Abstract—This paper deals with manipulation planning. The problem consists in automatically computing paths for a system composed of one or several robots, with one or several grippers and one or several objects that can be grasped and moved by the robots. The problem gives rise to constraints that can be either explicit—an object is in a gripper—or implicit—an object is held by two different grippers. This paper proposes an algorithm that handles such sets of constraints and solves them in an explicit way as much as possible. When all constraints cannot be made explicit, substitution is operated between variables to make the resulting implicit constraint with as few variables as possible. The manipulation planning problem is modelled as a constraint graph that stores all the constraints of the problem.

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. Those constraints can be expressed numerically on the configuration variables of the robot and object. As such, constraint resolution is a key feature one must take care of. It has a direct influence on the completeness [3] and the efficiency [13] of the overall approach.

Pioneering work in motion planning [31, 11] have been initiated in the late 1980’s. Random path planning methods developed in the 1990’s [15] have been first adapted to the manipulation planning problem by Siméon et al. [26]. Recently, manipulation planning has regained interest with various approaches that tackle the inherent complexity of the problem. The domain is traditionally divided into several categories.

In navigation among movable obstacles (NAMO), the robot needs to move obstacles in order to reach the goal configuration [29, 22, 5]. Rearrangement planning consists in automatically finding a sequence of manipulation paths that move several objects from initial configurations to specified goal configurations [18, 23, 19]. Other research papers investigate multi-arm motion planning [9, 10, 6]. From a geometric perspective, manipulation planning is a hybrid problem the configuration space of which is a union of subspaces—called state—defined by constraints (gripper A holds object B for instance) on the positions of objects and robots. Motion within or from one to another state are also subject to numerical constraints. Mirabel et al. [20] propose a representation of the problem using a graph the nodes of which are states and the edges of which are manipulation paths. Nodes and edges both store numerical constraints.

Similar graph structures are also more or less explicitly used in various papers [5, 14, 11]. The partially discrete nature of the problem has also given rise to approaches that integrate task and motion planning techniques [4, 2, 7, 27, 12, 8]. Toussaint and Lopes [30] combine branch and bound with non-linear optimization techniques.

A manipulation path as computed in [20] is a concatenation of elementary linear interpolations subject to numerical constraints. To evaluate a configuration along such a path, the linear interpolation is first computed and then, the resulting configuration is projected on the sub-manifold that satisfies the constraints related to the current manipulation edge (Figure 1). This projection is performed numerically by iteratively solving the tangent linear system (Newton-Raphson). During path planning, a lot of constrained path are produced and evaluated for collision checking. It is therefore important that constraint resolution takes as little time as possible.

The contribution of this paper is a method that solves a set of numerical constraints. The method derives from [3, 5]. Some of the constraints are explicit, some are implicit. Resolution is performed via substitution of variables that can be explicitly expressed with respect to other variables, thus defining a non-linear constraint with less variables. The main benefit is a gain in terms of time of computation since explicit constraints are much faster to solve.

The paper is organized as follows. In Section II, we define implicit and explicit constraints, we explain how implicit constraints are numerically solved and we exhibit sufficient conditions under which the combination of two explicit constraints is still an explicit constraint. In Section III, we describe our algorithm that simplifies the resolution of a set of constraints by substituting variables that can be computed with respect to others and then by applying Newton-Raphson method. Section IV is devoted to the extension of our work to functions with values in the space of rigid transformations $SE(3)$. Finally, section V provides some experimental results showing...
the benefit of using explicit representation of constraints when possible.

II. PROBLEM DEFINITION

Let us consider a manipulation planning problem with \( R \) robots and \( N \) objects. The configuration space of the whole system is the Cartesian product of the configuration spaces of the robots and of the objects:

\[
\mathcal{C} = C_{r_1} \times \cdots \times C_{r_R} \times C_{o_1} \times \cdots \times C_{o_N}.
\]

For simple objects, \( C_o \) is the space of rigid-body motions denoted by \( SE(3) \), but more complex objects like a desk with drawers for instance, are represented by a kinematic chain.

A solution to a manipulation planning problem is a continuous curve in the configuration space is almost never a manipulation path. See [16] for an overview of the various possible.

