A simple path optimization method for motion planning

Mylène Campana, Florent Lamiraux, Jean-Paul Laumond

A simple path optimization method for motion planning

Mylène Campana[1,2], Florent Lamiraux[1,2] and Jean-Paul Laumond[1,2]

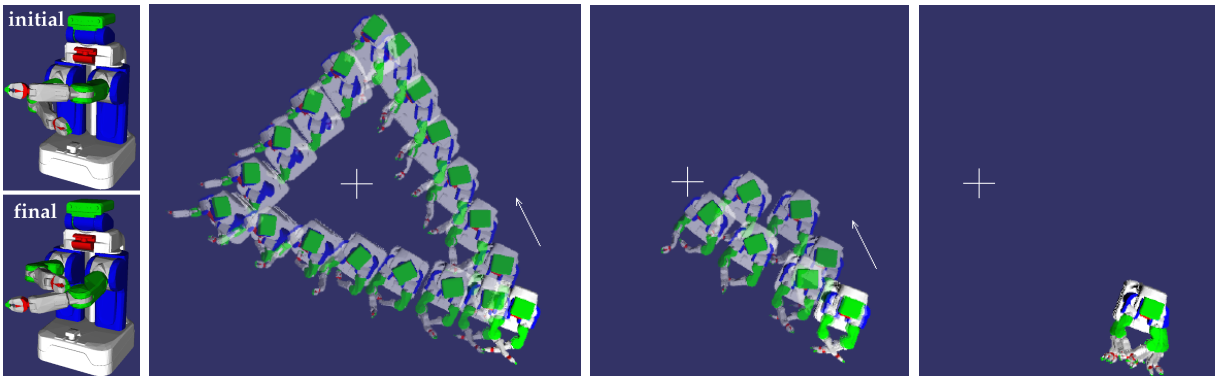

Fig. 1: In this motion planning problem, PR2 robot has just to exchange the positions of its arms. The task is simple, however, in absence of explicit indication, any probabilistic motion planner will compute a path that makes the PR2 mobile basis purposelessly moving. This is the case of RRT-connect algorithm (left). Path optimization is expected to remove unnecessary motions. The popular random shortcut algorithm fails in this case (middle). Our algorithm succeeds (right).

***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 gradient-based algorithm that transforms a polygonal collision-free path into a shorter one, while both:**

- **requiring mainly collision checking, and few time-consuming obstacle distance computation,**
- **constraining only part of the configuration variables that may cause a collision, and not the entire configurations, and**
- **reducing parasite motions that are not useful for the problem resolution.**

The algorithm is simple and requires few parameter tuning. Experimental results include navigation and manipulation tasks, e.g. an robotic arm manipulating throw a window and a PR2 robot working in a kitchen environment, and comparisons with a random shortcut optimizer.

I. INTRODUCTION AND PROBLEM STATEMENT

Motion planning for systems in cluttered environments has been addressed for more than thirty years [1], [2]. Most planners today randomly sample the system configuration space [3] in order to find a collision-free path. 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 [4], [5] 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 [6] 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 [7] are also close to an optimal solution, but computation time is significantly higher than classical approaches. Moreover the roadmap is only valid for a given motion planning problem.

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 methods are still very popular [8], [9]. However, random shortcut requires fine tuning of the termination condition (see Algorithm 2) and is no efficient due to randomness for long trajectories where only a small part needs to be optimized. Figure 2 presents another situation where random shortcut will always fail to optimize the initial path, since it cannot decouple the robot degrees of freedom (DOF) on which the optimization occurs, contrary to our algorithm.

