Prehensile Manipulation Planning: Modeling, Algorithms and Implementation

Florent Lamiraux, Joseph Mirabel

Prehensile Manipulation Planning: Modeling, Algorithms and Implementation

Florent Lamiraux[1], and Joseph Mirabel[1],
[1]LAAS-CNRS, University of Toulouse, France

This paper presents a software platform tailored for prehensile manipulation planning named Humanoid Path Planner. The platform implements an original way of modeling manipulation planning through a constraint graph that represents the numerical constraints that define the manipulation problem. We propose an extension of the RRT algorithm to manipulation planning that is able to solve a large variety of problems. We provide replicable experimental results via a docker image that readers may download to run the experimental results by themselves.

***Index Terms*—robotics, manipulation planning, constrained path planning, path planning**

I. Introduction

Today, robots in industrial manufacturing are mostly programmed by hand. They repeat the same motion thousands of times with great accuracy. However, automating a task with some variability is very challenging since it requires more programming effort to integrate sensors and motion planning in the process. A good example of this difficulty is the Amazon picking challenge [1]. The work described in this paper is a small step towards simplifying industrial process automation in the presence of some variability, like the variation of the initial position of some object or unknown obstacles. The work only covers motion planning – and more accurately manipulation planning. The integration into a whole process is still under development. We think that it is important not only to develop algorithms, but also to provide them within an open-source software platform in order to make the evaluation and then the integration of those algorithms easier.

Therefore this paper describes a software platform called Humanoid Path Planner tailored for manipulation planning in robotics. It can handle many types of robots, from manipulator arms to legged humanoid robots. Figure 1 displays an example of manipulation problem. The main contributions are:

- an original and general modeling of prehensile manipulation based on nonlinear constraints,
- an original solver for nonlinear constraints that can handle implicit and explicit constraints,
- a manipulation planning algorithm that tackles a great variety of manipulation planning problems,
- an open-source software suite that implements all the above, following state-of-the-art development tools and methods.
- a docker image of the aforementioned software with installation instructions provided with this paper. This image makes the experimental results replicable.

Installation instructions can be found at https://humanoid-path-planner.github.io/hpp-doc. This paper extends the work presented in previous papers [2],[3] with the following new material:

- description of the configuration space as a Cartesian product of Lie groups (Section III),
- unified and detailed definition of the grasp and placement constraints that are only mentioned in Mirabel *et al* [2] (Section V),
- automatic construction of the constraint graph (Section V),
- the docker image of the software,
- a description of the software platform (Section VII),
- experimental results for several different problems.

The paper is organized as follows. Section II presents some related work for constrained motion planning and manipulation planning. Section III introduces some preliminary notions like kinematic chains and Lie groups that are used to model the configuration space of each joint. Section IV introduces nonlinear constraints and solvers that are at the core of the manipulation problem definition. Section V defines the problem of prehensile manipulation in the general setting. Section VI provides a general algorithm that solves manipulation planning problems. Finally, Section VII is devoted to the software platform implementing the notions introduced in the previous sections. Experimental results are provided for a large variety of problems.

Each section is implemented by one or several software packages. For some values that need to be computed, rather than providing formulas, we sometimes give a link to the C++ or python implementation.

II. Related work

Motion planning has given rise to a lot of research work over the past decades. The problem consists in finding a collision-free path for a given system in an environment populated with obstacles. The field covers a large variety of different applications ranging from navigation for autonomous vehicles in partially known environments [4] to path planning for deformable objects [5], [6] and many other applications like coverage path planning [7], [8], or pursuit evasion planning [9].

Planning motions for high dimensional robots like humanoid robots or multi-arm systems has been shown to be

Manuscript received March 13, 2021; revised August 24, 2021. Corresponding author: F. Lamiraux (email: florent.lamiraux@laas.fr)