A. Numerical constraint

More precisely, let us denote by \( (q_{r_1}, \cdots, q_{r_R}, q_{o_1}, \cdots, q_{o_N}) \in \mathcal{C} \), the configuration of the whole system. When robot \( i \in \{1, \cdots, R\} \) is grasping object \( j \in \{1, \cdots, N\} \), the position \( o_j \) completely depends on the position of robot \( i \), and the positions of all other objects are fixed:

\[
q_{o_j} = f_i(q_{r_i}) \quad (1)
\]

\[
q_{o_k} = p_k \in C_{o_k} \quad \forall k \in \{1, \cdots, N\}, k \neq j \quad (2)
\]

where \( f_i \) is a smooth mapping from \( C_{r_i} \) to \( C_{o_j} \), and \( p_k \) are constant positions of the objects that are not manipulated.

The above constraint is said to be explicit since some configuration variables (of the objects) can be computed with respect to others (of the robot).

Definition 1. An implicit constraint is defined by a smooth mapping \( g \) from \( \mathcal{C} \) to a vector space \( \mathbb{R}^m \). A configuration \( q \in \mathcal{C} \) is said to satisfy constraint \( g \) for threshold \( \varepsilon \) iff

\[
\|g(q)\| \leq \varepsilon.
\]

Definition 2. Combination of implicit constraints. Let \( m_1, m_2 \) be two positive integers, let \( g_1 \) from \( \mathcal{C} \) to \( \mathbb{R}^{m_1} \) and \( g_2 \) from \( \mathcal{C} \) to \( \mathbb{R}^{m_2} \) be two smooth mappings. The combination of the implicit constraints defined by \( g_1 \) and \( g_2 \) is the implicit constraint defined by the mapping \( g \) from \( \mathcal{C} \) to \( \mathbb{R}^{m_1 + m_2} \) that maps to \( q \in \mathcal{C} \) the vector

\[
\begin{pmatrix}
g_1(q) \\
g_2(q)
\end{pmatrix}.
\]

B. Implicit constraint resolution

As mentioned in the introduction, in [20], manipulation paths are concatenations of linear interpolations projected on the sub-manifold defined by manipulation constraints. In other words, evaluating a configuration along such a path consists in first computing the linear interpolation and then to solve the manipulation constraints from this configuration to obtain a configuration that satisfies the constraints. An important consequence is that paths are not discretized but implicitly defined everywhere on their definition interval. Figure 1 shows a simple example.

Algorithm 1 implicit-constraint-solver(q, g, \( \varepsilon \)):

\[
\begin{align*}
\alpha &= .1, i = 0, \alpha_{\text{max}} = .95 \\
\text{while} \quad &\|g(q)\| > \varepsilon \text{ and } i \leq 20 \\
&\text{// } (.)^+ \text{ denotes the Moore-Penrose pseudo-inverse} \\
&q \leftarrow q - \alpha \left( \frac{\partial g}{\partial q}(q) \right)^+ g(q) \\
&i \leftarrow i + 1 \\
&\text{// Make } \alpha \text{ tend toward } \alpha_{\text{max}} \\
&\alpha \rightarrow \alpha_{\text{max}} - .8 * (\alpha_{\text{max}} - \alpha) \\
\text{end while} \\
&\text{if } \|g(q)\| \leq \varepsilon \text{ then} \\
&\quad \text{return } q \\
&\quad \text{else} \\
&\quad \text{return failure}
\end{align*}
\]

Definition 3. An Implicit Constraint Solver is a mapping that takes as input an implicit constraint \( g \), a tolerance threshold \( \varepsilon \) and that returns a mapping \( h \) from a part \( D \) of \( \mathcal{C} \) to \( \mathcal{C} \) such that

\[
\forall q \in D, \|g(h(q))\| < \varepsilon
\]

\( h \) is called a projector on the sub-manifold satisfying \( g \).

The procedure described by Algorithm 1 previously introduced
in [3, 5] is an implicit constraint solver. In this case, \( D \) is the subspace of configuration for which the procedure converges to a solution. Note that the pseudo-inverse is computed by a singular value decomposition of complexity \( O(m^2n + mn^2 + n^3) \) where \( n \) is the dimension of the configuration space and \( m \) is the dimension of non-linear constraints. Constraint substitution as described in Section III makes both the dimension of constraints and the number of variables decrease in the Newton-Raphson resolution.

