Optimizing paths in manipulation planning

Joseph Mirabel, Florent Lamiraux

Optimizing paths in manipulation planning

Joseph Mirabel
LAAS
University of Toulouse, CNRS
Toulouse, France
Email: joseph.mirabel@laas.fr

Florent Lamiraux
LAAS
University of Toulouse, CNRS
Toulouse, France
Email: florent.lamiraux@laas.fr

***Abstract*—We propose a path optimization method for manipulation. The method tries to minimize path length in the configuration space of the whole system (robots and objects). The problem is cast into a quadratic program with linear constraints. The current path is shorten iteratively. When a collision is detected along an iteration, the method backtracks to the previous collision-free path, adds a linear constraint that prevents the bodies in collision to come close to each other and starts again. The algorithm stops when the global minimum under the collision constraints is reached. The method is run on three different benchmarks, using the real geometric models of the robots Baxter, UR-3 and PR-2.**

I. Introduction

Manipulation planning is an instance of path planning where objects are moved by robot grippers. This implies a lot of constraints on the motion of the system composed of the robots and of the objects. Namely, when an object is not grasped, it should remain still in a stable placement, while when an object is grasped by a gripper, it should move in a rigid manner with the gripper.

This problem has given rise to a lot of interest for the past forty years. Pioneering works by [23] and [1] first considered low dimensional problems where robots and objects move in translation. [20] is the first work that applies random motion planning methods to the problem of manipulation planning. Since then, random methods have been applied extensively to the manipulation planning problem [21, 17, 4, 12, 18, 13, 6, 8, 5, 2, 10, 9]. Alternative methods based on optimal control have also been proposed [22].

Despite their efficiency and the fact that they are easy to implement, random sampling methods suffer a big drawback: the resulting path is usually far from optimal and includes useless detours. Asymptotic optimal versions of random methods exist [19], but optimality comes at a cost of much longer search time.

Numerical optimization methods like STOMP [11] or CHOMP [24] have been proposed for the classical path planning problem without manipulation. The main problem in casting path optimization into a numerical constrained optimization problem is the collision avoidance part. Most methods sample the initial trajectory and add inequality constraints on the distance between each pair of body that can potentially collide with each other. As a result, the optimization problem includes thousands of inequality constraints and is intractable without pre-processing the robot and/or environment models. STOMP and CHOMP for instance both precompute a geometrical approximation of the robot using spheres. [14] proposes a trajectory optimization method with continuous collision detection. This method performs better than STOMP and CHOMP. This method relies on distance computation where we only rely on collision detection, the latter being cheaper to computer, easier to implement and more robust in practice. Moreover, none of those methods have been extended to manipulation motions. A sequential quadratic programming approach is proposed in [7]. Their method optimizes the parameters of a task plan (object intermediate position, robot trajectories...). However, the method is a preliminary work and has only been used on circles moving in a two dimensional plane.

A few years ago, [3] proposed a simple path optimization method that handles collision checking in a most efficient way. The idea is to express the path optimization problem as a quadratic program (QP), to iterate over shorter and shorter paths and to introduce a linearized constraint on the distance between to bodies only when a collision between those bodies is detected along an iteration. Each time a collision is detected, the search resumes at the latest collision-free iteration.

This paper proposes an extension of the latter method to manipulation paths. The initial method optimizes path length for robots without constraints. The extension described in this paper extracts the free variables of the set of waypoints of the initial trajectory and optimizes over those variables as described in Section IV.

II. Definition of the problem

In this section we recall the principle of the method described in [3]. We denote by $\mathcal{C} \subset \mathbb{R}^n$ the configuration space of the robot. A path resulting from a random sampling algorithm is usually a concatenation of linear interpolations between waypoints: $(\mathbf{q}_0, \mathbf{q}_1, \cdots, \mathbf{q}_{N+1})$ where N is the number of intermediate waypoints. $\mathbf{q}_0$ and $\mathbf{q}_{N+1}$ are the initial and final configurations of the path. They should therefore not be modified by the optimization method. The problem we want to solve is to find a sequence of new values for $(\mathbf{q}_1, \cdots, \mathbf{q}_N)$ such that the resulting concatenation of linear interpolations is collision-free and shorter than the initial one.

We denote by $\mathbf{x}$ the optimization variables.

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

a) Parameterization: for simplicity, we denote also by $\mathbf{x}$ the mapping from interval $[0, N+1]$ to the free configuration space, such that for any $j \in \{0, \cdots, N\}$, for any $\kappa \in [j, j+1]$,

$$\mathbf{x}(\kappa) = (\kappa - j)\,\mathbf{q}_{j+1} + (1 - (\kappa - j))\,\mathbf{q}_j \tag{1}$$

b) Cost: Let $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix of positive weights. We define the distance between two configurations $\mathbf{q}_2$ and $\mathbf{q}_1$ 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 make rotation and translation variables homogeneous, and to give more importance to rotation degrees of freedom that move points farther to their rotation axis. Given $\mathbf{q}_0$ and $\mathbf{q}_{N+1}$ fixed, the cost we want to minimize is defined by

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

where λ_k are constant weights associated to each linear interpolation. The role of those weights will be explained in the next section.

Let us notice that the cost is not exactly the length of path $\mathbf{x}$ for norm $\|.\|_W$, but it can be established that paths minimizing the above cost also minimize length.

An important point is that the cost is a quadratic function of the optimization variables $\mathbf{x}$. We denote by H the constant Hessian matrix of the cost.

In the next Section, we recall how the previous method described in [3] works.