*This work has been supported by the project ERC Advanced Grant 340050 Actanthrope and by the FP 7 project Factory in a Day under grant agreement n 609206

[1]{mcampana, florent, jpl}@laas.fr
CNRS, LAAS, 7 avenue du colonel Roche, F–31400 Toulouse, France
[2]Univ. de Toulouse, LAAS, F–31400 Toulouse, France

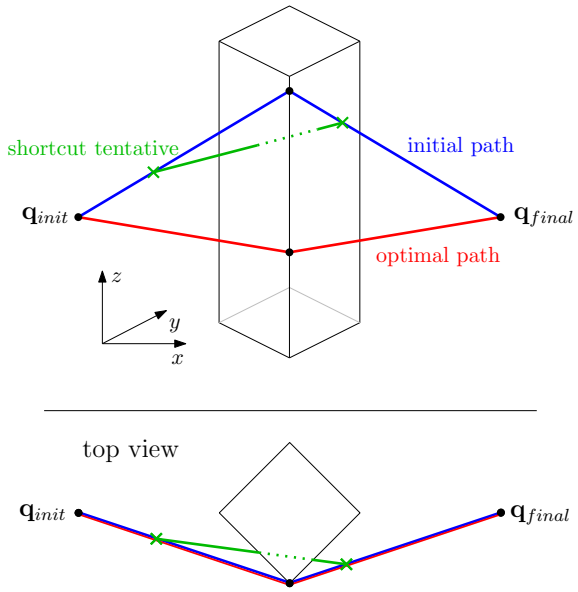

Fig. 2: Example of a path which random shortcut will never manage to optimize: each shortcut tentative will provide a collision or will not decrease the path length. The optimal path belongs to the $x-y$ plane containing q_{init} and q_{end}.

On the other hand, numerical optimization methods like CHOMP [10] 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 along the trajectory. These methods therefore require a pre-processing step of the robot (and/or environment) model in order to make it simpler: [10] covers PR2 bodies with spheres, while [11] needs to decompose objects into convex subsets. Moreover, in the applications we address, optimality is not desirable as such since the shortest path between two configurations in a cluttered environment usually contains contacts with the obstacle, or is most of the time out of the scope of numerical optimization algorithms [12].

The idea of our method is to find a good trade-off between the simplicity of blind methods like shortcut algorithm, 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 linearized transformation constraint between the objects detected in collision. Only few distances between objects are evaluated, therefore no pre-processing of the robot or environment models is necessary. The underlying optimization algorithm is a quadratic program.

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.

II. RELATED WORK

CHOMP algorithm [10] 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 [13] avoids to compute 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 Euclidian 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 [14].

Some optimization-based planners may not require an initial guess but some naive straight-line manually or randomly-sampled initialization as TrajOp [15]. Its trajectory 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 [16] 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 that returns C^1 shortcuts made of parabolas and lines (bang-bang control) [9]. These local refined trajectories are time-optimal since they comply with acceleration and velocity constraints.

In some way, our method shares similarities with [17] since this latter method 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, the iterations are composed of Cubic-B-splines. The benefit of this method is that the result is a differentiable path.

III. PATH OPTIMIZATION

A. 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, $1 \leq i \leq m$ 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 shown in this paper, we consider 4 types of joints described in Table I. 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 submanifold of $\mathbb{R}^n$.

The velocity of each joint J_i, $1 \leq i \leq m$ 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.

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

a) 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. We do not have enough space in this paper to develop the theory in a more rigorous way. The reader can easily state that "following a constant velocity" makes sense for the four types of joints defined in Table I. We refer to [18] Chapter 5 for details about Riemanian geometry.

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 I: Translation and rotation joint position are defined by 1 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: 2 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 unbounded 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 $\omega \in \mathbb{R}^3$.

B. 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 $SO(3)$.

C. Problem definition

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:

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

1) 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 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 rotations, the weight is equal to the maximal distance of the robot bodies moved by the joint to the center of the joint.

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

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

where the λ_{i-1} coefficients benefit will be explained in the results section. For now we can assume that $\forall i, \lambda_{i-1} = 1$.

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.

D. Resolution

We assume that the direct interpolation between the initial and final configurations contains collisions. Let H denote the constant Hessian of the cost function, an iteration is described as follow:

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

where 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 q_0 and q_{n+1}. Since this solution is in collision, we set $\alpha_i = \alpha$ where α is a parameter that will be explained in the algorithm section.

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. We use a continuous collision checker inspired of [19] to validate our paths and to return the first colliding configuration along a path. This step corresponds to `validatePath` in Algorithm 1.