C. Explicit constraints

We denote by

- \( \dim C \) the dimension of the configuration vectors in \( C \),
- \( I_C \) the set of positive integers not greater than \( \dim C \),
- \( I \) a subset of \( I_C \),
- \( \bar{I} \) the complement in \( I_C \) of \( I \),
- \( |I| \) the cardinal of \( I \).

If \( q \in C \) is a configuration, we denote by \( q_I \in \mathbb{R}^{|I|} \) the vector composed of the components of \( q \) of increasing indices in \( I \).

a) Example: if \( q = (q_1, q_2, q_3, q_4, q_5, q_6) \) and \( I = \{1, 3, 6\} \), then \( q_I = (q_1, q_3, q_6) \), \( q_{\bar{I}} = (q_2, q_4, q_5) \).

Definition 4. An explicit constraint \( E = (in, out, f) \) is a mapping from \( C \) to \( C \), defined by the following elements:

- a subset of input indices \( in \subset \{1, \cdots, \dim C\} \),
- a subset of output indices \( out \subset \{1, \cdots, \dim C\} \),
- a smooth mapping \( f \) from \( \mathbb{R}^{\dim in} \) to \( \mathbb{R}^{\dim out} \),

satisfying the following properties:

- \( in \cap out = \emptyset \),
- for any \( p \in C \), \( q = E(p) \) is defined by
  \[ q_{out} = p_{out} \]
  \[ q_{out} = f(p_{in}) \]

With the same notation as above, if we define \( g \) as the mapping from \( C \) to \( \mathbb{R}^{\dim out} \) defined by

\[ g(p) = p_{out} - f(p_{in}), \quad (3) \]

\( E \) is a projector on the submanifold satisfying \( g \) with threshold equal to \( 0 \).

In other words, an explicit constraint can be formulated as an implicit constraint.

D. Combination of explicit constraints

We now establish some sufficient conditions under which the combination of 2 explicit constraints yields an explicit constraint and we show how to compute the resulting explicit constraint.

Let \( E_1 = (in_1, out_1, f_1) \) and \( E_2 = (in_2, out_2, f_2) \) be 2 distinct explicit constraints. If

\[ out_1 \cap out_2 = \emptyset \quad \text{and} \quad (in_1 \cap out_2 = \emptyset \text{ or } in_2 \cap out_1 = \emptyset), \quad (4) \]

the combination of both explicit constraints yields another explicit constraint \( E = (f, in, out) \) with input and output sets \( in \) and \( out \) defined later on.

Without loss of generality, by switching the constraints \( E_1 \) and \( E_2 \), we can reduce hypothesis (4) to

\[ out_1 \cap out_2 = \emptyset \quad \text{and} \quad in_1 \cap out_2 = \emptyset. \quad (5) \]

Under this assumption, the input and output spaces of \( f \) are defined by

\[ in = (in_1 \cup in_2) \setminus (in_2 \cap out_1), \]
\[ out = out_1 \cup out_2. \]

To evaluate \( E \), we build a tuple \( T \) of \( \dim C \) elements initialized to \( \emptyset \). For each

- \( i \in out_1 \), we set \( T[i] \) to \( E_1 \),
- \( i \in out_2 \), we set \( T[i] \) to \( E_2 \),

where \( T[i] \) denotes the \( i \)-th element of \( T \). Figures 2 gives an example.

\[
\begin{align*}
\cdots E_2 \ E_2 \ & \cdots \emptyset \emptyset \ E_1 \ E_1 \ E_1 \ & \cdots \emptyset \emptyset \\
\underbrace{\cdots E_2 \ E_2 \ & \cdots \emptyset \emptyset} \ & \underbrace{\cdots E_1 \ E_1 \ E_1 \ & \cdots \emptyset \emptyset} \\
\end{align*}
\]

Fig. 2: Combination \( E \) of two compatible explicit constraints \( E_1 \) and \( E_2 \). In red, output variables of \( E \); in blue, input variables of \( E \).