III. Path optimization: the classical case

Input: the initial path $\mathbf{x}_0$
Input: integer $m \geq 2$
1 $\mathcal{L} \leftarrow \emptyset$
2 **for** $i \leftarrow 0$ **to** ∞ **do**
3 $\mathbf{x}_i^* \leftarrow$ `Optimum`$(\mathbf{x}_i, \mathcal{L})$
4 **if** `CollisionFree`$(\mathbf{x}_i^*)$ **then return** $\mathbf{x}_i^*$
5 $\mathbf{y}_0 \leftarrow \mathbf{x}_i$; $l \leftarrow 0$
6 **repeat**
7 $\mathbf{y}_{l+1} \leftarrow \mathbf{y}_l + \frac{\mathbf{x}_i^* - \mathbf{x}_i}{m}$
8 $l \leftarrow l + 1$
9 **until** *not* `CollisionFree`$(\mathbf{y}_{l+1})$ *and* $l < m$
10 **if** `CollisionFree`$(\mathbf{y}_{l+1})$ **then return** $\mathbf{y}_{l+1}$
11 $\mathcal{L} \leftarrow \mathcal{L} \cup$ `NewConstraint`$(\mathbf{y}_l, \mathbf{y}_{l+1})$
12 $\mathbf{x}_{i+1} \leftarrow \mathbf{y}_l$
13 **end**

Algorithm 1: OptimizePath

For any non negative integer i, we denote by

$$\mathbf{x}_i \triangleq (\mathbf{q}_{1,i}, \cdots, \mathbf{q}_{N,i}) \tag{3}$$

the values of the waypoints at iteration i. Algorithm 1 takes as input the initial path $\mathbf{x}_0$ and a parameter m bigger than 2.

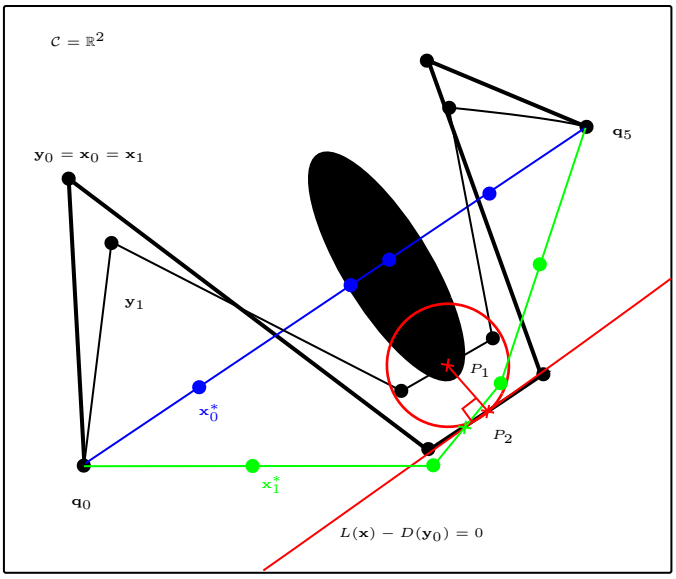

Fig. 1. The path optimization method as described in Algorithm 1 for a point moving in the plane. The initial path $\mathbf{x}_0$ is in bold black. The global optimum $\mathbf{x}_0^*$ is the straight line in blue. $\mathbf{y}_1$ (thin black) obtained by Equation (4) is in collision on the third linear interpolation at parameter $\kappa = 2.46$. $\mathbf{x}_1$ is set to $\mathbf{y}_0 = \mathbf{x}_0$ and a linear constraint is added: namely $P_2 = \mathbf{x}(\kappa)$ should stay on the red line. $\mathbf{x}_1^*$ is the global minimum with the constraint. It is collision-free: this is the result of the algorithm (Line 4).

As stated in the previous section, the cost (2) is quadratic with respect to the optimization variables. At the optimum, the waypoints are aligned along the line segment $[\mathbf{q}_0, \mathbf{q}_N]$. To improve efficiency, we choose the λ_k in such a way that at optimum, the distance between waypoints is proportional to their distance in the initial path. This avoids the undesirable effect of the waypoints tending to be at equal distance to each other. However, this is optional and for simplicity, the λ_k can be thought of as being equal to 1. We denote by $\mathbf{x}_0^*$ this minimal cost path.

At each iteration i, the optimum $\mathbf{x}_i^*$ under the current constraints is computed. If $\mathbf{x}_i^*$ is collision-free, the algorithm returns $\mathbf{x}_i^*$. Otherwise, a sequence $\mathbf{y}_l$, starting with $\mathbf{y}_0 \leftarrow \mathbf{x}_i$, is computed by moving from the current value to $\mathbf{x}_i^*$.

$$\mathbf{y}_{l+1} \leftarrow \mathbf{y}_l + \frac{\mathbf{x}_i^* - \mathbf{x}_i}{m} \tag{4}$$

Equation (4) is applied until $\mathbf{y}_{l+1}$ is in collision. Then, $\mathbf{x}_{i+1} \leftarrow \mathbf{y}_l$ and the method adds a linear constraint as described in the next section.

A. Linear constraint

If $\mathbf{y}_{l+1}$ as computed above is in collision, we denote by

1) κ the parameter along the path where a collision has been detected,
2) $\mathcal{B}_1$ and $\mathcal{B}_2$ the bodies of the robot that are in collision,
3) P_1 and P_2 two points on $\mathcal{B}_1$ and $\mathcal{B}_2$ respectively that coincide when the robot is in configuration $\mathbf{y}_{l+1}(\kappa)$,
4) $P_1(\mathbf{q})$, $P_2(\mathbf{q})$ the position of P_1, respectively P_2 when the robot is in configuration $\mathbf{q}$.