E. Constraints

Let T be a positive real number such that each path $\mathbf{x}_i$ is a mapping from interval $[0,T]$ into $\mathscr{C}$: $\mathbf{x}_i(0) = q_0$, $\mathbf{x}_i(T) = q_{n+1}$ for all i. Let us denote by $t_{coll\ i}$ the abscissa of the first collision detected on path $\mathbf{x}_{i+1}$, which previous iteration $\mathbf{x}_i$ was collision-free (see Figure 3). Thus in configuration $\mathbf{x}_{i+1}(t_{coll\ i})$ 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, we will only consider the first case. Reasoning about the second case is similar, except that the constraint is on the transformation of $\mathscr{B}_1$ with respect to the environment.

Let $M^* \in SE(3)$ denote the homogeneous transformation between frames of $\mathscr{B}_1$ and $\mathscr{B}_2$ in configuration $\mathbf{x}_i(t_{coll\ i})$. Let $M \in SE(3)$ denote the same transformation in configuration $\mathbf{x}(t_{coll\ i})$ where $\mathbf{x}$ is any path in $\mathscr{C}^{wp}$. Let $R \in SO(3)$ (resp. R^*) and $\mathbf{t} \in \mathbb{R}^3$ (resp. $\mathbf{t}^*$) be the rotation and translation parts of M (resp. M^*). See also Figure 4 for notations.

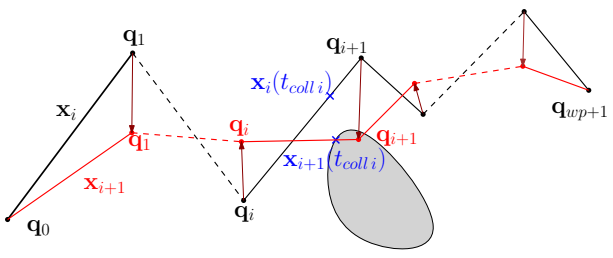

Fig. 3: Illustration of one iteration of our optimization. $\mathbf{x}_{i+1}$ appears to be in collision with the obstacle, the first colliding configuration $\mathbf{x}_{i+1}(t_{coll\ i})$ is returned by the continuous collision checker. The corresponding constraint will be computed in configuration $\mathbf{x}_i(t_{coll\ i})$.

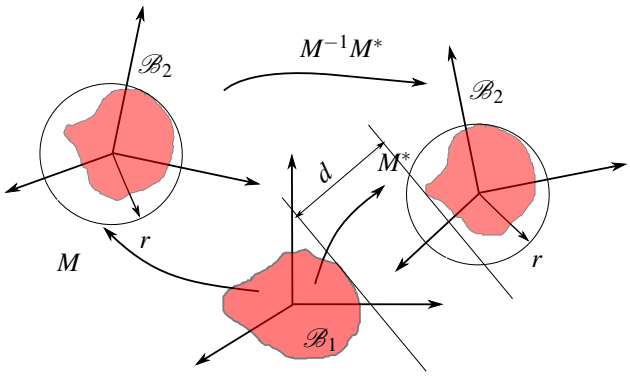

Fig. 4: Transformation constraint notations.

We denote by F the mapping from $\mathscr{C}^{wp}$ to $\mathbb{R}^6$ defined by

$$F(\mathbf{x}) = \begin{pmatrix} R^T(\mathbf{t}^* - \mathbf{t}) \\ \log(R^T R^*) \end{pmatrix} \tag{2}$$

$\log(R^T R^*)$ represents here a vector $\mathbf{v}$ of $\mathbb{R}^3$ such that $R^T R^*$ is the rotation of axis $\mathbf{v}$ and of angle $\|\mathbf{v}\|$. Note that F vanishes for any path $\mathbf{x}$ such that in configuration $\mathbf{x}(t_{coll\ i})$ the relative position of $\mathscr{B}_2$ w.r.t. $\mathscr{B}_1$ is equal to M^*. Let r denote the *radius* of the bounding sphere of $\mathscr{B}_2$ centered at the origin of $\mathscr{B}_2$ local frame. Let us denote by d the minimal distance between $\mathscr{B}_1$ and $\mathscr{B}_2$ in configuration $\mathbf{x}_i(t_{coll\ i})$. We have the following property.

