Open access • Journal Article • DOI:10.1080/01691864.2016.1168317

A gradient-based path optimization method for motion planning — Source link

Mylène Campana, Florent Lamiraux, Jean-Paul Laumond

Institutions: University of Toulouse

Published on: 11 Apr 2016 - Advanced Robotics (Taylor & Francis)

Topics: Any-angle path planning, Motion planning, Path (graph theory) and Configuration space

A gradient-based path optimization method for motion planning

Mylène Campana, Florent Lamiraux, Jean-Paul Laumond

A gradient-based path optimization method for motion planning

Mylène Campana*, Florent Lamiraux and Jean-Paul Laumond
LAAS-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France

Abstract Most algorithms in probabilistic sampling-based path planning compute collision-free paths made of straight line segments lying in the configuration space. Due to the randomness of sampling, the paths make detours that need to be optimized. The contribution of this paper is to propose a basic gradient-based (GB) algorithm that transforms a polygonal collision-free path into a shorter one. While requiring only collision checking, and not any time-consuming obstacle distance computation nor geometry simplification, we constrain only part of the configuration variables that may cause a collision, and not entire configurations. Thus parasite motions that are not useful for the problem resolution are reduced without any assumption. Experimental results include navigation and manipulation tasks, e.g. a manipulator arm filling boxes and a PR2 robot working in a kitchen environment. Comparisons with a random shortcut optimizer and a partial shortcut have also been studied.

keywords: path optimization; motion planning; robotics

1 Introduction

Motion planning for systems in cluttered environments has been addressed for more than thirty years [1]. To explore the connected components of collision-free configuration spaces, pioneering contributions in the 90s introduced certain levels of random searches, i.e. random walks in [2], random sampling in [3] and [4]. Today most motion planners are inspired by these seminal approaches. The main issue using these techniques is that the computed path makes unnecessary detours and needs to be post-processed before being executed by a virtual or real robot. Alternative strategies exist however.

- Planning by path-optimization [5, 6] where obstacle avoidance is handled by constraints or cost using computation of the nearest obstacle distance. Most of these planners are using non-linear optimization [7] under constraints. Such planners provide close-to-optimality paths and have smaller time computation for easy problems, but they are mostly unable to solve narrow passage issues.
- Optimal random sampling [8] is also close to an optimal solution, but computation time is significantly higher than classical approaches.

Optimization is always regarding to one or several criteria. The most common in motion planning are the path length, which penalize detours, the obstacle clearance for safety and the execution time, which is influenced by the number of via points, introducing velocity discontinuities. One can also mention the minimization of the acceleration or jerk peaks to safely play the motion on a robot.

In this paper, we propose a method aimed at shortening path length after a path planning step. Note that we do not address path planning, but that we take the result of a probabilistic motion planner as the input to our path optimization method.

For this shortening purpose, random shortcut (RS) methods are still very popular [9, 10, 11]. However, RS requires fine tuning of the termination condition and is not efficient for long trajectories where only a minor part needs to be optimized, as in Figure 1. Figure 2 presents another situation where RS will always fail to optimize the initial path, since it cannot decouple the robot degrees of freedom (DOF) on which the optimization occurs. This problem is addressed by our method. Processing a path pruning [10], in order to remove redundant nodes from the initial path, is a classical preliminary step for path length shortening. A pruning will efficiently solve the example in Figure 1, however it will fail tackling the issue in Figure 2, as RS.

On the other hand, numerical optimization methods like CHOMP [12] can be used as a post-processing step. They have clear termination conditions, but collision avoidance is handled by inequality constraints sampled at many points

*Corresponding author. Email: mcampana@laas.fr

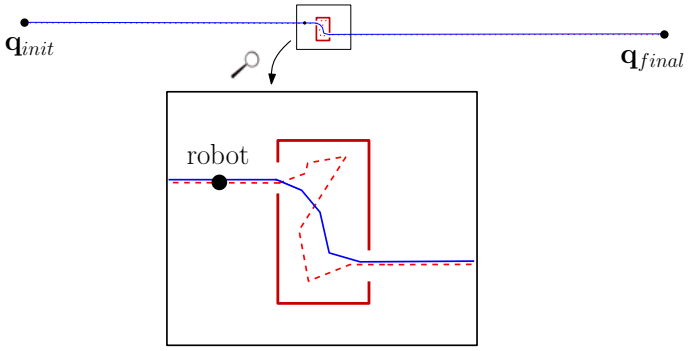


Figure 1: Case of a long initial path from $\mathbf{q}_{init}$ to $\mathbf{q}_{final}$ (above) containing a small part that can be optimized (below). Random shortcut is unlikely to optimize the initial dashed part containing detours in the box, whereas our method succeeds (in blue). This type of issue is common in navigation problems, where environments contain long corridors.

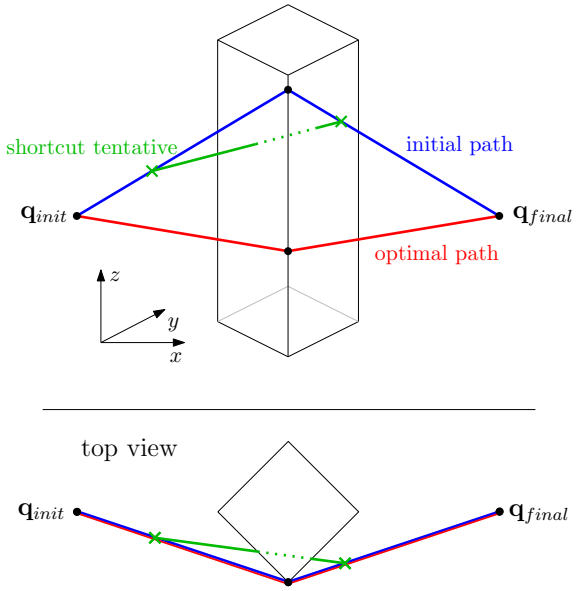


Figure 2: Example of a path in $\mathbb{R}^3$. The optimal path belongs to the $x-y$ plane containing $\mathbf{q}_{init}$ and $\mathbf{q}_{final}$. Random shortcut will never manage to optimize the initial path: each shortcut attempt will provide a collision.

along the trajectory. These methods therefore require a pre-processing step of the robot (and/or environment) model in order to make it simpler: [12] covers PR2 bodies with spheres, while [13] needs to decompose objects into convex subsets.

Finally it should be noticed that optimality in robot motion is a notion that should be clarified. Most of the time motion planners provide an optimized motion, which is not optimal at all, but is the output of a given optimization method. When optimal motions exist, numerical algorithms mostly fail in accounting for their combinatorial structure. In addition, optimization algorithms bypass (not overcome) the question of the existence of optimal motions [14]. In that perspective, a path optimization algorithm has to be evaluated with respect to other existing optimization techniques, from qualitative properties and from computational performance.

The idea of our method is to find a good trade-off between the simplicity of blind methods like shortcut algorithms, and the complexity of distance based optimization techniques. The method iteratively shortens the initial path with gradient-based information. When a collision is detected at a given iteration, the method backtracks to the latest valid iteration and inserts a one-dimensional constraint between the objects detected in collision. Only collisions between objects are evaluated, therefore no pre-processing of either the robot or environment models is necessary to increase distance computation speed. Respecting the problem geometry also preserves that a solution can still be found, e.g. for narrow passages as holes or grippers. The method is also repeatable since no randomness is introduced. The underlying optimization algorithm is a Linearly Constrained Quadratic Program (LCQP).

Another important feature of our contribution is that we optimize paths on the robot configuration space in a proper mathematical way. Most other optimization algorithms represent $SO(3)$ rotations by a vector directed along the rotation axis and the norm of which is the rotation angle, also known as the exponential map of $SO(3)$, or even worse by Euler angles.

Related work is presented in Section II. Section III explains how the path-optimizer works, from the formulation of the problem to the implemented algorithm. Finally, we conclude on experimental results in Section IV.

2 RELATED WORK

Previous work on path optimization has been conducted. CHOMP algorithm [12] optimizes an initial guess provided as input. It minimizes a time invariant cost function using efficient covariant Hamiltonian gradient descent. The cost is quantified by non-smooth parts (with high velocities) and an obstacle avoidance term, provided by the distance to the nearest obstacle for each iteration of the trajectory. Calculating these nearest distances however is time-consuming because the distances between all pairs of objects must be computed at each time step along the path. To reduce the computation time, the method starts by building offline a map of distances that will be called during the optimization at the requested time. Besides, meshes are pre-processed into bounding spheres so that distances are computed faster at the cost of a geometry approximation.

STOMP method [15] avoids computing an explicit gradient for cost optimization using a stochastic analysis of local random samples. But as for CHOMP, the obstacle cost term requires a voxel map to perform its Euclidean Distance Transforms, and represents the robot bodies with overlapping spheres. Such technique provides lots of distance and penetration information but remains very time consuming and is not as precise as some distance computation techniques based on the problem meshes as Gilbert-Johnson-Keerthi [16].

Some optimization-based planners may not require an initial guess but some naive straight-line manually or randomly-sampled initialization as TrajOp [13]. The path is iteratively optimized with sequential convex optimization by minimizing at each step its square length, linear and non-linear constraints considered as penalties. To compute the collision-constraints, nearest obstacle distances are calculated at each discrete time of the trajectory vector. This can be a burden for a high-dimensional robot or a complex environment as we propose to use, and may be compensated with a short path composed of only one or two waypoints.

The elastic strips framework [17] is also an optimization based method. The path is modeled as a spring and obstacles give rise to a repulsive potential field. Although designed for on-line control purposes, this method may be used for path shortening. In this case however, the number of distance computation is very high. The authors also proposed to approximate the robot geometry by spheres.

Some heuristics use random shortcuts on the initial guess combined with a trajectory re-building. For example, [11] returns smooth shortcuts made of parabola and line combinations, relying on the classical bang-bang control approach. These local refined trajectories are time-optimal since they comply with acceleration and velocity constraints. Also based on random shooting, [18] is guiding configuration-generation with local holonomic considerations. Nevertheless this method remains only locally optimal, and is not addressing high-DOF problems. [10] is using medial axis retraction for clearance and a PRS (shortcut is applied only on some random DOF) which can address the problem of Figure 2. However it is relatively slower than RS, and [10] only investigates it for freeflyer robots. Furthermore PRS is not taking advantage of information returned by the checker of which links are colliding, in order to guide the selection of a relevant group of DOF to shortcut.