Note that $\mathcal{B}_2$ is the environment if the collision occurred between a body of the robot and the environment. Given the above definitions, we define d as the mapping from $\mathcal{C}$ to $\mathbb{R}$ that maps to any value of $\mathbf{q}$ the half squared distance between P_1 and P_2 when the robot is in configuration $\mathbf{q}$:

$$d(\mathbf{q}) = \frac{1}{2}\|P_2(\mathbf{q}) - P_1(\mathbf{q})\|^2 \tag{5}$$

We define D as the corresponding mapping defined on $\mathcal{C}^N$:

$$D(\mathbf{x}) = d(\mathbf{x}(\kappa))$$

We deduce immediately from this definition that

$$D(\mathbf{y}_l) > 0 \qquad\qquad D(\mathbf{y}_{l+1}) = 0$$

We define L as the linearization of D at $\mathbf{y}_l$:

$$L(\mathbf{x}) = D(\mathbf{y}_l) + \frac{\partial D}{\partial \mathbf{x}}(\mathbf{y}_l)(\mathbf{x} - \mathbf{y}_l).$$

From Definition 1, we notice that D only depends on two waypoints $\mathbf{q}_j$ and $\mathbf{q}_{j+1}$. $\frac{\partial D}{\partial \mathbf{x}}(\mathbf{y}_l)$ is a row vector of the form:

$$\frac{\partial D}{\partial \mathbf{x}}(\mathbf{y}_l) = \left(0 \cdots 0 \ (1-\beta)\frac{\partial d}{\partial \mathbf{q}}(\mathbf{y}_l(\kappa)) \ \beta\frac{\partial d}{\partial \mathbf{q}}(\mathbf{y}_l(\kappa)) \ 0 \cdots 0\right)$$

where $\beta = (\kappa - j)$. Note that in [3] the constraint is expressed differently, but the linearization yields the same linear constraint.

Finally, the following constraint is added.

$$L(\mathbf{x}) - D(\mathbf{y}_l) = 0 \tag{6}$$

Figure 1 illustrates the algorithm on a simple case. Note that in this simple case, the workspace and configuration spaces are the same. Therefore P_2 is the point robot.

In the next section, we explain how to extend this method to paths with constraints.

IV. Extension to manipulation

In manipulation planning, the configuration space $\mathcal{C} \subset \mathbb{R}^n$ of the system is the Cartesian product of the configuration spaces of the robots and objects. A manipulation path can be decomposed into a sequence of paths such that along each elementary path, a given constraint applies to the configuration of the system.

a) Example: let us consider the simple case of a manipulator arm that manipulates one object (Figure 2). The configuration of the system is denoted by $(\mathbf{q}_{rob}, \mathbf{q}_{obj})$ where $\mathbf{q}_{rob}$ is the configuration of the robot and $\mathbf{q}_{obj} \in SE(3)$ is the configuration of the object. Any manipulation path is a concatenation of *transit* and *transfer* paths. The constraint related to *transit* is that the object should remain static in a stable pose. The constraint related to *transfer* is that the relative pose between the object and the gripper should remain constant along the motion. In both cases, the position of the object depends on the configuration of the robot. The optimization method described in the previous section can thus be applied to the paths of the robot along *transfer* and *transit* sections without modifying configurations that connect these sections. The path of the object is then deduced from the path of the robot.

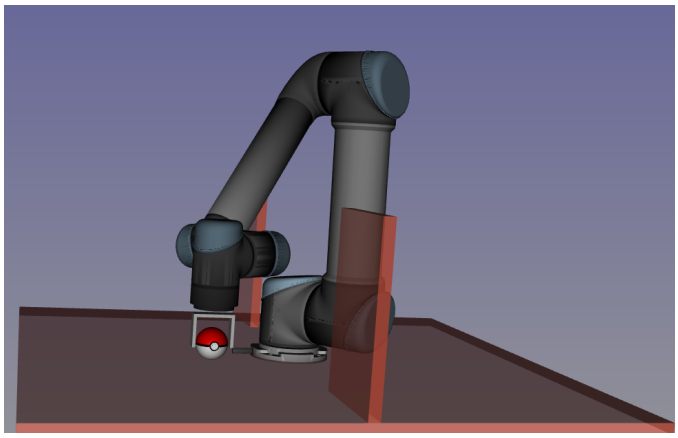

Fig. 2. Manipulator arm manipulating an object.

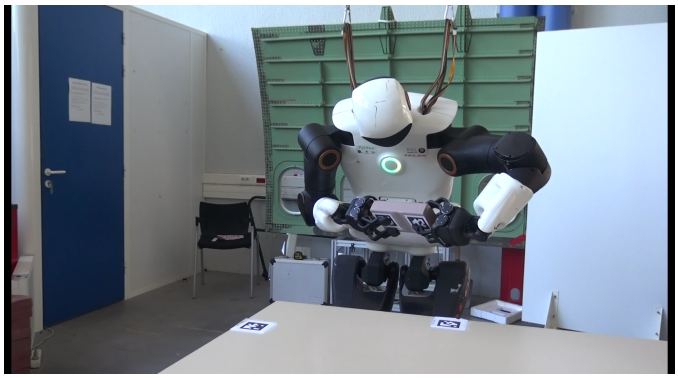

Fig. 3. A humanoid robot manipulating an object with two hands.

We now generalize the simple reasoning above to general cases with several objects and robots. Our approach is based on the framework described in [16]. This framework defines a constraint graph the states of which contain sets of constraints applying to the system. In the example of Figure 2 above, the states are *placement* and *grasp*. The edges of this graph are called *transitions* and contain additional constraints that apply to the system along paths. For instance, in state *placement*, the object should lie in a stable pose. Along transition *transit*, an additional constraint states that the object should not move.

The result of manipulation planning is thus a sequence of motions each one associated to a set of constraints. As stated above in a simple case, some constraints can be explicit: the object is in a given position with respect to the gripper, or the object is at a fixed position. Some constraints can be implicit, or some explicit constraints can be incompatible and need to be expressed implicitly as explained in the next section.

A. Implicit and explicit constraints

Let us consider another example displayed in Figure 3. With the same notation as in example of Figure 2, The constraints that apply to the system in the current state are the following.