To compute \( q = E(p) \), for \( p \in C \), we first copy \( p \) into \( q \), we evaluate \( f_1(q_{in_1}) \) and update \( q_{out_1} \) with the result, and then we evaluate \( f_2(q_{in_2}) \) and update \( q_{out_2} \) with the result.

III. COMBINATION OF IMPLICIT AND EXPLICIT CONSTRAINTS

Let \( k_g \) and \( k_E \) be two non-negative integers. Given a set of \( k_g \) implicit constraints \( g_1, \cdots, g_{k_g} \) and \( k_E \) explicit constraints \( E_1, \cdots, E_{k_E} \), we wish to build a projector on the submanifold that satisfies all these constraints. We proceed as follows.

Let \( E \) be the first explicit constraint, and let \( g \) be the combination of \( g_1, \cdots, g_{k_g} \). For each \( j \) from 2 to \( k_E \).

- if \( E \) and \( E_j \) satisfy hypothesis (4), we replace \( E \) by the combination of \( E \) and \( E_j \),
- otherwise, we replace \( g \) by the combination of \( g \) with the implicit representation [5] of \( E_j \).

After this operation, we have one implicit constraint \( g \) with values in \( \mathbb{R}^m \), and at most one explicit constraint \( (in, out, f) \triangleq E \) that represent the same solution submanifold as \( g_1, \cdots, g_{k_g} \) and \( E_1, \cdots, E_{k_E} \) together.

\[ g(q) = 0 \quad (6) \]
\[ q_{out} = f(q_{in}) \quad (7) \]

A. Constraint resolution

Up to variable reordering, we assume that output variables and input variables are contiguos:

\[ q = (q_{out}, q_{in}, q_{linear}) \]

System [6][7] becomes

\[ g(f(q_{in})), q_{in}, q_{linear}) = 0 \quad (8) \]
\[ q_{out} = f(q_{in}) \quad (9) \]
We define \( \tilde{g} \) the mapping from \( \mathbb{R}^{\text{out}} \) to \( \mathbb{R}^m \) as

\[
\tilde{g}(q_{in}, q_{\text{inout}}) = g(f(q_{in}), q_{in}, q_{\text{inout}})
\]

To solve (8), we apply Algorithm 1 to \( \tilde{g} \). Note that \( \tilde{g} \) has less variables than \( g \). We need to compute the Jacobian of \( \tilde{g} \) with respect to \( q_{\text{out}} = (q_{in}, q_{\text{inout}}) \):

\[
\frac{\partial \tilde{g}}{\partial q_{in}}(q_{\text{out}}) = \frac{\partial g}{\partial q_{\text{out}}}(f(q_{in}), q_{\text{out}}) \frac{\partial f}{\partial q_{in}}(q_{in})
\]

\[
+ \frac{\partial g}{\partial q_{in}}(f(q_{in}), q_{\text{out}})
\]

As \( g \) is a combination of implicit constraint, its Jacobian is simply obtained by stacking the Jacobians of all the implicit constraints. When \( f \) is obtained from the combination of several explicit constraints, computing \( \partial f/\partial q_{in} \) is not as trivial.

**B. Jacobian of the combination of explicit constraints**

As in Section II-D, we denote by \( E \) the combination of two explicit constraints \( E_1 \) and \( E_2 \). The rows of the Jacobian of \( f \) correspond to output variables \( q_{\text{out}1} \) and \( q_{\text{out}2} \). The columns correspond to input variables with indices in \((in_1 \cup in_2) \setminus (in_2 \cap out_1)\). We define the following disjoint subsets of indices:

\[
\begin{align*}
  a &= in_1 \setminus in_2, \\
  b &= in_1 \cap in_2, \\
  c &= in_2 \setminus in_1 \setminus out_1, \\
  d &= in_2 \cap out_1.
\end{align*}
\]

With these definitions, we have

\[
\begin{align*}
  in_1 &= a \cup b, \\
  in_2 &= b \cup c \cup d, \\
  in &= a \cup b \cup c.
\end{align*}
\]

Note that in Figure 2 \( d = in_1 \cap in_2 = \emptyset \).