In some way, our method shares similarities with [19] since this latter work relies on collision checking and backtracks when an iteration is detected in collision, instead of trying to constantly satisfy distance constraints. Unlike our method however, collision constraints are handled by interpolating configurations which, at some points of the trajectory, freeze the whole robot configuration instead of a pertinent subpart. Thus, this method cannot solve Figure 2 issue.

3 PATH OPTIMIZATION

This section describes the establishment of the Gradient-based optimizer. The method works as a classical LCQP, reducing the path length expressed as a cost function and avoid collisions with linearized constraints. Details of the LCQP elements will be given in the following subsections, and associated to functions that will populate the algorithm, presented in the last section.

Name	dimension	config space	velocity
translation	1	$\mathbb{R}$	$\mathbb{R}$
bounded rotation	1	$\mathbb{R}$	$\mathbb{R}$
unbounded rotation	2	$\mathbf{S}^1 \subset \mathbb{R}^2$	$\mathbb{R}$
$SO(3)$	4	$\mathbf{S}^3 \subset \mathbb{R}^4$	$\mathbb{R}^3$

Table 1: Translation and rotation joint positions are defined by one parameter corresponding respectively to the translation along an axis and a rotation angle around an axis. Unbounded rotation is defined by a point on the unit circle of the plane: two parameters corresponding to the cosine and the sine of the rotation angle. $SO(3)$ is defined by a unit quaternion. The velocity of translation and bounded rotation joints is the derivative of the configuration variable. The velocity of an unbounded rotation joint corresponds to the angular velocity. The velocity of a $SO(3)$ joint is defined by the angular velocity vector $\omega \in \mathbb{R}^3$.

3.1 Kinematic chain

A robot is defined by a kinematic chain composed of a tree of joints. We denote by $(J_1, \cdots, J_m)$ the ordered list of joints. Each joint J_i, $i \in \{1..m\}$, is represented by a mapping from a sub-manifold of $\mathbb{R}^{n_i}$, where n_i is the dimension of J_i in the configuration space, to the space of rigid-body motions $SE(3)$. The rigid-body motion is the position of the joint in the frame of its parent. In the examples presented in this paper, we consider four types of joints described in Table 1.

A configuration of the robot

$$\mathbf{q} = (\underbrace{q_1, \cdots, q_{n_1}}_{J_1}, \underbrace{q_{n_1+1}, \cdots, q_{n_1+n_2}}_{J_2}, \cdots q_n),\ n \triangleq \sum_{i=1}^{m} n_i$$

is defined by the concatenation of the joint configurations. The configuration space of the robot is denoted by $\mathscr{C} \subset \mathbb{R}^n$. Note that the configuration of the robot belongs to a sub-manifold of $\mathbb{R}^n$.

The velocity of each joint J_i, $1 \leq i \leq m$, belongs to the tangent space of the joint configuration space, and is defined by a vector of $\mathbb{R}^{p_i}$, where p_i is the number of DOF of J_i. Note that the velocity vector does not necessarily have the same dimension as the configuration vector.

The velocity of the robot is defined as the concatenation of the velocities of each joint:

$$\dot{\mathbf{q}} = (\underbrace{\dot{q}_1, \cdots, \dot{q}_{p_1}}_{J_1}, \underbrace{\dot{q}_{p_1+1}, \cdots, \dot{q}_{p_1+p_2}}_{J_2}, \cdots \dot{q}_p),\ p \triangleq \sum_{i=1}^{m} p_i$$

3.1.1 Operations on configurations and vectors

By analogy with the case where the configuration space is a vector space, we define the following operators between configurations and vectors:

$$\mathbf{q}_2 - \mathbf{q}_1 \in \mathbb{R}^p,\ \ \mathbf{q}_1, \mathbf{q}_2 \in \mathscr{C}$$

is the constant velocity moving from $\mathbf{q}_1$ to $\mathbf{q}_2$ in unit time, and

$$\mathbf{q} + \dot{\mathbf{q}} \in \mathscr{C},\ \ \mathbf{q} \in \mathscr{C},\ \dot{\mathbf{q}} \in \mathbb{R}^p$$

is the configuration reached from $\mathbf{q}$ after following constant velocity $\dot{\mathbf{q}}$ during unit time.

Note that the definitions above stem from the Riemanian structure of the configuration space of the robot. The above sum corresponds to the exponential map. One can easily state that "following a constant velocity" makes sense for the four types of joints defined in Table 1. We refer to [20] Chapter 5 for details about Riemanian geometry.

3.2 Straight interpolation

Let $\mathbf{q}_1, \mathbf{q}_2 \in \mathscr{C}$ be two configurations. We define the straight interpolation between $\mathbf{q}_1$ and $\mathbf{q}_2$ as the curve in $\mathscr{C}$ defined on interval $[0,1]$ by:

$$t \rightarrow \mathbf{q}_1 + t(\mathbf{q}_2 - \mathbf{q}_1)$$

This interpolation corresponds to the linear interpolation for translation and bounded rotations, to the shortest arc on $\mathbf{S}^1$ for unbounded rotation and to the so called slerp interpolation for $SO(3)$.

3.3 Problem definition

3.3.1 Optimization variables

We consider as input a path composed of a concatenation of straight interpolations between $wp+2$ configurations: $(\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{wp+1})$. This path is the output of a random sampling path planning algorithm between $\mathbf{q}_0$ and $\mathbf{q}_{wp+1}$. We wish to find a sequence of waypoints $\mathbf{q}_1$,...,$\mathbf{q}_{wp}$ such that the new path $(\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{wp+1})$ is shorter and collision-free. Note that $\mathbf{q}_0$ and $\mathbf{q}_{wp+1}$ are unchanged and that the workspace of the robot contains obstacles. We denote by $\mathbf{x}$ the optimization variable:

$$\mathbf{x} \triangleq (\mathbf{q}_1, \cdots, \mathbf{q}_{wp})$$

Each path $\mathbf{x}$ is a mapping from interval $[0,1]$ into $\mathscr{C}$: $\mathbf{x}(0) = q_0$, $\mathbf{x}(1) = q_{wp+1}$. Finally, a continuous collision checker inspired of [21] is used to validate paths. It also returns the first colliding configuration and its abscissa along the path.

3.3.2 Cost

Let $W \in \mathbb{R}^{p \times p}$ be a diagonal matrix of weights:

$$W = \begin{pmatrix} w_1 I_{p_1} & & 0 & \\ & w_2 I_{p_2} & & \\ & & \ddots & \\ 0 & & & w_m I_{p_m} \end{pmatrix}$$

where I_{p_i} is the identity matrix of size p_i and w_i is the weight associated to the joint J_i. We define the length of the straight interpolation between two configurations as:

$$\|\mathbf{q}_2 - \mathbf{q}_1\|_W \triangleq \sqrt{(\mathbf{q}_2 - \mathbf{q}_1)^T W^2 (\mathbf{q}_2 - \mathbf{q}_1)}$$

Weights are used to homogenize translations and rotations in the velocity vector. For a rotation, the weight is equal to the maximal distance of the robot bodies moved by the joint to the center of the joint, in a given configuration. For a translation, it is equal to 1.

Given $\mathbf{q}_0$ and $\mathbf{q}_{wp+1}$ fixed, the cost we want to minimize is defined by

$$C(\mathbf{x}) \triangleq \frac{1}{2} \sum_{k=1}^{wp+1} \lambda_{k-1} \|\mathbf{q}_k - \mathbf{q}_{k-1}\|_W^2$$

where the λ_{k-1} coefficients will be explained in the results section. For now it is assumed that $(\lambda_{k-1})_{k \in \{1..wp+1\}}$. Note that C is not exactly the length of the path, but it can be established that minimal length paths also minimize C. This latter cost is better conditioned for optimization purposes.

The gradient of the cost function $\nabla C(\mathbf{x})$ is computed as follows:

$$\nabla C(\mathbf{x}) = \left((\lambda_k (\mathbf{q}_{k+1} - \mathbf{q}_k)^T - \lambda_{k+1} (\mathbf{q}_{k+2} - \mathbf{q}_{k+1})^T) W^2 \right)_{k \in \{0..wp-1\}}$$

From the gradient expression, we notice that the Hessian H is constant:

$$\mathrm{H} = \begin{pmatrix} (\lambda_0 + \lambda_1) W^2 & -\lambda_1 W^2 & 0 & \cdots & & 0 \\ -\lambda_1 W^2 & (\lambda_1 + \lambda_2) W^2 & -\lambda_2 W^2 & 0 & \cdots & 0 \\ 0 & -\lambda_2 W^2 & (\lambda_2 + \lambda_3) W^2 & -\lambda_3 W^2 & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\lambda_{wp-2} W^2 & (\lambda_{wp-2} + \lambda_{wp-1}) W^2 & -\lambda_{wp-1} W^2 \\ 0 & \cdots & \cdots & 0 & -\lambda_{wp-1} W^2 & (\lambda_{wp-1} + \lambda_{wp}) W^2 \end{pmatrix}$$

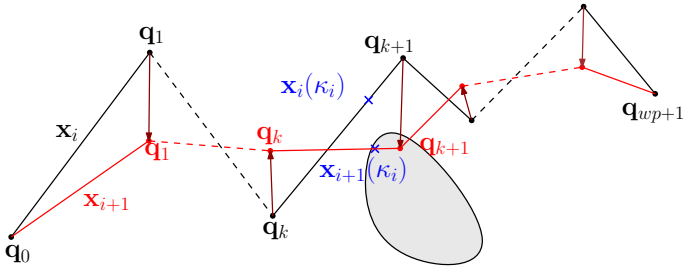


Figure 3: Illustration of one iteration of the path optimization. $\mathbf{x}_{i+1}$ appears to be in collision with the obstacle. The first colliding configuration $\mathbf{x}_{i+1}(\kappa_i)$ at abscissa κ_i is returned by the continuous collision checker. The corresponding constraint will be computed in the backtracked configuration $\mathbf{x}_i(\kappa_i)$.

3.4 Unconstrained resolution

We assume that the direct interpolation between the initial and final configurations contains collisions. An iteration at stage i is described as follow:

$$\begin{aligned} \mathbf{p}_i &= -\mathrm{H}^{-1}\nabla C(\mathbf{x}_i)^T \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \alpha_i \mathbf{p}_i \end{aligned} \tag{1}$$

where $\mathbf{p}_i$ is computed by an unconstrained version of function `computeIterate`, and α_i is a real-valued parameter. Taking $\alpha_i = 1$ yields the unconstrained minimal cost path, i.e. all waypoints aligned on the straight line between $\mathbf{q}_0$ and $\mathbf{q}_{wp+1}$. Since this solution is in collision, we set $\alpha_i = \alpha_{init}$ where α_{init} is a parameter in interval $[0,1]$.