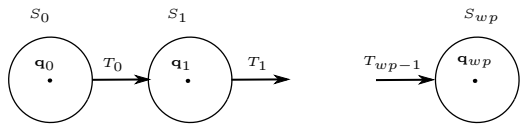

Fig. 4. A manipulation path is a sequence of elementary paths subject to a given set of constraints. Each waypoint $\mathbf{q}_j$ lies in a state S_j, $1 \leq j \leq N-1$.

- quasi-static equilibrium and fixed pose of the feet: $h(\mathbf{q}_{rob}, \mathbf{q}_{obj}) = 0$,
- object is in left hand: $\mathbf{q}_{obj} = f_l(\mathbf{q}_{rob})$
- object is in right hand: $\mathbf{q}_{obj} = f_r(\mathbf{q}_{rob})$

The two latter explicit constraints are clearly not compatible. However, the set of constraints can be solved by substituting $f_r(\mathbf{q}_{rob})$ for $\mathbf{q}_{obj}$:

$$\begin{aligned} h(\mathbf{q}_{rob}, f_r(\mathbf{q}_{rob})) &= 0 \\ f_r(\mathbf{q}_{rob}) - f_l(\mathbf{q}_{rob}) &= 0 \\ \mathbf{q}_{obj} &= f_r(\mathbf{q}_{rob}) \end{aligned}$$

[15] explains how to automatically solve this type of system composed of explicit and implicit constraints using Newton-Raphson algorithm.

B. Manipulation path

a) Notation and definitions: let $\mathbf{q} \in \mathbb{R}^n$ denote a configuration of the whole system. We denote by

- I a subset of $\{1, \cdots n\}$,
- $|I|$ the cardinal of I,
- $I(\mathbf{q}) \in \mathbb{R}^{|I|}$ the vector composed of the components of $\mathbf{q}$ of indices in I in increasing order.

For instance, if $I = \{2, 4, 5\}, |I| = 3, I(\mathbf{q}) = (\mathbf{q}_2, \mathbf{q}_4, \mathbf{q}_5) \in \mathbb{R}^3$.

b) State and transition constraints: for any j between 1 and N, waypoint $\mathbf{q}_j$ is subject to the constraints of state S_j and of transition T_{j-1}. Altogether, these constraints can be reduced to the following form as explained in [15]:

$$\exists\ out_j,\ in_j,\ free_j \quad \text{a partition of } \{1, \cdots, n\} \quad (7)$$

$$out_j(\mathbf{q}_j) = f_j(in_j(\mathbf{q}_j), free_j(\mathbf{q}_j)) \quad (8)$$

$$h_j(in_j(\mathbf{q}_j)) = 0, \quad (9)$$

- $out_j(\mathbf{q}_j)$ are the coordinates that are computed by explicit constraints,
- $in_j(\mathbf{q}_j)$, the input variables of the implicit and explicit constraints,
- $free_j(\mathbf{q}_j)$ the variables that are not constrained either by explicit or implicit constraints.

In example of Figure 2, if $\mathbf{q}_{j-1}$ and $\mathbf{q}_j$ are in *placement*, T_{j-1} is *transit*. We denote by rob and obj the subset of indices corresponding respectively to the robot and object in the configuration variable:

$$obj(\mathbf{q}) = \mathbf{q}_{obj} \quad rob(\mathbf{q}) = \mathbf{q}_{rob}$$

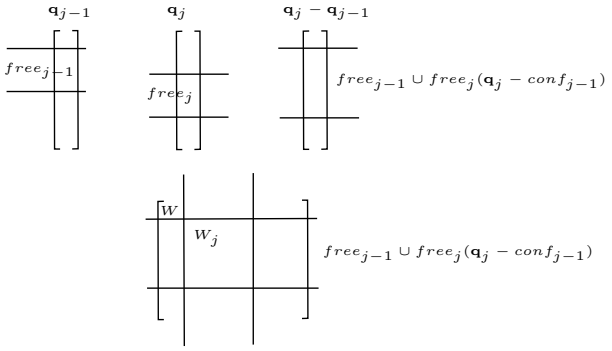

Fig. 5. Cost to be optimized as a quadratic function of the free waypoint coordinates. The part of the cost relative to the segment linking two waypoints is the distance between the sub-vectors of coordinates included in $free_j \cup free_j$ and weighed by the submatrix of W containing the rows and columns of indices in $free_j \cup free_j$.

With this notation,

$$\begin{aligned} out_j &= obj,\ \ in_j = \emptyset,\ \ free_j = rob \\ obj(\mathbf{q}_j) &= obj(\mathbf{q}_{j,0}). \end{aligned}$$

If $\mathbf{q}_{j-1}$ and $\mathbf{q}_j$ are in *grasp*, T_{j-1} is *transfer*. The above constraints are

$$\begin{aligned} out_j &= obj,\ \ in_j = \emptyset,\ \ free_j = rob \\ obj(\mathbf{q}_j) &= f_{grasp}(rob(\mathbf{q}_j)). \end{aligned}$$

C. Optimization of manipulation paths

In equations (7-9), $free_j(\mathbf{q}_j)$ are the free variables in the sense that they can be modified freely without affecting implicit constraint (9). Only the output variables $out_j(\mathbf{q}_j)$ are modified.