Property 1: Let $\mathbf{x} \in \mathscr{C}^{wp}$ be a path and let us denote by $\mathbf{t}$ and $\mathbf{v}$ respectively the 3 first coordinates and the 3 last coordinates of $F(\mathbf{x})$. If

$$\|\mathbf{t}\| + r\|\mathbf{v}\| \leq d$$

then in configuration $\mathbf{x}(t_{coll\ i})$, $\mathscr{B}_1$ and $\mathscr{B}_2$ are collision-free. The proof is straightforward: expressed in reference frame of $\mathscr{B}_1$, the motion of $\mathscr{B}_2$ between configurations $\mathbf{x}(t_{coll\ i})$ and $\mathbf{x}_i(t_{coll\ i})$ is a translation of vector $\mathbf{t}$ followed by a rotation of angle $\|\mathbf{v}\|$. No point of $\mathscr{B}_2$ has moved by more than $\|\mathbf{t}\| + r\|\mathbf{v}\|$. Therefore no collision is possible.

When a collision is detected at $t_{coll\ i}$ along path $\mathbf{x}_{i+1}$, we wish to add constraint

$$F(\mathbf{x}) = 0 \tag{3}$$

However, as this constraint is non-linear, we linearize it around $\mathbf{x}_i$:

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

for any later iteration $\mathbf{x}_j$, $j \geq i+1$.

We refer to [20] for solving constrained quadratic programs (QP). This corresponds to `computeIterate` in Algorithm 1.

F. Relinearization of constraints

As constraint (3) is not exactly enforced, but linearized, a new collision may appear at $t_{coll\ i}$ on a later iteration between $\mathscr{B}_1$ and $\mathscr{B}_2$. To avoid this undesired effect, we check the values of all the constraints at each iteration. The constraints

that fail to satisfy Property 1 are relinearized around the latest valid path $\mathbf{x}_k$, $k > i+1$, as follows:

$$\frac{\partial F}{\partial \mathbf{x}}(\mathbf{x}_k)(\mathbf{x}-\mathbf{x}_k) = -F(\mathbf{x}_k)$$

This step is performed by `solveConstraints` in Algorithm 1.

G. Algorithm

In this part we describe the path-optimizer Algorithm 1 according to the previous step definitions. The main difficulty here is to handle the scalar parameter α determining how much of the computed step $\mathbf{p}$ will be traveled through, and also to return a collision-free path. Typically, we chose $\alpha_{init} = 0.2$ to process small steps. $\alpha_{init} = 0.5$ is more encountered in the optimization literature, procuring larger steps but with more risks of collision.

Algorithm 1 Gradient-based path-optimization.

```
1:  procedure OPTIMIZE(x_0)
2:      α ← α_init; ε ← 10^-3
3:      minReached ← false
4:      validConstraints ← true
5:      while (not(noCollision and minReached)) do
6:          p = computeIterate()
7:          minReached = (||p|| < ε or α = 1)
8:          x_1 ← x_0 + α p
9:          if (α ≠ 1) then
10:             if (not(solveConstraints(x_1))) then
11:                 α ← α * 0.5
12:                 validConstraints ← false
13:             else
14:                 validConstraints ← true
15:         if (not(validatePath(x_1))) then
16:             noCollision ← false
17:             if (α ≠ 1) then
18:                 addCollisionConstraints()
19:                 α ← 1
20:             else
21:                 if (validConstraints) then
22:                     α ← α_init
23:         else
24:             x_0 ← x_1
25:             noCollision ← true
26:             α ← 0.5 * (1 + α)
        return x_0
27: end procedure
```

Each time a collision-constraint is added, the solution of the current QP is tested (i.e. $\alpha = 1$). If this path is collision-free, the algorithm returns it as the solution (lines 7-25). Otherwise, smaller steps are iteratively applied and tested toward the minimum (lines 21-22), and constraints are added each time a collision is found (line 18).

If the constraint relinearization fails to validate Property 1, α is halved and the algorithm restarts from the latest valid path (lines 10-12).