We iterate step (1) until path $\mathbf{x}_{i+1}$ is in collision. When a collision is detected, we introduce a constraint and perform a new iteration from $\mathbf{x}_i$ as explained in the next section.

3.5 Linear Constraints

Let us assume that at iteration i, j linear constraints have been inserted before the current iteration. These constraints are stored as lines of a matrix as follows:

$$\Phi_i = \begin{pmatrix} L_1 \\ \vdots \\ L_j \end{pmatrix}$$

where the step $\mathbf{p}_i$ is constrained to be in the kernel of Φ_i as follows:

$$\Phi_i \mathbf{p}_i = 0$$

These linear constraints are built from the linearization of a collision-constraint function, which will be detailed in the following section.

3.5.1 New constraint

As illustrated in Figure 3, let us denote by κ_i the abscissa of the first collision detected on path $\mathbf{x}_{i+1}$, which previous iteration $\mathbf{x}_i$ was collision-free. Thus in configuration $\mathbf{x}_{i+1}(\kappa_i)$ a collision has been detected. Two cases are possible:

1. the collision occurred between two bodies of the robot: $\mathscr{B}_1$ and $\mathscr{B}_2$, or
2. the collision occurred between a body of the robot $\mathscr{B}_1$ and the environment.

In the rest of this section, the first case only will be considered. Reasoning about the second case is similar, except that the constraint is on the position of $\mathscr{B}_1$ with respect to the environment.

The principle of the method is to compute a linear constraint, initialized on the collision-free configuration $\mathbf{x}_i(\kappa_i)$ to avoid the encountered collision $\mathbf{x}_{i+1}(\kappa_i)$. To handle this, we introduce a one-dimensional constraint based on the orthogonal direction of the encountered collision.

In the collision-configuration $\mathbf{x}_{i+1}(\kappa_i)$, let $\mathbf{P}_c \in \mathbb{R}^3$ be a contact point expressed in the global frame (Figure 4 left). We denote by:

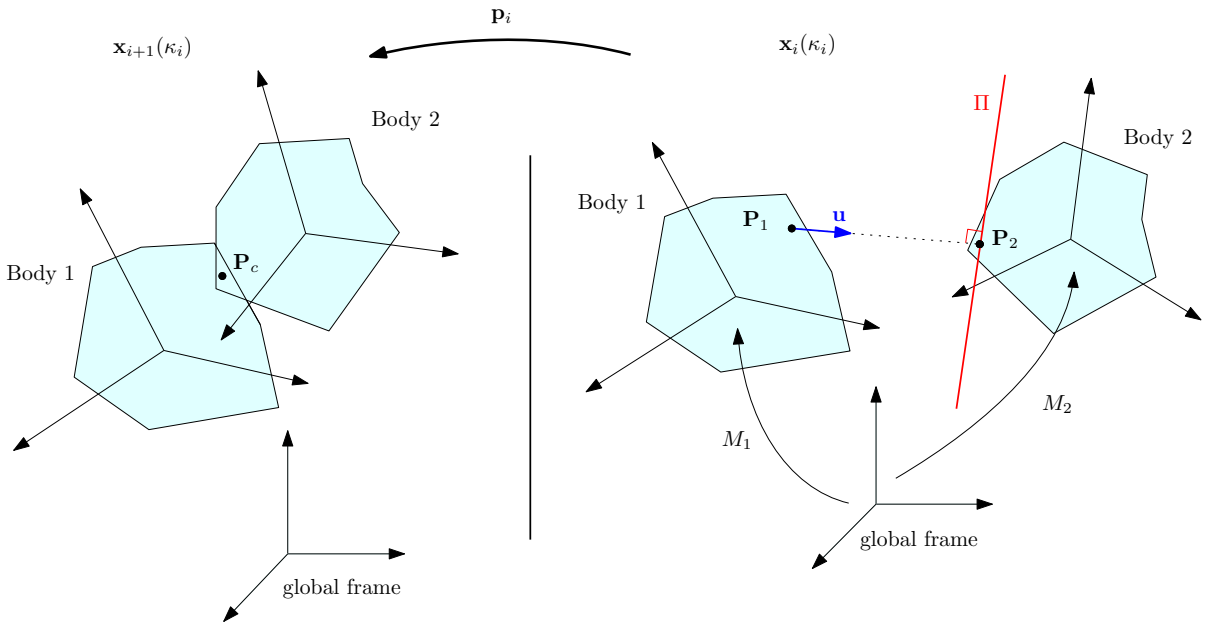


Figure 4: Bodies representation in collision-configuration (left) and backtracked collision-free configuration (right). A contact point (i.e. any point in the intersection of the bodies) $\mathbf{P}_c$ can be returned by the Flexible Collision Library [22], used to detect collisions. Constraint defined by (4) aims at keeping $\mathbf{P}_2(\mathbf{q})$ in plane Π fixed to $\mathscr{B}_1$.

- $\mathbf{P}_1^{loc}$ (resp. $\mathbf{P}_2^{loc}$) the coordinate vector of $\mathbf{P}_c$ in the local frame of $\mathscr{B}_1$ (resp. $\mathscr{B}_2$),
- $M_1(\mathbf{q}) \in SE(3)$ (resp. $M_2(\mathbf{q}) \in SE(3)$) the rigid-body transformation representing the position of $\mathscr{B}_1$ (resp. $\mathscr{B}_2$) in the global frame, in configuration $\mathbf{q}$,
- $M_2^1(\mathbf{q}) = M_1(\mathbf{q})^{-1} M_2(\mathbf{q})$ the position of $\mathscr{B}_2$ local frame in $\mathscr{B}_1$ local frame in configuration $\mathbf{q}$,
- $\mathbf{P}_1(\mathbf{q})$ (resp. $\mathbf{P}_2(\mathbf{q})$) the points moving with $\mathscr{B}_1$ (resp. $\mathscr{B}_2$) of local coordinate $\mathbf{P}_1^{loc}$ (resp. $\mathbf{P}_2^{loc}$) in $\mathscr{B}_1$ (resp. $\mathscr{B}_2$) local frame.

We define $\mathbf{u}$ as the coordinate vector of the unit vector linking points $\mathbf{P}_1$ and $\mathbf{P}_2$ in configuration $\mathbf{x}_i(\kappa_i)$, expressed in local frame of $\mathscr{B}_1$ (Figure 4 right):

$$\mathbf{u} = \frac{M_2^1(\mathbf{x}_i(\kappa_i))\mathbf{P}_2^{loc} - \mathbf{P}_1^{loc}}{\|M_2^1(\mathbf{x}_i(\kappa_i))\mathbf{P}_2^{loc} - \mathbf{P}_1^{loc}\|}$$

Note that $\mathbf{u}$ is well defined since configuration $\mathbf{x}_i(\kappa_i)$ is collision-free.

Let g be the real valued function mapping the projection of vector $\overrightarrow{\mathbf{P}_1\mathbf{P}_2(\mathbf{q})}$ on $\mathbf{u}$ to a configuration $\mathbf{q}$. For any $\mathbf{q} \in \mathscr{C}$:

$$g(\mathbf{q}) = \left(M_2^1(\mathbf{q})\mathbf{P}_2^{loc} - \mathbf{P}_1^{loc} | \mathbf{u}\right) \tag{2}$$

Let f be the function defined from $\mathscr{C}^{wp}$ to $\mathbb{R}$ by:

$$f(\mathbf{x}) = g(\mathbf{x}(\kappa_i)) \tag{3}$$

The constraint defined for any path $\mathbf{x}$ by:

$$f(\mathbf{x}) - f(\mathbf{x}_i) = 0. \tag{4}$$

aims at keeping point $\mathbf{P}_2(\mathbf{q})$ in a plane fixed to $\mathscr{B}_1$, orthogonal to $\mathbf{u}$ and and passing by $\mathbf{P}_2$ in configuration $\mathbf{x}_i(\kappa_i)$ (Figure 4 right).

We linearize the constraint around $\mathbf{x}_i$:

$$\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)(\mathbf{x} - \mathbf{x}_i) = 0$$

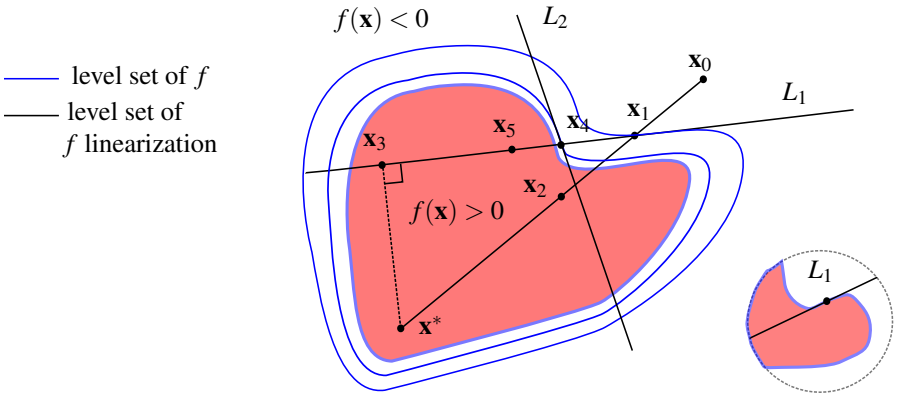


Figure 5: Illustration of linearly constrained quadratic program on a problem analogous to the path shortening problem: we want to minimize the quadratic cost $\frac{1}{2}\|\mathbf{x}-\mathbf{x}^*\|^2, \mathbf{x} \in \mathbb{R}^2$ under the non-linear constraint $f(\mathbf{x}) \leq 0$. $\alpha_{init} = \frac{1}{4}$. The algorithm starts from $\mathbf{x}_0$. The first iterate is $\mathbf{x}_1$ which satisfies the constraint. The second iterate is $\mathbf{x}_2$ that does not satisfy the constraint. The algorithm backtracks to $\mathbf{x}_1$ and inserts linear constraint $L_1 : \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_1)(\mathbf{x}-\mathbf{x}_1) = 0$. $\mathbf{x}_3$ is the global minimum under L_1. As $\mathbf{x}_3$ does not satisfy the constraint, the algorithm moves to $\mathbf{x}_4$ that satisfies the constraints, and then to $\mathbf{x}_5$ that does not. The algorithms backtracks to $\mathbf{x}_4$, inserts constraint $L_2 : \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_2)(\mathbf{x}-\mathbf{x}_2) = 0$, and returns $\mathbf{x}_4$ as a solution since the dimension of the search space is 0. Notice that the same non-linear constraint may give rise to several linear constraints. The convergence of the algorithm relies on the fact that the kernel of constraint $L2$ is not contained in the kernel of the current constraints (L_1 only here). The convergence analysis can be roughly summarized as follows. $L2$ to be linearly dependent of L_1 requires that f is stationary along L_1 at $\mathbf{x}_4$. This is unlikely (but possible) since $\mathbf{x}_4$ is not far from the boundary of the domain defined by $f(\mathbf{x}) \leq 0$. If L_2 was not linearly independent from L_1, The algorithm would keep searching new iterates between $\mathbf{x}_5$ and $\mathbf{x}_4$. If by any chance constraint f linearized around each of those collision-free iterates was each time linearly dependent from L_1 the iterates would converge to the boundary of the domain defined by $f(\mathbf{x}) \leq 0$. By continuous differentiability of f, this would mean that f is stationary at the boundary. In other words, the straight line passing by $\mathbf{x}_1$ and $\mathbf{x}_3$ would cross the boundary tangentially (as in bottom right picture). This is possible but unlikely, unless the problem has been defined as such on purpose.