The extension of the optimization method described in Section III consists in restricting optimization to the free variables of all waypoints. (3) thus becomes:

$$\mathbf{x}_i = (free_1(\mathbf{q}_{1,i}), \cdots, free_N(\mathbf{q}_{N,i}))$$

a) Cost: to remain quadratic, the computation of the cost is restricted to the free variables. However the free variables of two successive waypoints may not be the same:

$$C(\mathbf{x}) \triangleq \frac{1}{2} \sum_{j=1}^{N+1} \lambda_{j-1} \| free_{j-1} \cup free_j(\mathbf{q}_j - \mathbf{q}_{j-1}) \|^2_{W_j} \quad (10)$$

where W_j is the symmetric positive definite matrix composed of the rows and columns of W of indices in $free_{j-1} \cup free_j$. See Figure 5.

b) Linear constraints: the path optimization algorithm is the same as in the classical case. The difference resides in the expression of the linear constraints. If at iteration $i+1$ (4) a collision occurs at parameter κ between waypoints $\mathbf{q}_j$ and $\mathbf{q}_{j+1}$, a linear constraint is added as in Section III-A. The only difference between Section III-A and the manipulation case is the dependency of some components of waypoints $\mathbf{q}_j$

and $\mathbf{q}_{j+1}$ with respect to the free variables $free_j(\mathbf{q}_j)$ and $free_{j+1}(\mathbf{q}_{j+1})$. According to (8),

$$\begin{aligned} out_j(\mathbf{q}_j) &= f_j(in_j(\mathbf{q}_j), free_j(\mathbf{q}_j)), \\ out_{j+1}(\mathbf{q}_{j+1}) &= f_{j+1}(in_{j+1}(\mathbf{q}_{j+1}), free_{j+1}(\mathbf{q}_{j+1})), \end{aligned}$$

The gradient of the linear constraint is now:

$$\frac{\partial D}{\partial \mathbf{x}}(\mathbf{y}_l) = (0 \cdots 0 \ (1-\beta)M_j \ \beta M_{j+1} \ 0 \cdots 0),$$

with

$$\begin{aligned} M_j &= \frac{\partial d}{\partial \mathbf{q}}(\mathbf{x}(\kappa)) \frac{\partial \mathbf{q}_j}{\partial free_j(\mathbf{q}_j)}(free_j(\mathbf{q}_j)), \\ M_{j+1} &= \frac{\partial d}{\partial \mathbf{q}}(\mathbf{x}(\kappa)) \frac{\partial \mathbf{q}_{j+1}}{\partial free_{j+1}(\mathbf{q}_j)}(free_{j+1}(\mathbf{q}_{j+1})), \end{aligned}$$

where matrix $\frac{\partial \mathbf{q}_j}{\partial free_j(\mathbf{q}_j)}(free_j(\mathbf{q}_j))$ denotes the variation of the components of $\mathbf{q}_j$ with respect to the free components. This matrix of n rows and $|free_j|$ columns is of the following form:

$$\frac{\partial \mathbf{q}_j}{\partial free_j(\mathbf{q}_j)}(free_j(\mathbf{q}_j)) = \begin{array}{l} free_j\{ \\ out_j\{ \\ in_j\{ \end{array} \begin{pmatrix} I_{|free_j|} \\ \frac{\partial f_j}{\partial free_j(\mathbf{q}_j)} \\ \frac{\partial f_j}{\partial in_j(\mathbf{q}_j)} \end{pmatrix} \tag{11}$$

where f_j is defined by (8). In the above expression, the indices $free_j$, out_j and in_j are assumed to be consecutive and ordered. In the general case the lines of the matrix corresponding to these subsets may be interleaved.

D. Convergence analysis and refinement

Algorithm 1 solves a sequence of quadratic programs (QP) with linear constraints (lines 2–13). The cost is the same for all QP and one linear constraint is added at each iteration (line 11). Note that the inner loop (line 6) finishes at most after $m-1$ iterations.

In most cases, the new linear constraint does not belong to the vector space spanned by the previous linear constraints and the search space dimension decreases by 1. The algorithm thus terminates in a number of iterations that is at most the initial dimension of search space, that is the number of free parameters.

1) Refinement: to cope with the case when a new linear constraint belongs to the subspace spanned by the previous linear constraints, we replace line 11 of Algorithm 1 by Algorithm 2. This refinement looks for pairs of trajectories on line segment $[\mathbf{y}_l, \mathbf{y}_{l+1}]$ such that one of the trajectories is in collision, and the other one is not. As soon as a pair of trajectories yields a linear constraint that is not spanned by the previous ones, the refinement loops breaks and Algorithm 1 resumes with this new constraint. In practice, the **for** loop terminates at the first iteration since the new constraint being spanned by the previous ones is very unlikely. For security, we exit when the number of iterations reaches 3.

if `NewConstraint` $(\mathbf{y}_l, \mathbf{y}_{l+1}) \in$ *span*$(\mathcal{L})$ **then**
  $\mathbf{y}_{free} \leftarrow \mathbf{y}_l$; $\mathbf{y}_{coll} \leftarrow \mathbf{y}_{l+1}$;
  for $l \leftarrow 0$ **to** ∞ **do**
    $\mathbf{y} \leftarrow \frac{\mathbf{y}_{free}+\mathbf{y}_{coll}}{2}$;
    if `CollisionFree` $(\mathbf{y})$ **then** $\mathbf{y}_{free} \leftarrow \mathbf{y}$;
    else $\mathbf{y}_{coll} \leftarrow \mathbf{y}$;
    if `NewConstraint` $(\mathbf{y}_{free}, \mathbf{y}_{coll}) \notin$ *span*$(\mathcal{L})$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup$ `NewConstraint` $(\mathbf{y}_{free}, \mathbf{y}_{coll})$;
      break;
    if $l \geq 3$ **then return** $\mathbf{y}_{free}$;
  end
else
  $\mathcal{L} \leftarrow \mathcal{L} \cup$ `NewConstraint` $(\mathbf{y}_l, \mathbf{y}_{l+1})$

Algorithm 2: Algorithm refinement

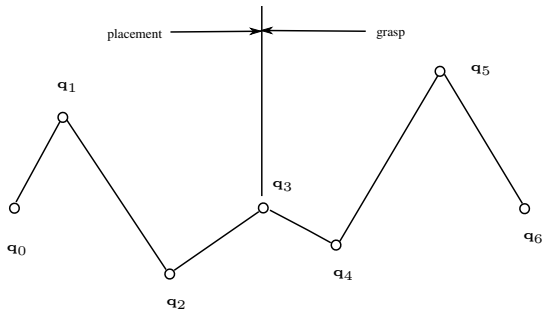

Fig. 6. Example of a manipulation path to be optimized. The system is composed of a robot and a movable object. $\dim \mathcal{C} = 12$. The 6 first components of the configuration represent the configuration of the robot. The 6 last components represent the pose of the object. The initial path is composed of $N = 5$ waypoints: $\mathbf{q}_1$, $\mathbf{q}_2$ in *placement*, $\mathbf{q}_4$, $\mathbf{q}_5$ in *grasp* and $\mathbf{q}_3$ in *grasp* $\cap$ *placement*.