To compute the Jacobian of \( f \), we separate the variables:

\[
f(q_{in}) = \begin{pmatrix} f_1(q_a, q_b, q_c) \\ f_2(q_a, q_b, q_c) \end{pmatrix}
\]

To make notation tractable, we make an assumption on the order of the output variables and the input variables. The implementation is of course more general as explained in the next section. Note that in figures 3 and 2, the order of output variables is different.

\( f_1 \) only depends on \( q_a \) and \( q_b \). The Jacobian rows corresponding to \( f_1 \) thus contain the Jacobian of \( f_1, f_2 \) depends on \( q_c \), \( q_d \), and \( q_d \). The blocks corresponding to \( f_2 \) output and \( q_c \), \( q_d \) input contain the Jacobian of \( f_2 \), while variables \( q_d \) are output variables of \( f_1 \) the corresponding block is thus a product of Jacobians. Figure 3 provide expressions of the blocks that compose the Jacobian of \( f \).

C. Implementation details

To compute the value and Jacobian of a combination of explicit constraints, we have implemented a class called MatrixBlockView. The constructor takes as input two sets of indices rows and cols. For \( i, 0 \leq i < \text{rows} \) and \( j, 0 \leq j < \text{cols} \), let \( I \) be the \( i \) - 1 - th element of rows and \( J \) be the \( j \) - 1 - th element of cols. If

\[
\begin{align*}
  &\text{M} = \text{MatrixXd} \text { is a matrix of the Eigen library, then} \\
  &\text{Mv} = \text{MatrixBlockView (M, rows, cols)} \text{ emulates the Eigen matrix such that} \\
  &\text{Mv} (i,j) \text { is a reference to M (I,J) element.}
\end{align*}
\]

The Jacobian matrix \( J \) of explicit constraint \( f \) in the previous section represented in Figure 5 is thus computed as follows. \( J_{1,1} \) and \( J_{2,2} \) are the Jacobian matrices of \( f_1 \) and \( f_2 \).

IV. FUNCTION VALUED IN SE(3)

For clarity, the developments of the previous sections make the assumption that functions take values in \( \mathbb{R}^m \) where \( m \) is a positive integer. In equation (1) however, function \( f_1 \) represents the position of a rigid object and takes value in \( SE(3) \). Although the position of a rigid object can be represented by 6 values \((x,y,z,\text{roll, pitch, yaw})\), it is well known that this representation suffers a lot of shortcomings.

To represent an element of \( SE(3) \), we use a 7-dimensional vector \((q_0, \cdots q_6)\) such that:

\[
\begin{align*}
  &\| (q_0, q_1, q_5, q_6) \| = 1, \text{ quaternion } q_3i + q_4j + q_5k + q_6 \text{ encodes the rotation.}
\end{align*}
\]

The velocity of a rigid object, called a twist can be represented by a 6-dimensional vector \((v, \omega)\) where

\[
\begin{align*}
  &v \in \mathbb{R}^3 \text{ is the velocity of the point of the object that coincides with the origin of the world frame, expressed in the basis of the object,} \\
  &\omega \in \mathbb{R}^3 \text{ is the angular velocity of the object expressed in the basis of the object.}
\end{align*}
\]

See (21) chapter 2, Section 3.2 for details.

\footnote{The group of rigid-body transformations}
A. Addition and subtraction

Let \((\mathbf{v}, \omega) \in \mathbb{R}^6\) represent a twist and \(\mathbf{q} = (q_0, \cdots, q_6)\) represent an element of \(SE(3)\). Applying the constant twist \((\mathbf{v}, \omega)\) during unit time to \(\mathbf{q}\) leads to another element of \(SE(3)\) that we denote by:

\[
\mathbf{p} = \mathbf{q} + (\mathbf{v}, \omega)
\]

This definition of the addition is an extension of the addition on vector spaces where applying a constant velocity \(\mathbf{v}\) during unit time starting from a vector \(\mathbf{q}\) leads to \(\mathbf{q} + \mathbf{v}\).

We define the subtraction accordingly: if \(\mathbf{q}_1\) and \(\mathbf{q}_2\) are two elements of \(SE(3)\), \(\mathbf{q}_2 - \mathbf{q}_1\) is a vector of \(\mathbb{R}^6\) representing the (minimal) constant twist leading from \(\mathbf{q}_1\) to \(\mathbf{q}_2\) in unit time.