The computation of the linearized constraint is described in the next section. Then, a line is added in the constraint Jacobian matrix Φ_i:

$$\Phi_{i+1} = \begin{pmatrix} L_1 \\ \vdots \\ L_{j+1} \end{pmatrix} \text{ with}$$
$$L_{j+1} = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)$$

This stage is performed by function `addCollisionConstraint`.

Finally, we refer to [23] for solving LCQP. The step computation corresponds to a constrained version of `computeIterate`.

3.5.2 Linearized constraint computation

Let $\mathbf{q}_{k,i}$ denote the waypoint k along path $\mathbf{x}_i$. There exist $\beta \in [0,1]$ and k such that $\mathbf{x}_i(\kappa_i)$ can be written as a combination of two waypoints:

$$\mathbf{x}_i(\kappa_i) = \mathbf{q}_{k,i} + \beta(\mathbf{q}_{k+1,i} - \mathbf{q}_{k,i})$$

Thus the linearized constraint Jacobian $\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)$ is built by matrix blocks using the Jacobian $\frac{\partial g}{\partial \mathbf{q}}$ expressed in each of the two waypoints and β. This step is performed by `computeCollisionConstraint`.

3.6 Convergence Analysis and algorithm refinement

Although linearized constraints may differ from the initial geometrically relevant non-linear constraint when the iterate goes away from the linearization path, we will show in this section that our algorithm converges under some reasonable assumptions. The underlying idea of the proof is sketched in Figure 5.

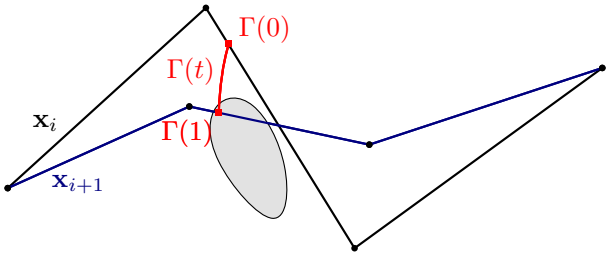


Figure 6: Representation in the robot configuration space of the trajectory Γ, defined in Equation (8).

From the definition of Φ_i, it is straightforward that:

$$\text{Ker } \Phi_{i+1} \subset \text{Ker } \Phi_i \tag{5}$$

In other words, any path iteration complying with the set of constraints contained in Φ_{i+1} will satisfy the set Φ_i. Let us assume that:

$$L_{j+1}\mathbf{p}_i \neq 0 \tag{6}$$

We will elaborate later on this assumption. This means that $\mathbf{p}_i \notin \text{Ker } \Phi_{i+1}$, and as $\mathbf{p}_i \in \text{Ker } \Phi_i$, then:

$$\text{Ker } \Phi_i \neq \text{Ker } \Phi_{i+1}$$

From (5), we deduce that:

$$\dim(\text{Ker } \Phi_{i+1}) < \dim(\text{Ker } \Phi_i)$$

This result proves that under Assumption (6), each additional constraint is linearly independent from the previous ones. Thus, the dimension of search space decreases and our algorithm terminates in a finite number of iterations.

3.6.1 Note about Assumption (6)

As in the previous section, we assume that $\mathbf{x}_i$ is collision-free and that $\mathbf{x}_{i+1}$ is in collision at abscissa κ_i. According to (3), the evaluation of the constraint function f along the iteration line $\mathbf{x}_i + t\alpha_i\mathbf{p}_i$, $t \in [0,1]$ going from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$ can be written as follows:

$$f(\mathbf{x}_i + t\alpha_i\mathbf{p}_i) = g\left((\mathbf{x}_i + t\alpha_i\mathbf{p}_i)(\kappa_i)\right) \tag{7}$$

The argument of function g above is a trajectory in the robot configuration space that we denote by Γ:

$$\Gamma(t) = (\mathbf{x}_i + t\,\alpha_i\mathbf{p}_i)(\kappa_i),\ t \in [0,1] \tag{8}$$

The trajectory Γ is defined by taking the constant abscissa κ_i and by moving from path $\mathbf{x}_i$ along step $\mathbf{p}_i$ (see Figure 6). Note that configuration $\mathbf{x}_{i+1}(\kappa_i)$ is reached when t is equal to 1. Substituting (8) into (7) and differentiating with respect to t yields

$$f(\mathbf{x}_i + t\alpha_i\mathbf{p}_i) = g(\Gamma(t)) \tag{9}$$

$$\alpha_i \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i + t\alpha_i\mathbf{p}_i)\mathbf{p}_i = \frac{d}{dt}g(\Gamma(t)) \tag{10}$$

Property 1 *From the definition (2) of g, the right hand side of (10) represents the velocity of point* $\mathbf{P}_2$ *in reference frame* $\mathscr{B}_1$ *projected on vector* $\mathbf{u}$ *along trajectory* Γ.

Therefore the following expressions

$$\alpha_i \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_i)\mathbf{p}_i = \frac{d}{dt}g(\Gamma(0)) \qquad\qquad \alpha_i \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{i+1})\mathbf{p}_i = \frac{d}{dt}g(\Gamma(1))$$

correspond to $(\mathbf{v}_{\mathbf{P}_2/\mathscr{B}_1}|\mathbf{u})$, respectively in configurations $\mathbf{x}_i(\kappa_i)$ and $\mathbf{x}_{i+1}(\kappa_i)$, where $\mathbf{v}_{\mathbf{P}_2/\mathscr{B}_1}$ represents the velocity of point $\mathbf{P}_2$ in reference frame $\mathscr{B}_1$. Note that Assumption (6) is equivalent to the first above equality to be different from 0. According to the geometric interpretation, Assumption (6) is violated if and only if $\mathbf{v}_{\mathbf{P}_2/\mathscr{B}_1}$ is orthogonal to $\mathbf{u}$. Although very unlikely, this case might appear for some new constraint. In this case, inserting constraint L_{j+1} is useless since

$$\text{Ker } \Phi_i = \text{Ker } \Phi_{i+1}$$

3.6.2 Algorithm refinement

When the new constraint is not linearly independent from the set of previous constraints, the algorithm enters an additional loop, performed by Algorithm 1, in order to find a new constraint that is linearly independent. The loop keeps looking for paths along line segment $[\mathbf{x}_i, \mathbf{x}_{i+1}]$ by dichotomy. A pair containing the latest free path and the latest path in collision, denoted by $(\mathbf{x}_{Free}, \mathbf{x}_{Coll})$ is stored along the loop. New iterations are chosen in the middle of this pair.

- If the new path is collision-free, it replaces $\mathbf{x}_{Free}$ in the pair.
- If the new path is in collision, it replaces $\mathbf{x}_{Coll}$ in the pair.

In both cases, a new constraint is built following the method described in Section 3.5.1. Two cases are then possible:

1. at some point in the loop the new constraint is linearly independent from the previous constraints. The new constraint is added to Φ_i to give rise to Φ_{i+1}, and the loop is interrupted, or
2. each new constraint is linearly dependent from the previous constraints and the loop never ends.

In the second case, the iterations of the loop converge to a path that we denote by $\bar{\mathbf{x}}$. $\mathbf{x}_{Free}$ and $\mathbf{x}_{Coll}$ also both converge to $\bar{\mathbf{x}}$. $\bar{\mathbf{x}}$ necessarily lies at the boundary between free paths and paths in collision. Let us denote by $\bar{\kappa}$ the abscissa along $\bar{\mathbf{x}}$ where $\mathscr{B}_1$ and $\mathscr{B}_2$ come to contact, and let us denote by $\bar{\mathbf{P}}$ the contact point.

At each iteration, the new linear constraint

$$L_{j+1} = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{Free})$$

is tested. As iterations $\mathbf{x}_{Free}$ and $\mathbf{x}_{Coll}$ tend toward $\bar{\mathbf{x}}$, it can be established by a geometric reasoning analogous to Property 1, that $L_{j+1}\mathbf{p}_i$ tends to the norm of the velocity of point $\bar{\mathbf{P}}$ belonging to $\mathscr{B}_2$ in the frame of $\mathscr{B}_1$. The following property summarizes the result of this section.

Property 2 *As long as along iteration $\mathbf{p}_i$, the trajectory $(\mathbf{x}_i + t\alpha_i\mathbf{p}_i)(\bar{\kappa})$ does not enter in collision with contact point velocity equal to 0 in the frame of $\mathscr{B}_1$, Algorithm 1 converges in a finite number of steps.*

The above property means that the gradient-based algorithm described in this paper converges in all cases, except for ill-defined problems.

3.7 Algorithm

The gradient-based path-optimizer is described in Algorithm 2. Note that the LCQP optimal step $\mathbf{p}$, computed by `computeIterate`, is known. The collision detection on a path is handled by `validatePath`, which returns *true* if the given path is collision-free.

Algorithm 1 Description of `findNewConstraint()` which returns a linearly independent constraint w.r.t. previous constraints stacked in Φ.

Input: $(\mathbf{x}_{Free}, \mathbf{x}_{Coll})$ latest collision-free and in collision paths, $\mathbf{p}$ and α such that $\mathbf{x}_{Coll} \leftarrow \mathbf{x}_{Free} + \alpha\,\mathbf{p}$

Output: linearized constraint $\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{Free})$

Require: constraint $\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{Free})$ built from $\mathbf{x}_{Coll}$ would produce a rank loss in constraint Jacobian Φ

```
solved ← false
while (not(solved)) do
    α ← 0.5α
    x ← x_Free + α p
    if (validatePath(x)) then
        x_Free ← x
    else
        x_Coll ← x
    ∂f/∂x(x_Free) ← computeCollisionConstraint(x_Coll, x_Free)
    solved ← isFullRank(Φ , ∂f/∂x(x_Free))
```

Algorithm 2 Gradient-based (GB) algorithm for path-optimization.