V. Example

Let us consider a manipulation path for the example displayed in Figure 2. The configuration space of the system is of dimension 12. A manipulation path for this system is described in Figure 6. The input, output, and free variables for this path are the following:

$$\begin{array}{lll} free_1 = \{1, \cdots, 6\} & in_1 = \emptyset & out_1 = \{7, \cdots, 12\} \\ free_2 = \{1, \cdots, 6\} & in_2 = \emptyset & out_2 = \{7, \cdots, 12\} \\ free_3 = \emptyset & in_3 = \{1, \cdots, 6\} & out_3 = \{7, \cdots, 12\} \\ free_4 = \{1, \cdots, 6\} & in_4 = \emptyset & out_4 = \{7, \cdots, 12\} \\ free_5 = \{1, \cdots, 6\} & in_5 = \emptyset & out_5 = \{7, \cdots, 12\} \end{array}$$

Note that $\mathbf{q}_3$ has no free variable due to the closed chain constraint. The optimization algorithm will thus modify the robot configuration for waypoints $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$, and $\mathbf{q}_4$ in order to make the *transit* and *transfer* sub-paths shorter.

For $j = 4, 5$, the pose of the object $out_j(\mathbf{q}_j)$ depends on the configuration of the robot:

$$out_j(\mathbf{q}_j) = f_{grasp}\left(free_j(\mathbf{q}_j)\right)$$

Note that along the *transfer* part of the path (between $\mathbf{q}_3$ and $\mathbf{q}_6$), the position of the object depends on the position of the robot. The rows labeled out_j of matrix (11) correspond to the variation of the object pose with respect to the variation of the robot configuration.

VI. Splines

Up to now, we have considered paths as linear interpolations between waypoints. This assumption makes the theoretical developments clearer. However, linear interpolations are continuous but not differentiable.

We have extended the method to piece-wise polynomial trajectories as follows.

A. Fitting initial path

The initial path computed by a random sampling method is composed of piecewise linear interpolations. The first step consists in fitting a C^1 piecewise polynomial curve following exactly the same path. For that, we replace each linear interpolation $(\mathbf{q}_j, \mathbf{q}_{j+1}, j \in \{0, N\}$ by a Bezier curve with 4 control points: $(P_{0,j}, P_{1,j}, P_{2,j}, P_{3,j})$ such that

$$\begin{aligned} P_{0,j} = P_{1,j} &= \mathbf{q}_j \\ P_{2,j} = P_{3,j} &= \mathbf{q}_{j+1} \end{aligned}$$

B. Cost

The cost of each path is defined by

$$C(\mathbf{x}) \triangleq \frac{1}{2} \sum_{j=0}^{N} \lambda_j \int_0^1 \|free_j \cup free_{j+1}(B_j(t))\|^2_{W_{j-1}} \quad (12)$$

where B_j is the j-th Bezier curve. This cost is again quadratic in the selected components of the Bezier curve control points that constitute the optimization variables:

$$\begin{aligned} \mathbf{x} \triangleq\ & (free_j(P_{0,j}), free_j \cup free_{j+1}(P_{1,j}), \\ & free_j \cup free_{j+1}(P_{2,j}), free_{j+1}(P_{0,j}))_{j \in \{0,\cdots,N\}} \end{aligned}$$

C. Boundary conditions

To make sure that the trajectory starts from $\mathbf{q}_0$ and ends at $\mathbf{q}_{N+1}$ with 0 velocities, we set the following constraint:

$$P_{0,0} = P_{1,0} = \mathbf{q}_0 \quad (13)$$

$$P_{2,N} = P_{3,N} = \mathbf{q}_{N+1} \quad (14)$$

D. Continuity constraints

To make sure that the trajectory is continuously differentiable, we set the following constraints on consecutive Bezier curves:

$$P_{3,j} = P_{0,j+1} \quad (15)$$

$$P_{3,j} - P_{2,j} = P_{1,j+1} - P_{0,j+1} \quad (16)$$

(15) corresponds to the continuity of the trajectory through waypoints, and (16) corresponds to derivability of the trajectory through waypoints.

Constraints (13),(14),(15),(16) are linear in the optimization variables. They are thus easy to integrate in the quadratic program in the same way as collision constraints.

E. Joint bound constraints

To make sure that the trajectory remains inside the joint bounds, we constrain all control points of the Bezier curves to remain within those bounds. An interesting property of Bezier curves is indeed that they are included in the convex hull of their control points.

These constraints are inequality constraints on some parameters of the optimization algorithm. They are handled easily using an active set method (See for instance qpOASES).

In the following section, we show some experimental results where our method optimizes paths using piecewise polynomial paths.

VII. Experimental results

We have applied our method to three different problems. For each problem, 20 different paths are planned to solve a manipulation problem using a random sampling algorithm. Each path is then optimized using the method described in this paper. The first optimization uses linear interpolation, while the second optimization builds continuously differentiable paths using sequences of Bezier curves with 4 control points (Section VI). Both optimization are run on the same input path, they are not consecutive.

The software is run on an "Intel(R) Core(TM) i7-3540M CPU @ 3.00GHz" with 4096 kB of cache memory, and 8039160 kB of RAM.