With this definition, we can extend the developments of the previous sections to functions with values in Cartesian products of vector spaces and \(SE(3)\).

V. Results

Our implementation of the above method is integrated in the package [hpp-constraints](https://github.com/ robotic-hpp) of the open-source platform HPP [20]. All tests are run on a processor Intel Xeon 3.5 Ghz, with 8 Go of RAM.

To analyse the influence of explicit constraint substitution into Newton-Raphson algorithm (Alg. 1), we compare our approach to the standard Newton-Raphson algorithm, as it is the most successful method [28, 3, 5] for general non-linear constraints.

A. Construction set

In this scenario, two UR3 robots, \((R_i)_{i \in \{0, 1\}}\) assemble a set of magnetised cylinders \((C_i)_{i \in \{0, 1\}}\) and spheres \((S_i)_{i \in \{0, 1\}}\) (Figure 4). The configuration space of the system is composed of vectors of the following form:

\[
\mathbf{q} = (\mathbf{q}_{R_0}, \mathbf{q}_{R_1}, \mathbf{x}_{C_0}, \mathbf{p}_{C_0}, \mathbf{x}_{C_1}, \mathbf{p}_{C_1}, \mathbf{x}_{S_0}, \mathbf{p}_{S_0}, \mathbf{x}_{S_1}, \mathbf{p}_{S_1}),
\]

where

- \(\mathbf{q}_{R_0}, \mathbf{q}_{R_1} \in \mathbb{R}^6\) represent the configurations of each robot,
- \(\mathbf{x}_{C_0}, \mathbf{x}_{C_1}, \mathbf{x}_{S_0}, \mathbf{x}_{S_1} \in \mathbb{R}^3\) represent the positions of the centers of the cylinders and spheres,
- \(\mathbf{p}_{C_0}, \mathbf{p}_{C_1}, \mathbf{p}_{S_0}, \mathbf{p}_{S_1} \in \mathbb{R}^4\) are unit quaternions representing the orientations of the cylinders and spheres.

We denote

\[
I_{R_0} = \{0, \cdots, 5\}, I_{R_1} = \{6, \cdots, 11\}, I_{C_0} = \{12, \cdots, 18\}, I_{C_1} = \{19, \cdots, 25\}, I_{S_0} = \{26, \cdots, 32\}, I_{S_1} = \{33, \cdots, 39\}
\]

the sets of indices corresponding to the configurations of respectively the robots, the cylinders, and the spheres.

The goal is to produce a sequence of manipulation paths in order to assemble two spheres on a cylinder.

We provide to the algorithm the high-level task sequence as follows:

1) \(R_0\) grasps \(S_0\),
2) \(R_1\) grasps \(C_0\),
3) \(S_0\) is assembled on \(C_0\),
4) \(R_0\) releases \(S_0\),
5) \(R_0\) grasp \(S_1\),
6) \(S_1\) is assembled on \(C_0\),
7) \(R_0\) releases \(S_1\),
8) \(R_1\) releases \(C_0\).

Given this sequence of actions, our software builds a constraint graph the nodes of which are states of the system and the edges of which are transitions that encode path constraints. Each state encodes a set of position constraints between robots and objects. The states are the following:

1) all objects lie on the ground,
2) \(R_0\) \(\rightarrow\) \(S_0\),
3) \(R_0\) \(\rightarrow\) \(S_0\) and \(R_1\) \(\rightarrow\) \(C_0\),
4) \(R_0\) \(\rightarrow\) \(S_0\) and \(R_1\) \(\rightarrow\) \(C_0\) and \(S_0\) \(\leftrightarrow\) \(C_0\),
5) \(R_1\) \(\rightarrow\) \(C_0\) and \(S_0\) \(\leftrightarrow\) \(C_0\),
6) \(R_0\) \(\rightarrow\) \(S_1\) and \(R_1\) \(\rightarrow\) \(C_0\) and \(S_0\) \(\leftrightarrow\) \(C_0\) and \(S_1\) \(\leftrightarrow\) \(C_0\),
7) \(R_0\) \(\rightarrow\) \(S_1\) and \(R_1\) \(\rightarrow\) \(C_0\) and \(S_0\) \(\leftrightarrow\) \(C_0\) and \(S_1\) \(\leftrightarrow\) \(C_0\),
8) \(R_1\) \(\rightarrow\) \(C_0\) and \(S_0\) \(\leftrightarrow\) \(C_0\) and \(S_1\) \(\leftrightarrow\) \(C_0\),
9) \(S_0\) \(\leftrightarrow\) \(C_0\) and \(S_1\) \(\leftrightarrow\) \(C_0\) and assembly lies on the ground.