Input: path to optimize $\mathbf{x}_0$
Output: optimized collision-free path $\mathbf{x}_0$

```
α ← α_init
minReached ← false
while (not(noCollision and minReached)) do
    p = computeIterate()
    minReached = (||p|| < 10^-3 or α = 1)
    x_1 ← x_0 + α p
    if (not(validatePath(x_1))) then
        noCollision ← false
        if (α ≠ 1) then
            computeCollisionConstraint(x_1, x_0)
            findNewConstraint()
            addCollisionConstraint()
            α ← 1
        else
            α ← α_init
    else
        x_0 ← x_1
        noCollision ← true
return x_0
```

The idea of the algorithm is to process iterations that reduce the path length according to the LCQP cost. If collision-constraints have been added to the LCQP, further iterations will comply with them. The algorithm stops when the LCQP minimum is reached and collision-free.

One main difficulty is to handle the scalar parameter α determining how much of the computed step will be traveled along. As presented in Algorithm 2, α takes two values, $\alpha_{init} < 1$ to process small steps, or 1 to go directly to the optimum under the latest set of constraints. This latter case is interesting since, if this optimal path is collision-free, the algorithm has converged and returns the path as the solution. Choosing to travel small steps from a valid path decrease the chances of being in collision. Besides if one occurs, the collision-constraint is computed on the last valid path, which is not too much deformed compared to the path that has collisions. As a result, collision-constraints are only computed (`computeCollisionConstraint`) and added (`addCollisionConstraint`) when performing a reduced iteration (i.e. $\alpha = \alpha_{init}$).

Note that even though the constraints are linearized, the algorithm converges.

4 RESULTS

This part gathers optimization results performed on the planning software Humanoid Path Planner [24]. Initial paths are obtained with two kinds of probabilistic planners: Visibility-PRM [25] and RRT-connect [26]. We denote them by PRM and RRT respectively. Unless another value is provided, α_{init} is set to 0.2. A further section provides a discussion on the α_{init} value setting.

4.1 From 2D basic examples

Figure 7 shows the result of our optimizer on 2D cases. Contact points which have led to constraints are represented. They permit to understand how the path is kept out of the obstacles while reducing detours. Note that, since not obstacle clearance is considered, the robot may pass close to obstacles.

Figure 1 illustrates a *very long* path example which RS or PRS will not manage to optimize in an affordable time, because of probabilistically failing to sample configurations in the box. The GB method succeeds to optimize the path contained in the box, with the following cost coefficients:

$$\lambda_{k-1} = \frac{1}{\sqrt{(\mathbf{q}_{k,0} - \mathbf{q}_{k-1,0})^T W^2 (\mathbf{q}_{k,0} - \mathbf{q}_{k-1,0})}},\ k \in \{1..wp+1\}$$

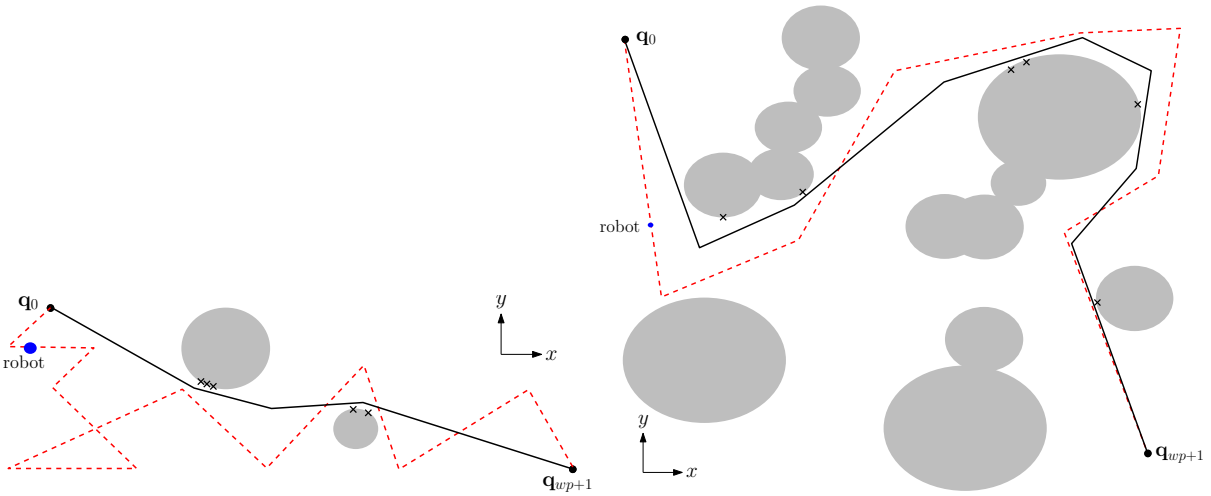


Figure 7: Path-optimization results on 2D robots, moving around gray obstacles. Initial paths are dashed and crosses represent contact points $\mathbf{P}_c$ related to collision-constraints. Note that, on the left, the detour completely disappears.

aiming at keeping the same ratio between path segment lengths at minimum as at initial path, represented by the waypoints $(\mathbf{q}_{k,0})_{k\in\{0..wp\}}$. Without these coefficients, the path that minimizes the cost corresponds to a straight line with the waypoints equidistantly allocated. This is not relevant for Figure 1 type of problems where a local passage is very constrained by obstacles. Note that this cost is also working with all other examples presented in this paper, and provides better quality results than the original cost.

4.2 To 3D complex problems

4.2.1 Comparison to random shortcut algorithms

Our algorithm is also experimented on more complex robots and environments[1]. In Figures 9, 8, 12 and 13, we present multiple situations where the GB algorithm is tested and compared to RS and PRS. After describing the random optimizers, we will present each benchmark and its qualitative path results. Then quantitative convergence graphs and averages will be given and discussed.

The RS implementation is given in Algorithm 3. RS shortens the path by randomly sampling configurations along it, and by trying to link them with collision-free interpolations. The termination condition of RS is a duration time limit t_{lim}, and is typically set as the GB convergence time. Concerning PRS, its implementation is identical to [10]: for a random DOF, configurations are sampled along the initial path as in Algorithm 3. The straight interpolation returns a path made of an interpolation on only the current DOF and based on the previous subpath for other DOF. If this path is collision-free, it is added to the final path as in RS. The process is stopped when the duration exceeds t_{lim}.

Before entering the manipulator examples, the GB algorithm is analyzed on a popular problem in the motion planning literature: a freeflyer puzzle, corresponding to Figure 12(b). The puzzle has to cross down the obstacle using the hole in the middle. The initial path planned with PRM contains detours above and below the obstacle, as well as small superfluous motions in the hole. Results of the three optimizers are similar in terms of path length. Note that for GB, trajectory parts above and below the obstacle are not completely shorten, i.e. the puzzle center is still committing detours. This is the result of adding collision-constraints on these parts of the trajectory, between one of the puzzle branches and the obstacle. In total, 43 collision-constraints have been produced. One idea could be to arbitrarily cancel constraints in these upper and lower parts of the trajectory, and to keep the ones in the hole. However we want the present GB algorithm to remain general and basic, such constraint relaxation is part of the possible future work.

In the double-arms benchmark (4 DOF), Figure 12(a), one arm has to get around a cylinder obstacle while the other arm stays in the same configuration. As expected, the initial path given by RRT activates both arms to solve the problem. Unlike RS, the GB optimizer manages to cancel the rotations of the free arm while optimizing the first arm motion, creating collision-constraints with the cylinder obstacle.

Some problems are involving a 6-axis manipulator arm, also called UR5, equipped with a bar or a gripper. In a

[1] Video of the experimental results is available at https://youtu.be/1MFn0en51qI

Algorithm 3 Random shortcut as adapted from [9] Section 6.4.1. `straightInterpolation` returns the linear interpolation between two configurations. $\mathbf{x}_{|I}$ denotes path $\mathbf{x}$ restricted to interval I. t_{lim} represents the duration limitation of the algorithm.

Input: path to optimize $\mathbf{x}$, time limit t_{lim}
Output: optimized collision-free path $\mathbf{x}$

$t_{start} \leftarrow$ `currentTime()`
$t \leftarrow 0$
while $t < t_{lim}$ **do**
  $failure \leftarrow true$
  $t_1 < t_2 \leftarrow$ random numbers in $[0,1]$
  $lp0 \leftarrow$ `straightInterpolation`$(\mathbf{x}(0), \mathbf{x}(t_1))$
  $lp1 \leftarrow$ `straightInterpolation`$(\mathbf{x}(t_1), \mathbf{x}(t_2))$
  $lp2 \leftarrow$ `straightInterpolation`$(\mathbf{x}(t_2), \mathbf{x}(1))$
  $newPath \leftarrow$ empty path defined on $[0,0]$
  if `validatePath`$(lp0)$ **then**
    $newPath \leftarrow lp0$;
  else
    $newPath \leftarrow \mathbf{x}_{|[0,t_1]}$
  if `validatePath`$(lp1)$ **then**
    $newPath \leftarrow$ `concatenate`$(newPath, lp1)$
  else
    $newPath \leftarrow$ `concatenate`$(newPath, \mathbf{x}_{|[t_1,t_2]})$
  if `validatePath`$(lp2)$ **then**
    $newPath \leftarrow$ `concatenate`$(newPath, lp2)$
  else
    $newPath \leftarrow$ `concatenate`$(newPath, \mathbf{x}_{|[t_2,1]})$
  $\mathbf{x} \leftarrow newPath$
  $t \leftarrow$ `currentTime()` $- t_{start}$
return $\mathbf{x}$

relatively free environment, represented in Figure 12(c), results from our method and RS are similar. Note also that the end-effector trajectory is completely different from the initial one: the robot is easily passing between the meshed spheres, keeping its end-effector above. For an UR5 working in a cluttered environment inspired by an industrial issue Figure 8, GB path optimization efficiently returns a shorter solution, close to the result of RS and to what can be observed in reality [27].

A problem involving a Baxter-like[2] robot manipulating in an office environment is presented in Figure 13(c). The robot starts with its end-effectors above the computer and has to turn and reach the shelf. According to the quality of the left-wrist trajectories, the GB optimizer provides the smoothest motion.

In the three following high-DOF examples involving the PR2 robot (35 DOF), note that results are better in terms of path quality, as a result of the parasite DOF motion removal.