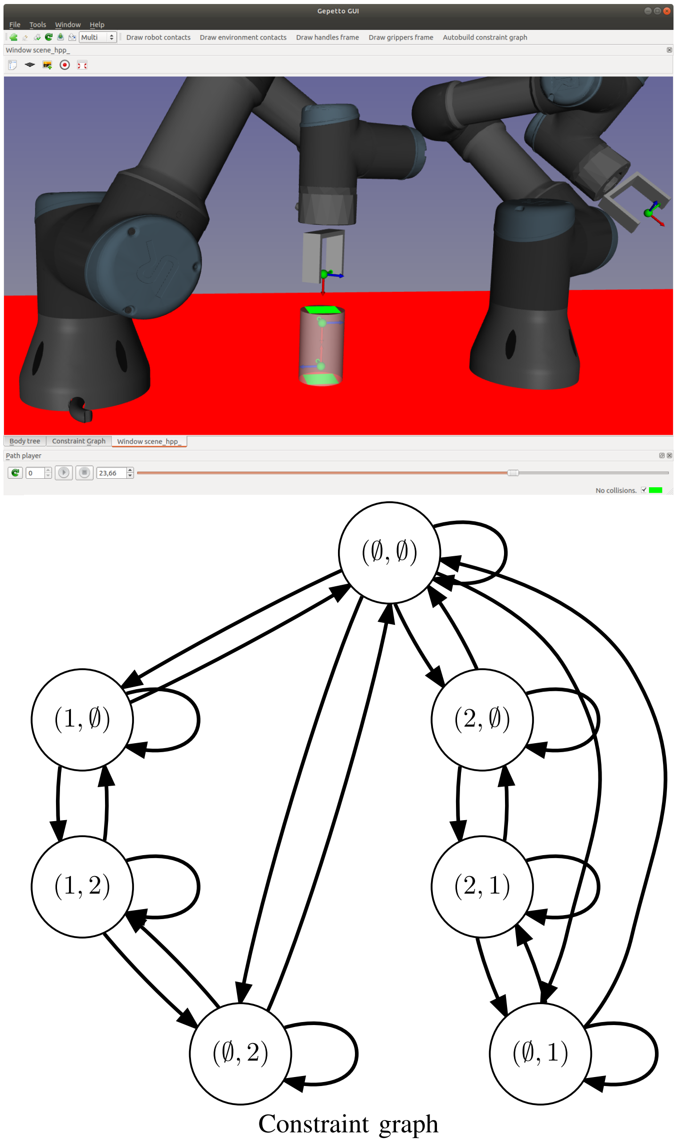

Fig. 1. Example of manipulation planning problem. Top: two UR3 robots with one gripper each (X=red, Y=green, Z=blue) manipulating a cylinder with two handles. The environment contains one rectangular contact surface (in red). The cylinder has two rectangular contact surfaces (in green). Bottom: the corresponding constraint graph. Names of states follow Expression (19): for example, $(\emptyset, 1)$ means that gripper of robot 2 grasps handle 1 of the cylinder. In this state, there is no placement constraint.

highly complex [10], [11]. Starting in the 1990's random sampling methods have been proposed to solve the problem, trading the completeness property against efficiency in solving problems in high dimensional configuration spaces [12], [13], [14]. The latter methods are said to be probabilistically complete since the probability to find a solution if one exists converges to 1 when the time of computation tends to infinity. Since then, asymptotically optimal random sampling algorithms have been proposed [15].

A. *Path planning with nonlinear constraints*

Some systems are subject to nonlinear constraints. These constraints define submanifolds of the configuration space the robot must stay on. For example, legged robots that must keep contact with the ground and enforce quasi-static equilibrium, or multi-arm systems grasping the same object are subject to this type of constraints. As the volume of the constrained manifold is usually equal to zero, sampling random configurations satisfying the constraints is an event of zero probability. To sample configurations on the constrained manifold, Dalibard *et al* [16] and Benrenson *et al* [17] project random configurations using a generalization of Newton-Raphson algorithm. Another method consists in expressing some configuration variables with respect to others [18], [3] whenever this can be done. Jaillet *et al* [19] propose another method based on nonlinear projection. They cover the constrained manifold by growing an atlas composed of local charts. This approximation provides a probability distribution that is closer to the uniform distribution over the manifold than the projection of a uniform distribution over the configuration space. Beobkyoon *et al* [20] propose a variation of the latter paper. The main difference lies in the fact that the nodes built on the tangent space are not immediately projected onto the manifold. Cefalo *et al* [21] put forward a general framework to plan task-constrained motions in the presence of moving obstacles. Kingston *et al* [22] provide an in-depth review of the various approaches to motion planning with nonlinear constraints.

B. *Manipulation planning*

Manipulation planning is a particular instance of path planning where some objects are moved by robots. Although several instances of the manipulation problem exist like manipulation by pushing [23], or by throwing [24], as well as multi-contact planning [25], [26], [27], in this paper, we are only concerned with prehensile manipulation. The configuration space of the whole system is subject to nonlinear constraints due to the fact that objects cannot move by themselves and should stay in a stable pose when not grasped by a robot. The accessible configuration space is thus a union of submanifolds as defined in the previous section. Each of these manifolds may moreover be a foliation where each leaf corresponds to a stable pose of an object or to a grasp of an object by a gripper. The geometrical structure of the problem has been well understood for a long time [28]. Some specific instances of the problem have even been addressed recently [29].