where \(\rightarrow\) means “grasps”, and \(\leftrightarrow\) means “is stuck on”. Any configuration of any manipulation path lies in one of the above states. Each of states 1), 2), 3), 5), 6), 8), and 9) is composed of explicit constraints that are all compatible. For instance, in state 8), the position of \(C_0\) depends on the configuration of \(R_1\), and the positions of \(S_0\), \(S_1\) depend on the position of \(C_0\). Hence, the configuration of \(R_0\) uniquely determines the positions of \(C_0\), \(S_0\), and \(S_1\). On the contrary, states 4) and 7) form closed kinematic chains and require an implicit resolution. Our work handles all those sets of constraints in a
seamless and efficient way.

a) Variable substitution: let us describe the variable substitution algorithm for state 4). We denote by $E_1$, $E_2$, $E_3$ the constraints in this state:

\[ \begin{align*}
E_1 & = R_0 \rightarrow S_0 \quad (x_{S_0}, p_{S_0}) = f_{R_0 \rightarrow S_0}(q_{R_0}) \\
E_2 & = R_1 \rightarrow C_0 \quad (x_{C_0}, p_{C_0}) = f_{R_1 \rightarrow C_0}(q_{R_1}) \\
E_3 & = S_0 \leftrightarrow C_0 \quad (x_{S_0}, p_{S_0}) = f_{S_0 \leftrightarrow C_0}(x_{C_0}, p_{C_0})
\end{align*} \]

We first compose $E_1$ and $E_2$. Using the notation of Section II.D we have

\[ \begin{align*}
in_1 &= I_{R_0}, \quad out_1 = I_{S_0}, \quad in_2 = I_{R_1}, \quad out_2 = I_{C_0}.
\end{align*} \]

We notice immediately that Condition (4) is satisfied and therefore $E_1$ and $E_2$ are compatible. They combine into another explicit constraint $E = (in, out, f)$ such that

\[ \begin{align*}
out &= I_{S_0} \cup I_{C_0}, \quad in = I_{R_0} \cup I_{R_1}, \\
f(q_{R_0}, q_{R_1}) &= \left( \frac{f_{R_0 \rightarrow S_0}(q_{R_0})}{f_{R_1 \rightarrow C_0}(q_{R_1})} \right)
\end{align*} \]

Using the same reasoning, we can state that $E$ and $E_3$ are not compatible since (Condition 4)

\[ \begin{align*}
out \cap out_3 &= (I_{S_0} \cup I_{C_0}) \cap I_{S_0} \neq \emptyset.
\end{align*} \]

We therefore apply the constraint resolution algorithm described in Section III. After substitution, constraints (6.1) rewrite

\[ \begin{align*}
f_{R_0 \rightarrow S_0}(q_{R_0}) - f_{S_0 \leftrightarrow C_0}(f_{R_1 \rightarrow C_0}(q_{R_1})) &= 0 \\
(x_{S_0}, p_{S_0}) &= f_{R_0 \rightarrow S_0}(q_{R_0}) \\
(x_{C_0}, p_{C_0}) &= f_{R_1 \rightarrow C_0}(q_{R_1})
\end{align*} \]

Implicit resolution is performed on the first line of the above system using algorithm (1). Note that this constraint of dimension 6 implies only 12 degrees of freedom $(q_{R_0}, x_{C_0}, p_{C_0})$ out of 36.

To compute a solution path, we generate target configurations in each node, accessible from the initial configuration, and then we connect those targets by a visibility-PRM algorithm [25] in the appropriate sub-manifold. This is a variant of the method described in [24]. The manipulation planning algorithm is not the topic of this paper.