In the example shown in Figure 9, PR2 simply has to cross its arms from the left arm up position to the right arm up one, without any assumption about the group of DOF to activate (i.e. no DOF is locked). The RRT planner returns detours and activates non-useful DOF such as the head, the torso lift and the mobile base. Such behavior induces a high initial path length. RS hardly optimizes the mobile base translation (Figure 9 middle) of the robot and other unnecessary DOF uses. Whereas the GB optimized-path mainly results in moving the arms as expected (Figure 9 right), just creating two collision-constraints between the arms. In the PRS result, only presented in the video, the motion is less optimized than with RS: the arms are moving widely and the mobile base remains activated. One solution, to remove such unnecessary DOF activation, can be to try applying a partial shortcut on each DOF between the initial and final configurations. It appears that this step is more costly in terms of computation time than the GB duration, therefore it cannot be afforded by our PRS implementation, due to the t_{lim} condition. However, this solution could be applied as a preliminary optimization stage for each optimizer.

Similar results are obtained on the PR2 performing manipulation tasks in a kitchen environment. Firstly, the robot moves its hands from the top to the bottom of a table. The different trajectories of the right gripper are indicated in

[2] A torso rotation was added and the grippers were removed.

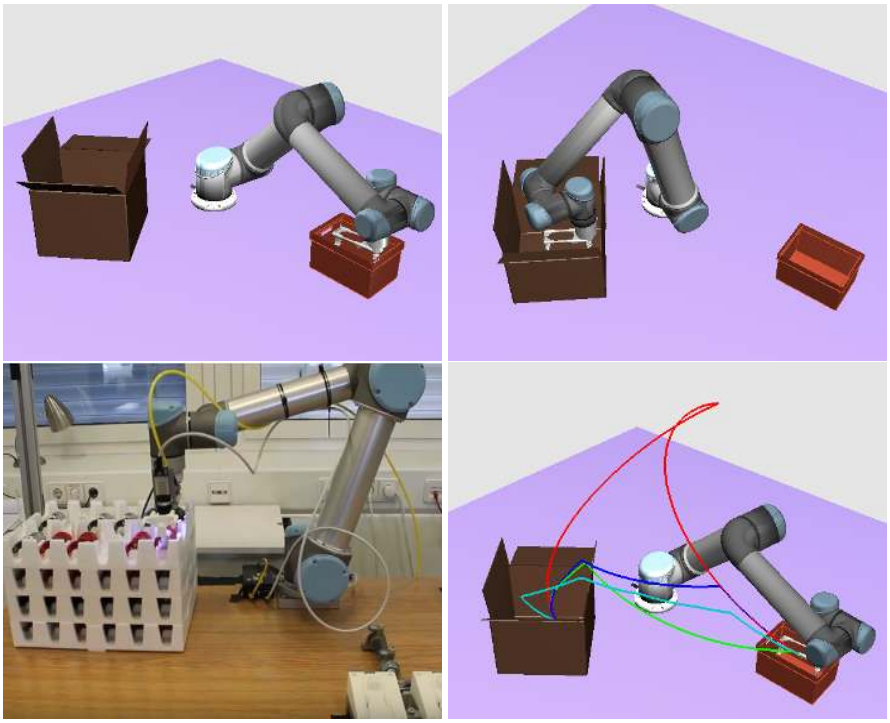


Figure 8: (Bottom left) An industrial use-case example proposed by Philips for the Factory-in-a-Day project [27]. A similar environment has been created (top) to illustrate that our method can comply with an industrial problem, where initial and final configurations of the UR5 are constrained in boxes. End-effector trajectories are illustrated (bottom right): RRT planning in red, a RS optimization in blue, a PRS one in cyan and the GB optimization in green.

Figure 13(a). Our optimizer manages to reduce the initial path length from PRM and improves the path quality just adding constraints between the table and the robot arms. Thus, the robot just slightly moves backward and uses its arm DOF to avoid the table, instead of processing a large motion to move away from the table. Secondly, another example of PR2 going from the set to the fridge door is presented in Figure 13(b) with the mobile base trajectories. Here, GB and RS results are similar in terms of length and rendering.

For some of the presented benchmarks, convergence graphs of the path length reduction are given Figure 14. The chosen initial paths are unchanged, i.e. correspond to Figures 12 and 13. Each graph illustrates the percent ratio of the optimized path length over the initial path length, during the optimization. It is not a surprise that GB is globally slower than RS due to the difference of the computations complexity during the optimization. Thus RS converges faster. However, it seems that GB catches up and overcomes RS before ending (see Figures 14(d) and 14(e)), thanks to the optimization of the mobile base motion. Therefore, it could be interesting to investigate the performance of a composed optimizer, starting by a RS stage until *convergence* and finishing by a GB stage to improve the path length reduction. In the puzzle example (see Figure 14(b)), the difference of optimization speed between GB and the random optimizers is significant. This can be partly explained by the fact that collision checking is rapidly performed in such basic geometry problem. This favors the random shortcut tries while GB spends time on the LCQP resolution.

Since the GB optimizer results depend on the shape of the initial guess, e.g. the number of waypoints and the proximity to obstacles, results averages for 50 initial paths of each benchmark are presented in Table 2. As mentioned, the paths are obtained from PRM or RRT. Due to their nature, these motion planners provide different types of path: the output of PRM contains less waypoints and does not tend to be close to the contact, behavior induced by the extension process of RRT.

In some cases, α_{init} is reduced to comply with very narrow passages, or increased in the opposite case.

The results seem to be consistent with the trajectory analysis and convergence graphs. Except the low-DOF problemz of the puzzle and the UR5, our method provides shorter or similar results compared to RS. Results even seem to be better when the number of DOF increases, as the baxter and PR2 examples. Note that parasite motion disappearance is not necessarily decisive for the path length reduction.

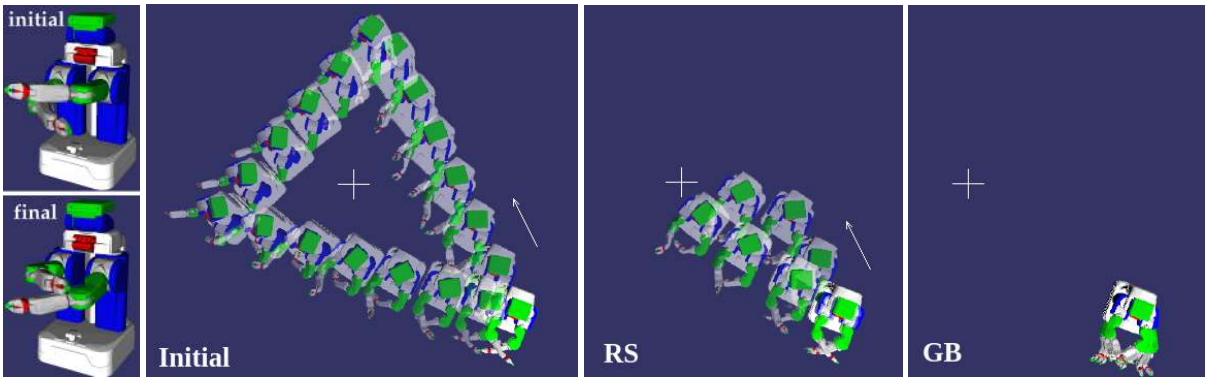


Figure 9: PR2-crossing-arms example: the PR2 robot has just to exchange the positions of its arms (left). The task is simple, however, in absence of explicit indication, any probabilistic motion planner will compute a path that makes the PR2 mobile base purposelessly move around the + marker. Path optimization is expected to remove unnecessary motions. RS fails in this case while GB succeeds.

Problem		Computation time	Relative remaining length (%) GB	RS	PRS
Freeflyer-puzzle	($\alpha_{init} = 0.05$)	742 ms	53.0	41.4	46.1
Double-arms	(RRT)	29.0 ms	44.7	53.6	56.6
UR5-with-spheres	(RRT, $\alpha_{init} = 0.3$)	453 ms	48.5	42.1	72.0
UR5-industrial-example		765 ms	40.3	29.6	43.4
Baxter-in-office		18.8 s	36.5	45.2	79.8
PR2-crossing-arms		882 ms	19.9	43.2	95.2
PR2-in-kitchen-1		13.5 s	28.3	42.7	90.6

Table 2: Average results for 50 runs of several examples presented in the paper. For each run, a solution path is planned by Visibility-PRM (unless 'RRT' for RRT-connect is mentioned) as initial guess for the three optimizers. RS and PRS results correspond to averages of 50 launches of the random optimizers on each initial guess. The GB computation time is the work duration allowed for the random optimizers. $\alpha_{init} = 0.2$ unless another value is specified. Boxes highlight the best path length reduction result among the three optimizers.

4.2.2 Analysis of α_{init} influence

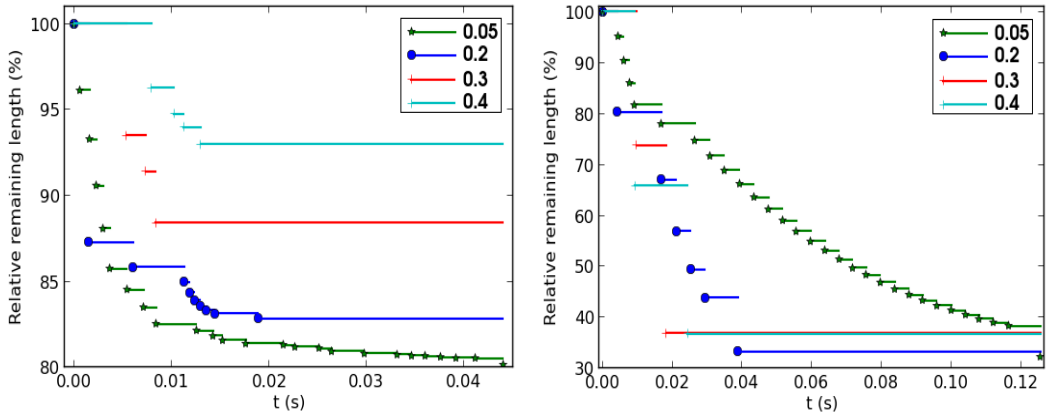
This section deals with the influence of the parameter α_{init} on the GB convergence. Reducing α_{init} makes Algorithm 2 process smaller iterations. Some expected behaviors are visible in Figure 10. For instance, Figure 10(a) illustrates an expected influence of a α_{init} reduction on the final path lengths. Besides, commonly to Figures 10(a) , 10(b) and 10(d), $\alpha_{init} = 0.05$ has the higher convergence time.