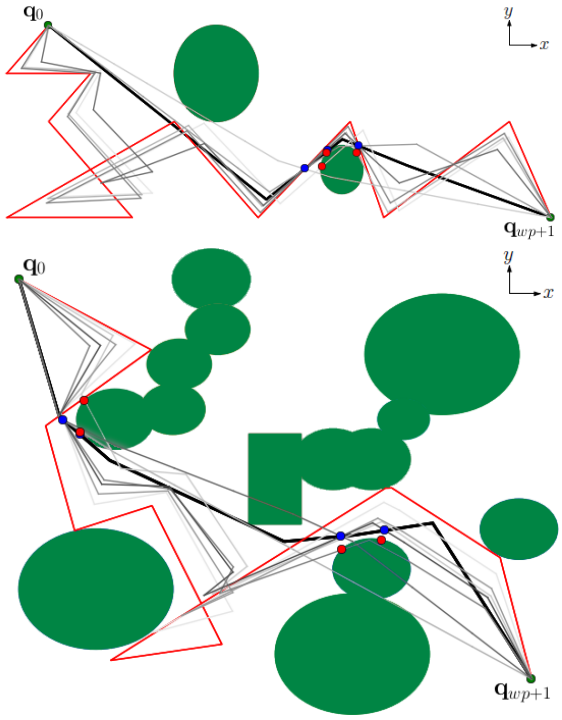

Fig. 5: Path-optimization results on a 2D-ponctual robot, moving around obstacles. Initial paths are in red, optimized ones in black. Grey paths represent intermediate iterations, red dots colliding configurations and blue dots the associated collision-free configurations of the backtracked paths where constraints have been computed.

IV. RESULTS

This part gathers optimization results performed on the planning software Humanoid Path Planner [21]. The initial trajectory is obtained with two kind of probabilistic planners: Visibility-PRM [22] and RRT-connect [23].

A. From 2D basic examples...

Figure 5 shows several iterations of our optimizer on 2D cases. Since in this special case, transformation constraints are equivalent to compulsory configurations to pass through, we can verify that our optimized path is applying all computed constraints. We can also understand which collisions have led to the constraints.

Besides, these examples give a better understanding of how the tuning of α has to balance lot of iterations and relevant collision-constraint addition. For example, if we alter a lot the initial path with a large gradient step and compute the corresponding collisions, constraints will be chosen on a very-not optimal path and will not be pertinent w.r.t. the obstacles we wanted to avoid.

Figure 5 illustrates a path example that random shortcut will not manage to optimize in an affordable time, because of probabilistically failing to sample configurations in the box. Our gradient-based algorithm succeeds to optimize the path contained in the box, with the following cost coefficients

$$\lambda_{i-1} = \frac{1}{\sqrt{(\mathbf{q}_{i0}-\mathbf{q}_{i-10})^T W^2 (\mathbf{q}_{i0}-\mathbf{q}_{i-10})}}$$

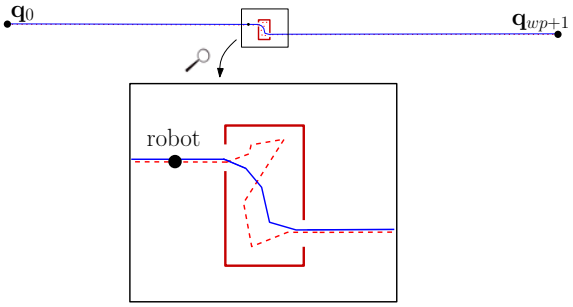

Fig. 6: Case of a long dashed initial path (above) containing a small part that can be optimized (below). Random shortcut is unlikely to optimize the part containing detours in the box, whereas our method succeeds (in blue).

aiming at keeping the same ratio between path segment lengths at minimum as at initial path, represented by the waypoints $(\mathbf{q}_{i0})_{1\leq i\leq n+1}$. Without these coefficients, the path that minimizes the cost corresponds to a straight line with the waypoints equidistantly allocated, which is not adapted for the Figure 6 type of problems with a local passage very constrained by obstacles. Note that this cost is also working with all other examples presented in this paper.

B. To 3D complex problems

We also experiment our algorithm on more complex robots, with transformation collision-constraints. Unless another value is given, α_{init} is set to 0.2.

In the included video, we present four situations where our algorithm has been tested and compared to random shortcut [24] (Algorithm 2). The termination condition of random shortcut allows it to try shortening the path until 5 iterations of non-improvement are reached (corresponding to *maxNbFailures* in Algorithm 2).