We analyse the improvements of the projection algorithm. From a set of 10000 random configurations, we compute the average computation time and success ratio of the projection with and without explicit constraint substitution. Figure 5 shows the results. As expected, explicit constraint substitution increases the success ratio and reduces the computation time. Note that in states 4) and 7), although constraints are solved implicitly, we observe an improvement due to the reduction of the number of variables. The accompanying video shows a solution path found to assemble the parts.

B. Romeo humanoid robot

In this scenario, the humanoid robot Romeo manipulates a placard with both hands, $\text{Hand}_{\text{right}}$ and $\text{Hand}_{\text{left}}$. Two

2Poses in SE(3) stand for 6 degrees of freedom although the are represented by 7 numbers.

3The model can be obtained at [http://doc.aldebaran.com/2-5.family/romeo/index_romeo.html](http://doc.aldebaran.com/2-5.family/romeo/index_romeo.html)

frames, $\text{Placard}_{\text{high}}$ and $\text{Placard}_{\text{low}}$, are defined on the placard, as shown on Figure 6. The three following states are considered:

1) $\text{Hand}_{\text{right}}$ grasps $\text{Placard}_{\text{high}}$.
2) $\text{Hand}_{\text{right}}$ grasps $\text{Placard}_{\text{high}}$ and $\text{Hand}_{\text{left}}$ grasps $\text{Placard}_{\text{low}}$.
3) $\text{Hand}_{\text{left}}$ grasps $\text{Placard}_{\text{low}}$.

Moreover, in all of these states, the following additional constraints are added to ensure robot balance:

- feet at fixed positions (dimension 12),
- robot center of mass above support polygon (dimension 2).

To our best knowledge, there is no simple explicit formulation of these constraints so a solver based on exact inverse kinematics would not be able to address this problem.

a) Variable substitution: let us describe the variable substitution algorithm for state 2) above. The configuration vector is of the form $q = (q_{\text{romeo}}, q_{\text{placard}})$ where $q_{\text{romeo}} \in \mathbb{R}^{40}$, and $q_{\text{placard}} \in \mathbb{R}^{7}$. The constraints are of the following form:

\[ \begin{align*}
q_{\text{placard}} &= f_{\text{left}}(q_{\text{romeo}}) \\
q_{\text{placard}} &= f_{\text{right}}(q_{\text{romeo}}) \\
f_{\text{balance}}(q_{\text{romeo}}, q_{\text{placard}}) &= 0
\end{align*} \]
Note that $f_{balance}$ output is of dimension 14. The first two constraints are explicit but incompatible. The system thus becomes

$$q_{placard} = f_{left}(q_{romeo})$$
$$f_{left}(q_{romeo}) - f_{right}(q_{romeo}) = 0$$
$$f_{balance}(q_{romeo}, f_{left}(q_{romeo})) = 0$$

The last two constraints are solved implicitly. They represent an implicit constraint of dimension 20 with 40 variables, instead of dimension 26 with 46 variables.

Fig. 7. Projection success ratio and computation time for humanoid robot problem. We compare fully implicit formulation (without substitution) to our combination of implicit and explicit constraint resolution (with substitution).

From a set of 10000 random configurations, we compare the success ratio and the average computation time of the projection algorithm with and without explicit constraint substitution. Figure 7 shows the results. For states 0) and 2), the success ratio and the computation time are both marginally influenced. In these states, the placard is held by one hand. For state 1), substitution has a great positive influence on the success ratio and computation time of the projection.

VI. CONCLUSION

This paper proposes a method to solve automatically nonlinear constraints expressed either implicitly or explicitly. Such constraints are in the core of the manipulation planning problem, especially when manipulation paths go through states where an object is grasped by several grippers.

Our method is integrated with success in a manipulation planning framework and experimental results show that huge improvement can be obtained.

It can of course be argued that the example provided in Section V-A can be solved with exact inverse kinematics. However, we claim that our approach applies to more general cases where exact inverse kinematics is not available, like in Section V-B where a humanoid robot in quasi-static equilibrium grasps objects.
ACKNOWLEDGMENTS

This work has been partially funded by the FLAG-ERA JTC project ROBOCOM++, which aims at rethinking robotics for the robot companion of the future. It has received funding from the European Community Seventh Framework Programme (FP7/2007-2013) under grant agreement 608849.

REFERENCES