For each benchmark, the results are reported in a table with minimal, maximal and average values of

- the cost of the initial path,
- the cost of the optimized path,
- the ratio between the final cost and the initial cost,
- the number of iterations of the internal loop (Alg. 1 line 6),
- the number of collision constraints inserted by the algorithm,
- the number of waypoints of the initial (and final) path,
- the time of computation in seconds.

The movie attached to this paper displays one pair of initial – optimal paths for each benchmark.

A. Baxter benchmark

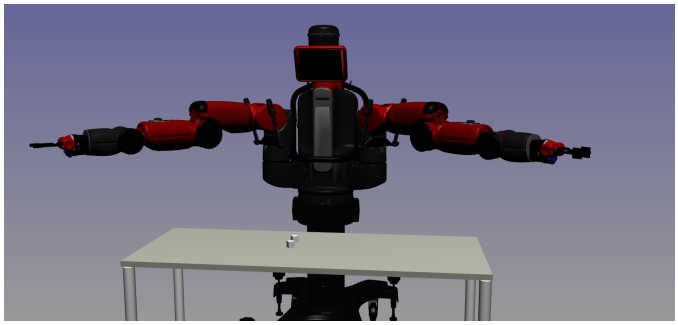

Fig. 7. Baxter robot manipulating boxes. The goal of the manipulation task is to swap the positions of the boxes possibly using both arms.

In this benchmark (Figure 7, tables I, II), Baxter is requested to swap two boxes on the table. The robot needs either to use both arms or to put a box in an intermediate position. The number of different collision objects is 40 and the number of pairs of objects to test for collision is 820.

	Initial cost	final cost	final cost / initial cost	number of iterations
min	50.98	11.74	0.12	8
max	170.31	93.24	0.55	179
mean	90.33	34.40	0.37	56.05
	number of collisions	number of waypoints	time of computation (seconds)	
min	4	17	2.74	
max	86	39	101.58	
mean	23.05	23.05	19.93	

TABLE I
RESULTS FOR BAXTER WITH LINEAR INTERPOLATIONS.

	Initial cost	final cost	final cost / initial cost	number of iterations
min	20.39	4.41	0.11	10
max	68.12	33.92	0.50	260
mean	36.13	12.51	0.34	97.2
	number of collisions	number of waypoints	time of computation (seconds)	
min	4	17	6.97	
max	128	39	212.29	
mean	42.65	23.05	48.41	

TABLE II
RESULTS FOR BAXTER WITH PIECEWISE BEZIER CURVES.

B. Construction set benchmark

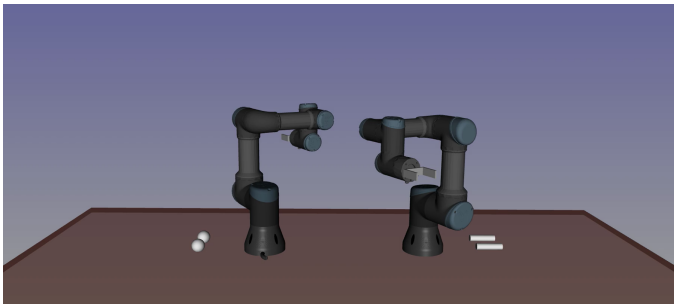

Fig. 8. Construction set: two robot arms assemble magnetic cylinders and spheres.

In this benchmark (Figure 8, tables III, IV), two UR-3 robots are requested to assemble two spheres on a cylinder. The number of different collision objects is 24 and the number of pairs of objects to test for collision is 293.

C. PR2 benchmark

In this benchmark (Figure 9, tables V-VI), PR 2 is requested to move a box from a table in front of it, to the top of a piece of furniture between it. The number of different collision objects is 170 and the number of pairs of objects to test for collision is 5089. The kitchen model (https://github.com/code-iai/iai_maps) is indeed composed of 146 boxes and cylinders.

	Initial cost	final cost	final cost / initial cost	number of iterations
min	220.48	128.45	0.35	1
max	653.56	230.49	0.69	47
mean	306.35	160.96	0.54	24.40
	number of collisions	number of waypoints	time of computation (seconds)	
min	0	21	2.24	
max	19	32	12.54	
mean	8.10	22.95	5.83	

TABLE III
RESULTS FOR THE CONSTRUCTION SET WITH LINEAR INTERPOLATIONS.

	Initial cost	final cost	final cost / initial cost	number of iterations
min	88.19	45.78	0.33	1
max	261.43	85.30	0.61	62
mean	122.54	57.35	0.48	27.15
	number of collisions	number of waypoints	time of computation (seconds)	
min	0	21	14.24	
max	27	32	61.51	
mean	9.55	22.95	22.99	

TABLE IV
RESULTS FOR THE CONSTRUCTION SET WITH PIECEWISE BEZIER CURVES.

	Initial cost	final cost	final cost / initial cost	number of iterations
min	44.34	12.15	0.15	1
max	101.18	58.16	0.81	79
mean	69.95	22.72	0.33	27.25
	number of collisions	number of waypoints	time of computation (seconds)	
min	0	12	4.04	
max	32	28	384.07	
mean	10.10	17.45	84.08	

TABLE V
RESULTS FOR PR-2 WITH LINEAR INTERPOLATIONS.

D. Result analysis

The average time of computation ranges from a 6 seconds to more than one minute for linear interpolations, and from around 20 seconds to 2 minutes for Bezier curves. The average number of iterations ranges from 25 to 56 for linear interpolations and from 27 to 100 for Bezier curves.

Note that for the same initial trajectory, Bezier curves have approximately twice as many parameters. This explains why the maximal number of iterations is higher.