On the 5-DOF double-arm problem, one arm has to get around an obstacle while the other is relatively far from obstacles. As expected, the initial path given by RRT-connect activates both arms to solve the problem. Contrary to random shortcut, our optimizer manages to cancel the rotation of the free-arm while optimizing the first arm motion. Initial path length 5.17 has been decreased to 4.55 by random shortcut and to 3.20 by the gradient-based optimizer.

For the following example with a 6-axis manipulator arm in a cluttered environment, path reduction only manages to reduce some but not all detours. This behavior can be explained by a path over-constraining due to the important proximity of the robot to the obstacles during the whole path, and because there are not so many DOF to play on to easily avoid collisions and return a small-cost solution. We advance this explanation because on the two following high-DOF examples of the video (and Figure 1), results are better in terms of path length and quality.

In the example presented Figure 1 and in the video, the mobile 40-DOF PR2 simply has to cross its arms from the left arm up position to the right arm up one. Lengths of the initial path, the random shortcut optimized path and the gradient-based optimized path are respectively 14.5, 6.34 and 3.62. The RRT-connect planner returns detours and activates non-useful DOF such as the head, the torso lift and the translation on the ground. Once again, random shortcut will hardly optimize the mobile base translation of the robot and other unnecessary DOF uses, unlike our optimized-path which mainly results in moving the arms as expected (Figure 1 right).

We obtain similar results on the PR2 performing a manipulation task in a kitchen environment (see the video). The robot has to move his hands from the top to the bottom of a table. Our optimizer manages to reduce the initial length 15.8 from the visibility-PRM planner and improves the path quality just adding transformation constraints between the table and the robot's arms and grippers. Thus, the robot just slightly moves backward and uses its arm DOF to avoid the table, instead of processing a large motion to get away from the table. With our method, the path length is downed to 4.59, against 8.24 for random shortcut.

Optimization computation time and path length averages are presented for 30 runs of the PR2-crossing-arms in the Table II. In the same way, time computation averages for

Algorithm 2 Random shortcut as adapted from [8] Section 6.4.1. `steeringMethod` returns the linear interpolation between two configurations. $\mathbf{x}_{|I}$ denotes path $\mathbf{x}$ restricted to interval I. *maxNbFailures* is a parameter that affects time of computation and quality of the result.

procedure RANDOMSHORTCUT($\mathbf{x}$)
  nbFailures $\leftarrow 0$
  while $nbFailures < maxNbFailures$ **do**
    $failure \leftarrow true$
    $T \leftarrow$ upper bound of $\mathbf{x}$ definition interval
    $t_1 < t_2 \leftarrow$ random numbers in $[0,T]$
    $lp0 \leftarrow$ `steeringMethod`$(\mathbf{x}(0),\mathbf{x}(t_1))$
    $lp1 \leftarrow$ `steeringMethod`$(\mathbf{x}(t_1),\mathbf{x}(t_2))$
    $lp2 \leftarrow$ `steeringMethod`$(\mathbf{x}(t_2),\mathbf{x}(T))$
    $newPath \leftarrow$ empty path defined on $[0,0]$
    if $lp0$ is collision-free **then**
      $newPath \leftarrow lp0$; $failure \leftarrow false$
    else
      $newPath \leftarrow \mathbf{x}_{|[0,t_1]}$
    if $lp1$ is collision-free **then**
      $newPath \leftarrow$ `concatenate`$(newPath, lp1)$
      $failure \leftarrow false$
    else
      $newPath \leftarrow$ `concatenate`$(newPath, \mathbf{x}_{|[t_1,t_2]})$
    if $lp2$ is collision-free **then**
      $newPath \leftarrow$ `concatenate`$(newPath, lp2)$
      $failure \leftarrow false$
    else
      $newPath \leftarrow$ `concatenate`$(newPath, \mathbf{x}_{|[t_2,T]})$
    $\mathbf{x} \leftarrow newPath$
    if $failure$ **then** $nbFailures \leftarrow nbFailures + 1$
  return $\mathbf{x}$
end procedure

