Shankar Sastry, and S Shankara Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[17] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*, chapter 19. New York : Springer, 1999.

[18] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[19] Claire J Tomlin, Ian Mitchell, Alexandre M Bayen, and Meeko Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.