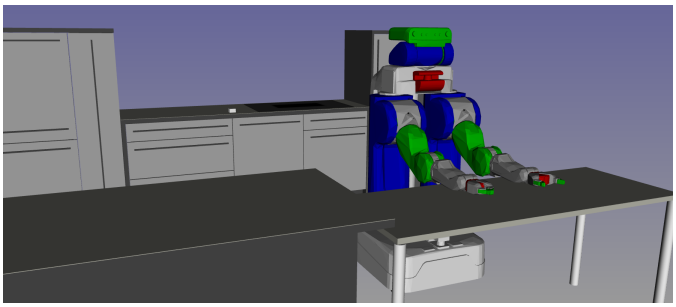

Fig. 9. PR2 moves a box from one place to another in a kitchen environment. The box is on the piece of furniture behind the robot in final configuration.

	Initial cost	final cost	final cost / initial cost	number of iterations
min	17.74	4.77	0.14	1
max	40.47	18.55	1	147
mean	28.08	8.64	0.33	48.53
	number of collisions	number of waypoints	time of computation (seconds)	
min	0	12	7.77	
max	71	28	420.63	
mean	20.63	17.45	115.06	

TABLE VI
RESULTS FOR PR-2 WITH PIECEWISE BEZIER CURVES.

Given the number of collision pairs in each example, and given the fact that robot bodies are composed of meshes, the time of computation is very reasonable.

VIII. CONCLUSION AND PERSPECTIVES

In this paper, we have presented an extension of an existing path optimization method to the problem of manipulation. The method has been tested on three benchmarks with satisfactory results.

The main advantage of the method is that it does not handle collision avoidance by sampling distance inequality constraints along the path. Instead, it introduces simple linearized constraints.

The main limitation of the method is that waypoint configurations that contain a closed kinematic chain – for instance configuration where a robot grasps an object – are not optimized.

In the future, we aim at addressing this latter issue by extending the algorithm to non-linear optimization with equality constraints. We will also test the combination of linear interpolations and Bezier curves by running first the method on linear interpolation and then on Bezier currves to make the resulting trajectory differentiable.

ACKNOWLEDGMENTS

REFERENCES

[1] Rachid Alami, Thierry Siméon, and Jean-Paul Laumond. A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps. In *5th International Symposium on Robotics Research*, Tokyo, Japan, 1989.

[2] Dmitry Berenson, Siddhartha S Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, page 0278364910396389, 2011.

[3] M. Campana, F. Lamiraux, and J. P. Laumond. A gradient-based path optimization method for motion planning. *Advanced Robotics*, 30(17-18):1126–1144, 2016. doi: 10.1080/01691864.2016.1168317. URL https://hal.archives-ouvertes.fr/hal-01301233.

[4] S. Dalibard, A. Nakhaei, F. Lamiraux, and J.P. Laumond. Manipulation of documented objects by a walking humanoid robot. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 518–523. IEEE, 2010.

[5] Andrew Dobson and Kostas Bekris. Planning representations and algorithms for prehensile multi-arm manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015.

[6] Mamoun Gharbi, Juan Cortés, , and Thierry Siméon. Roadmap composition for multi-arm systems path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Saint-Louis, USA, 2009.

[7] Dylan Hadfield-Menell, Christopher Lin, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Sequential quadratic programming for task plan optimization. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5040–5047. IEEE, 2016.

[8] K. Harada, T. Tsuji, and J.-P. Laumond. A manipulation motion planner for dual-arm industrial manipulators. in proceedings of. In *IEEE International Conference on Robotics and Automation*, pages 928—934, Hongkong, China, 2014.

[9] Kris Hauser and Victor Ng-Thow-Hing. Randomized multi-modal motion planning for a humanoid robot manipulation task. *The International Journal of Robotics Research*, 30(6):678–698, 2011.

[10] Sören Jentzsch, Andre Gaschler, Oussama Khatib, and Alois Knoll. MOPL: A multi-modal path planner for generic manipulation tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, September 2015. http://youtu.be/1QRvjBw58bU.

[11] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.

[12] Athanasios Krontiris and Kostas Bekris. Dealing with difficult instances of object rearrangement. In *Robotics Science and Systems*, Roma, Italy, 2015.

[13] Puttichai Lertkultanon and Quang-Cuong Pham. A single-query manipulation planner. *IEEE Robotics and Automation Letters*, 1(1):198–205, 2015.

[14] Wolfgang Merkt, Vladimir Ivan, and Sethu Vijayakumar. Continuous-time collision avoidance for trajectory optimization in dynamic environments. In *Intelligent Robots and Systems (IROS), 2019 IEEE/RSJ International Conference on*, 6 2019.

[15] Joseph Mirabel and Florent Lamiraux. Handling implicit and explicit constraints in manipulation planning. In *Robotics: Science and Systems*, Pittsburg, USA, June 2018. URL https://hal.archives-ouvertes.fr/hal-01804774.

[16] Joseph Mirabel, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylène Campana, Nicolas Mansard, and Florent Lamiraux. Hpp: a new software for constrained motion planning. In *IEEE/RSJ Intelligent Robots and Systems*, October 2016.

[17] Dennis Nieuwenhuisen, A Frank van der Stappen, and Mark H Overmars. An effective framework for path planning amidst movable obstacles. In *Algorithmic Foundation of Robotics VII*, pages 87–102. Springer, 2008.

[18] Jun Ota. Rearrangement of multiple movable objects-integration of global and local planning methodology. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2, pages 1962–1967. IEEE, 2004.

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

[20] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation planning with probabilistic roadmaps. *International Journal of Robotics Research*, 23(7/8), July 2004.

[21] Mike Stilman and James Kuffner. Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research*, 27(11-12):1295–1307, 2008.

[22] Marc Toussaint, Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics Science and Systems*, Pittsburgh, USA, June 2018.

[23] Gordon Wilfong. Motion planning in the presence of movable obstacles. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 279–288. ACM, 1988.

[24] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, and S. Srinivasa. CHOMP: covariant hamiltonian optimization for motion planning. *International Journal of Robotics Research*, 32 (9-10):1164–1193, 2013.