	Computation time	Path length	Traveled distance by the base
Initial		13.7	10.9
Gradient-based	781ms	3.69	10^{-6}
Random shortcut	823ms	3.82	2.94

TABLE II: Results for 30 runs of the PR2-crossing-arms example. For each run, a solution path is planned by Visibility-PRM as initial guess for each optimizer. α_{init} is set to 0.2. The right column represents the distance that is traveled by the robot mobile base during the path.

the PR2-in-kitchen example have been calculated: 59.5 s (gradient-based optimizer) and 76.2 s (random shortcut). Thus our method presents similar computation times and path lengths for mobile manipulation tasks. However, the extinction of the mobile base motion in the gradient-based optimized path is advanced in the right column of Table II. Thus the problem addressed Figure 2 is solved considering this DOF.

V. CONCLUSIONS

We managed to settle a path optimization for decoupled-DOF robots such as mobile manipulators. Our algorithm uses standard numerical tools as collision checking and QP resolution, and correlates them in a simple but effective way, playing on the scalar iteration parameter. Therefore, our method only require few distances computation, so geometry pre-processing or offline optimization are not necessary to remain time competitive. We demonstrate that the optimizer is time-competitive comparing to random shortcut and proposes better quality paths for high-DOF robots, removing unnecessary DOF motions. Our optimizer also manages to reduce a local detour in a long path while random shortcut methods will mostly fail.

The experimental results show that transformation constraints may in some cases be overconstraining. We also tried distance constraints without much success due to the non-continuous differentiability of distance. In a future work, we will study other one-dimensional constraints in order to achieve better results.

REFERENCES

[1] M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Pérez, and M. T. Masson, *Robot Motion: Planning and Control*, 1983.

[2] J. Schwartz and M. Sharir, "On the piano movers problem ii: General techniques for computing topological properties of real algebraic manifolds," *Advances of Applied Mathematics*, vol. 4, no. 3, pp. 298–351, 1983.

[3] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. 32, no. 2, pp. 108–120, 1983.

[4] C. Park, J. Pan, and D. Manocha, "ITOMP: incremental trajectory optimization for real-time replanning in dynamic environments," in *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil*, 2012.

[5] M. Garber and M. Lin, "Constraint-based motion planning using voronoi diagrams." in *Algorithmic Foundations of Robotics V, volume 7 of Springer Tracts in Advanced Robotics*, 2004, pp. 541–558.

[6] J. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed. Society for Industrial and Applied Mathematics, 2010.

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

[8] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. Overmars, "Multi-level path planning for nonholonomic robots using semi-holonomic subsystems," *International Journal of Robotics Research*, vol. 17, no. 8, pp. 840–857, 1998.

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

[10] N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[11] 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," *The International Journal of Robotics Research*, vol. 28, no. 5, 2014.

[12] J.-P. Laumond, N. Mansard, and J. Lasserre, "Optimality in robot motion: Optimal versus optimized motion," *Communications of the ACM*, vol. 57, no. 9, pp. 82–89, 2014.

[13] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 4569–4574.

[14] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[15] 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," *International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[16] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[17] J. Pan, L. Zhang, and D. Manocha, "Collision-free and smooth trajectory computation in cluttered environments," *International Journal of Robotics Research*, vol. 31, no. 10, 2012.

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

[19] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact collision checking of robot paths," in *Algorithmic Foundations of Robotics V*, 2004, pp. 25–41.

[20] J. Nocedal and S. Wright, *Numerical Optimization, Second Edition*. Springer New York, 2006.

[21] F. Lamiraux and J. Mirabel, "Humanoid path planner," http://projects.laas.fr/gepetto/index.php/Software/Hpp.

[22] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility based probabilistic roadmaps for motion planning," *Advanced Robotics Journal*, vol. 14, no. 6, 2000.

[23] J. J. Kuffner and S. M. Lavalle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, IEEE International Conference on*, vol. 2, 2000, pp. 995–1001.

[24] F. Lamiraux, "Random shortcut code in hpp," https://github.com/humanoid-path-planner/hpp-core/blob/df6d3ffb89555faa254bb42145ff398ed9d8a0c2/src/random-shortcut.cc#L62.