However, due to the strong non-linearity of the constraints, reduced iterations do not necessarily lead to a slower but refined solution. GB can stop earlier in local minimum, which may also have a better path length reduction. This is the case of Figure 10(b) where $\alpha_{init} = 0.2$ results in a shorter path than $\alpha_{init} = 0.05$. More surprisingly in Figure 10(b), $\alpha_{init} = 0.05$ yields the worse reduction.

Instead of investigating a way to find a constant α_{init} conditioned by the problem and the initial path, we plan to adapt α_{init} during the optimization process. This can be achieved by taking into account geometrical considerations inspired from the continuous collision checker. For instance, the collision checker is able to return a lower bound of the distance between objects.

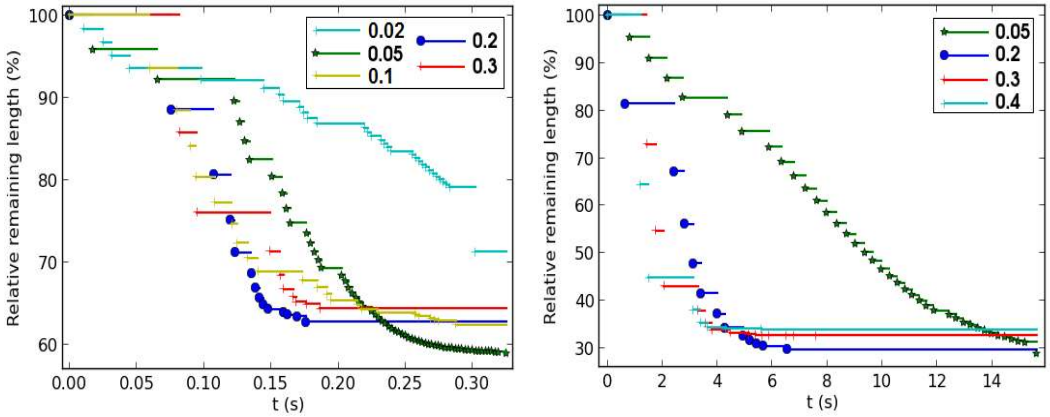
4.2.3 Influence of a pruning preliminary step

For the UR5-industrial benchmark, we compared the optimization convergence graphs with and without a pruning step. Pruning was implemented following [10], to remove redundant nodes in the initial path by creating valid shortcuts between the waypoints. RRT has been chosen as motion planner because it usually produces more waypoints than PRM, so the impact of pruning is more accountable. Results are given in Figure 11. The path length reduction of the three



(a) 2D sliding-robot (Fig. 7 right)

(b) Double-arms



(c) Freeflyer-puzzle

(d) Baxter-in-office

Figure 10: Influence of α_{init} on the convergence graphs of the GB optimizer. For each benchmark, the considered intial paths correspond to the ones of Figures 12 and 13.

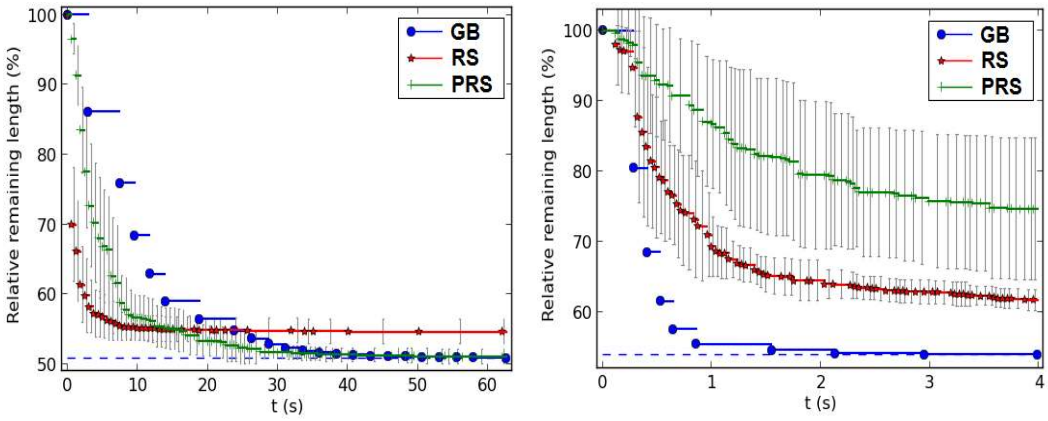
optimizers are still compared, but the notable information is the computation time of GB that is 16 times higher without pruning. Such lower computation time prevents the random optimizers to converge, so the GB result appears as better.

Note that, if pruning always reduces the GB optimization time, it sometimes spoils the path length reduction. In fact, there can be cases where multiple waypoints are useful to bypass an obstacle, rather than a long straight line in the configuration space.

5 CONCLUSIONS

We managed to settle a path optimization for navigation and manipulation problems, and tested it with various robots and environments. Our algorithm uses standard numerical tools as collision checking, linearized one-dimensional constraint and LCQP resolution. It correlates them in a simple but effective way, and the algorithm structure is organized so that its convergence is guaranteed. Furthermore, our method only requires collision checking, therefore neither geometry pre-processing nor offline optimization are necessary to counterbalance costly distance computations. We demonstrate that the optimizer may be time-competitive compared to random shortcut in complex models where collision tests are time-consuming. It also proposes better quality paths, reducing the path length and removing unnecessary DOF motions. Finally, our optimizer manages to reduce a local detour in a long path while random shortcut methods will mostly fail.

For future work, we have room for improvement. We can take advantage of the sparsity of the constraint Jacobian to reduce computation time. We may also adapt the iteration scalar parameter from geometries considerations on the current path, e.g. using a lower bound of the distance between certain objects.



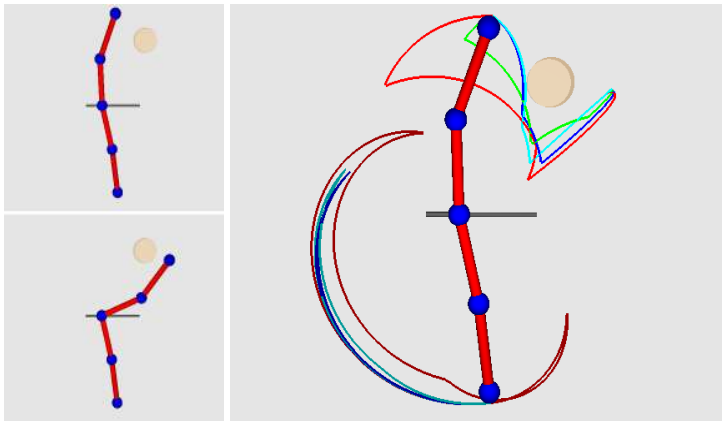
(a) Without pruning

(b) With pruning

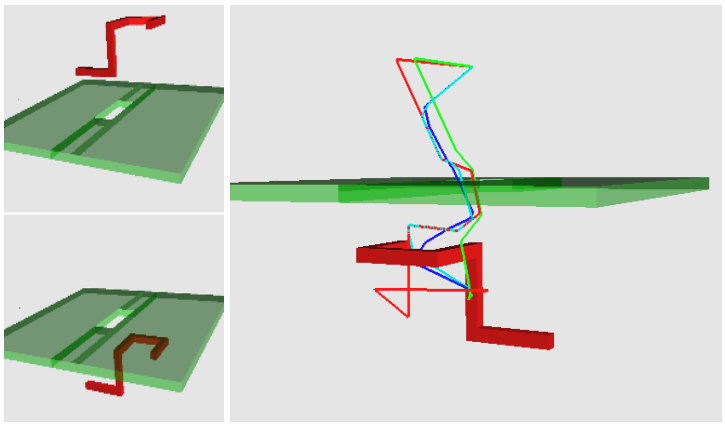
Figure 11: Influence of a pruning step on the optimization processes. RRT-connect provides an initial path with 62 waypoints, which is downed to 2 waypoints by Prune. Concerning the path length, it is only reduced of 6.2% by Prune. Thus, final path lengths provided by GB in both cases are equivalent, the major difference results in the computation time of GB.

Funding

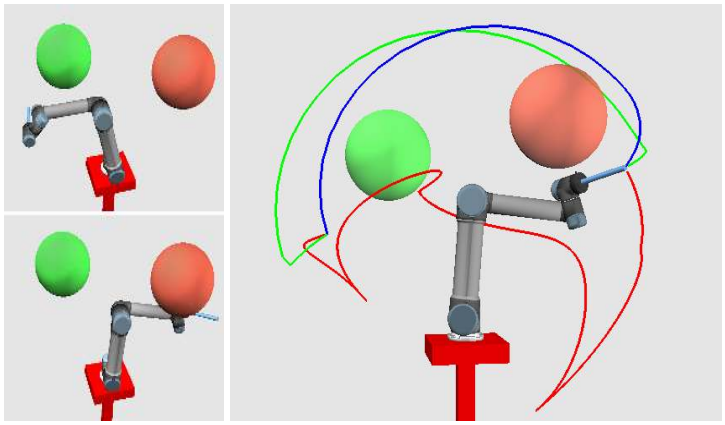
This work is part of the development of LAAS software development kit HPP supported by the project ERC Advanced [grant number 340050] Actanthrope, the FP7 project Factory in a Day [grant number 609206], and the FP7 project EUROC [grant number 608849].



(a) Double-arms

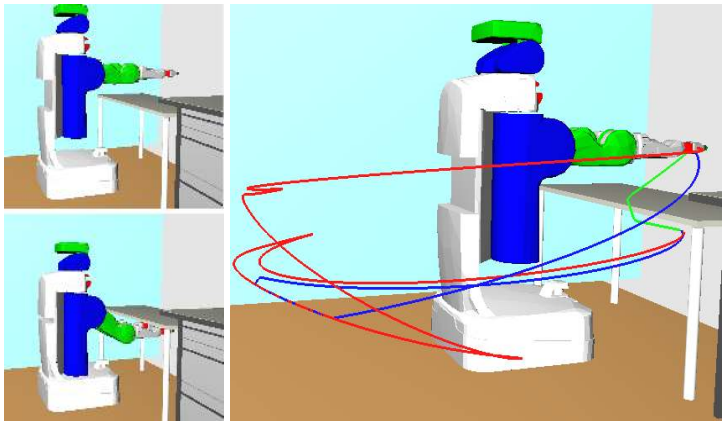


(b) Freeflyer-puzzle

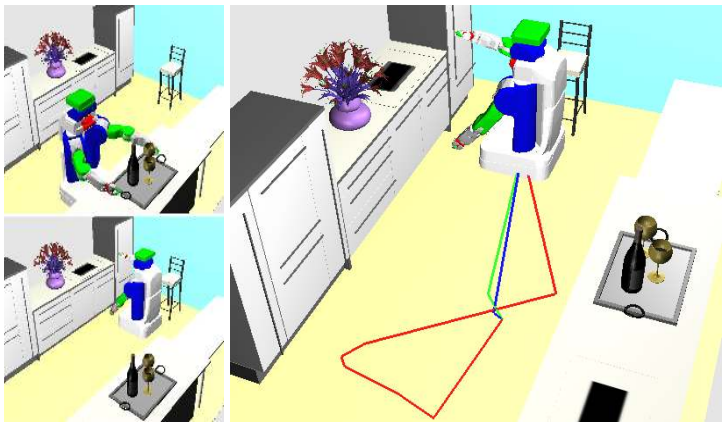


(c) UR5-with-spheres

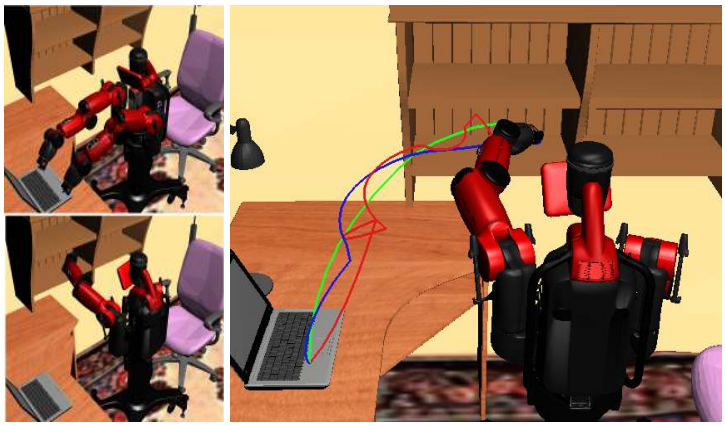
Figure 12: (Left) Initial and final configurations. (Right) Trajectories of end-effectors or centers along the different paths: the initial path is represented in red, the RS output in blue, the PRS output in cyan (top only) and the GB optimized path in green. The full robot motions can also be visualized in the joined video. The trajectory comparison highlights the optimization success of our method, which manages to deliver a shorter or similar path compared to the RS output. Note that, in the double-arms example, GB optimization also cancels the lower arm activation contrary to RS and PRS.



(a) PR2-in-kitchen-1

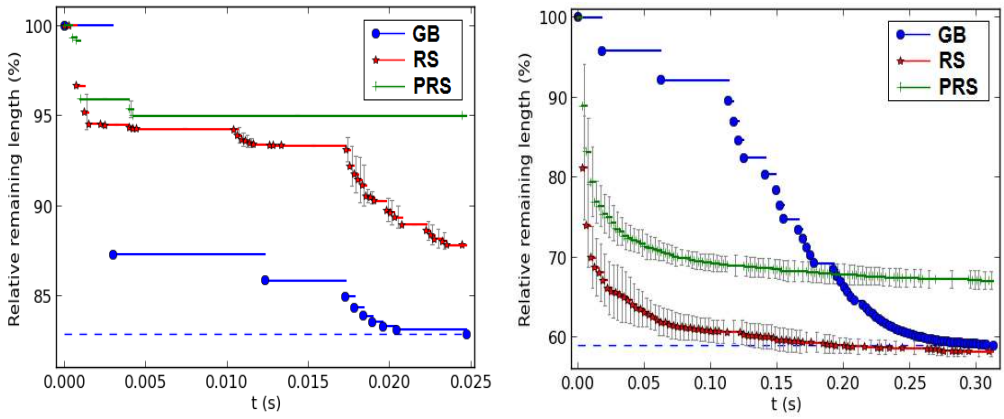


(b) PR2-in-kitchen-2



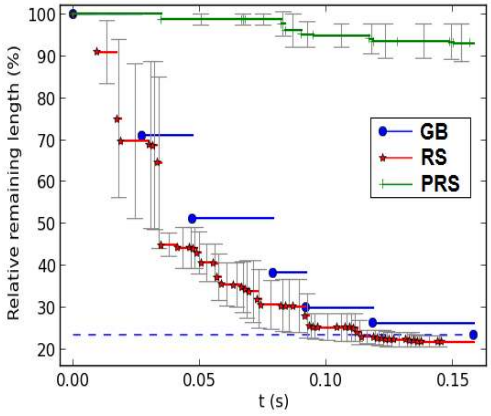
(c) Baxter-in-office

Figure 13: Other trajectories comparisons of end-effectors or mobile bases (initial path in red, RS output in blue, PRS output in cyan and GB output in green).

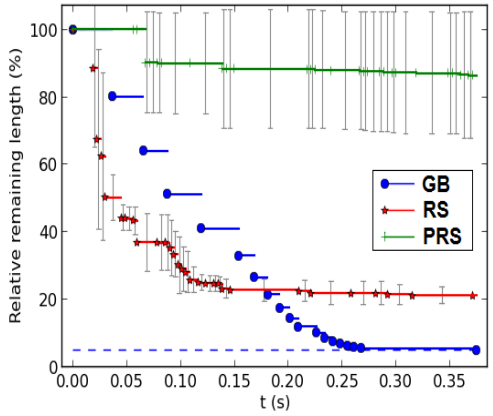


(a) 2D sliding robot (Figure 7)

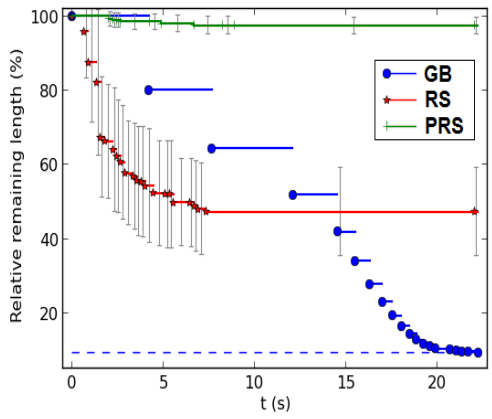
(b) Freeflyer-puzzle



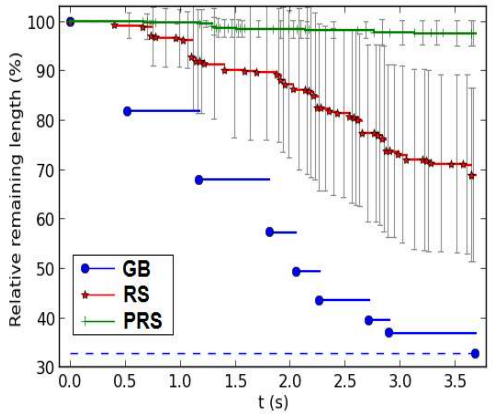
(c) UR5-with-spheres



(d) PR2-crossing-arms



(e) PR2-in-kitchen-1



(f) Baxter-in-office

Figure 14: Convergence graphs of the three optimizers during the optimization process. For each benchmark, the considered initial paths correspond to the ones of Figures 12 and 13. The remaining path length relative to the initial one is represented. The dashed blue line is the final result of GB. RS and PRS averages and standard deviations (in grey) are plotted for 50 launches.

References

[1] M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Pérez, and M. T. Masson. *Robot Motion: Planning and Control*. MIT Press, Cambridge (MA), 1983.

[2] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Rob. Res.*, 10(6):628–649, 1991.

[3] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Rob. Autom.*, 12(4):566–580, 1996.

[4] S. M. LaValle and J. J. Kuffner, Jr. *Algorithmic and Computational Robotics: New Directions*, chapter Rapidly-Exploring Random Trees: Progress and Prospects, pages 293–308. Wellesley (MA), 2001.

[5] C. Park, J. Pan, and D. Manocha. ITOMP: incremental trajectory optimization for real-time replanning in dynamic environments. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 207–215, Atibaia, São Paulo, Brazil, 2012.

[6] M. Garber and M.C. Lin. *Algorithmic Foundations of Robotics V*, chapter Constraint-Based Motion Planning Using Voronoi Diagrams, pages 541–558. Springer Berlin, 2004.

[7] J.T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Cambridge University Press, New York (NY), 2nd edition, 2009.

[8] S. Karaman Sertac and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, 2011.

[9] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. Overmars. Multi-level path planning for nonholonomic robots using semi-holonomic subsystems. *Int. J. Rob. Res.*, 17(8):840–857, 1998.

[10] R. Geraerts and M. Overmars. Creating high-quality paths for motion planning. *Int. J. Rob. Res.*, 26(8):845–863, 2007.

[11] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2493–2498, Anchorage (AK), 2010.

[12] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, and S. Srinivasa. CHOMP: covariant hamiltonian optimization for motion planning. *Int. J. Rob. Res.*, 32(9-10):1164–1193, 2013.

[13] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *Int. J. Rob. Res.*, 33(9):1251–1270, 2014.

[14] J.-P. Laumond, N. Mansard, and J.B. Lasserre. Optimality in robot motion: Optimal versus optimized motion. *Communications of the ACM*, 57(9):82–89, 2014.

[15] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4569–4574, Shanghai, China, 2011.

[16] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE J. Rob. Autom.*, 4(2):193–203, 1988.

[17] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *Int. J. Rob. Res.*, 21(12):1031–1052, 2002.

[18] R. Guernane and N. Achour. Generating optimized paths for motion planning. *Robotics and Autonomous Systems*, 59(10):789–800, 2011.

[19] J. Pan, L. Zhang, and D. Manocha. Collision-free and smooth trajectory computation in cluttered environments. *Int. J. Rob. Res.*, 31(10):1155–1175, 2012.

[20] P. A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, Princeton (NJ), 2008.

[21] F. Schwarzer, M. Saha, and J.C. Latombe. *Algorithmic Foundations of Robotics V*, chapter Exact Collision Checking of Robot Paths, pages 25–41. Springer Berlin, 2004.

[22] J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3859–3866, Saint Paul (MN), 2012.

[23] J. Nocedal and S. Wright. *Numerical Optimization*. Springer New York, 2nd edition, 2006.

[24] F. Lamiraux. Humanoid path planner. http://projects.laas.fr/gepetto/index.php/Software/Hpp, 2014.

[25] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Rob. J.*, 14(6):477–493, 2000.

[26] J. J. Kuffner and S. M. Lavalle. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 995–1001, San Francisco (CA), 2000.

[27] Robothon of Factory In A Day - Philips case. https://youtu.be/fhKlfVsupOE, 2015.