The first attempt to solve manipulation planning problems using random sampling was proposed by Siméon *et al* [30] where a reduction property simplifies the problem.

Papers about manipulation planning are commonly divided into several categories.

Navigation Among Movable Obstacles (NAMO) [31], [32] consists in finding a path for a robot that needs to move objects in order to reach a goal configuration. The final poses of the objects do not matter in this case.

Rearrangement planning [33], [34], [35], [36] consists in finding a sequence of manipulation paths that move some objects from an initial pose to a final pose. The final configuration of the robot is not specified. A simplifying assumption is the existence of a monotone solution, where each object is grasped at the most once and is moved from its initial pose to its final pose [32], [37], [38], [39], [33]. They mainly rely on two-level methods composed of a symbolic task planner and of a motion planner [40], [41], [42], [43].

Other contributions in manipulation planning explicitly address the problem of multi-arm manipulation [44], [45], [46], [47].

Schmitt *et al* [48] propose an approach where two robots manipulate an object in a dynamic environment. The output of the algorithm is a sequence of controllers rather than a sequence of paths.

Our work shares many ideas with Hauser and Ng-Thow-Hing [49] where the notion of constraint graph is present, although not as clearly expressed as in this paper. The main contribution of our work with respect to the latter paper is that the constraint graph is built automatically at the cost of a more restricted range of applications. We only address prehensile manipulation.

C. Open-source software platforms

Open-source software platforms are an important tool to enable fair comparison between algorithms. Several software platforms are available for motion planning and/or manipulation planning in the robotics community. Undoubtedly the most popular one is OMPL [50] which integrates many randomized path planning algorithms and is widely used for teaching purposes. Recently, Kingston *et al* [22] proposed an extension for systems subject to nonlinear constraints.

OpenRave [51] is a software platform that addresses motion and manipulation planning. It includes computation of forward kinematics.

One of the main differences between our solution and the previously cited ones lies in the way manipulation constraints are compiled into a graph. To our knowledge, none of the previous solutions can handle such a variety of problems as large as those described in Section VII-B.

III. PRELIMINARIES: KINEMATIC CHAINS AND LIE GROUPS

A kinematic chain is commonly understood as a set of rigid-body links connected to each other by joints. Each joint has one degree of freedom either in rotation or in translation. A configuration of the kinematic chain is represented by a vector. Each component of the vector represents the angular or linear value of the corresponding joint.

Although well-suited for fixed base manipulator arms, this representation is ill-suited for robots with a mobile base like wheeled mobile, aerial or legged robots, since the mobility of the base cannot be correctly represented by translation or rotation joints. Representing a free-flying object by three virtual translations followed by three virtual rotations referred to as roll, pitch and yaw is indeed a poor workaround due to the presence of singularities. A good illustration of this is the gimball lock issue that arose during Apollo 13 flight. To avoid singularities, the following definition is proposed.

A. Kinematic chain

A *kinematic chain* is a tree of *joints* where each joint represents the mobility of a rigid-body link with respect to another link or to the world reference frame. A configuration space called the *joint configuration space* is associated to each joint. The most common joints with their respective configuration spaces are

- linear *translation* with configuration space $\mathbb{R}$,
- bounded *rotation* with configuration space $\mathbb{R}$,
- unbounded *rotation* with configuration space $SO(2)$,
- *planar* joint with configuration space $SE(2)$,
- *freeflyer* joint with configuration space $SE(3)$.

$SO(n)$ and $SE(n)$ stand for *special orthogonal group* and *special Euclidean group* respectively. They represent the group of rotations and the group of rigid-body transformations in $\mathbb{R}^n$.

B. Lie groups

The joint configuration spaces listed in the previous paragraph: $\mathbb{R}^n$, $SO(n)$, and $SE(n)$ are all Lie groups. The group operation is $+$ for $\mathbb{R}^n$, and composition denoted as "." for $SE(n)$. We refer to Murray *et al* [52, Appendix A] for a thorough definition of Lie groups. Here we detail only those properties that are useful for the following developments.

For any Lie group $\mathcal{L}$ with neutral element $\mathfrak{n}$, the tangent space at the neutral element $T_{\mathfrak{n}}\mathcal{L}$ of the group naturally maps to the tangent space at any point of the group. This means that any *velocity* $\mathbf{v} \in T_{\mathfrak{n}}\mathcal{L}$ uniquely defines

1) a velocity $\mathbf{w} \in T_g\mathcal{L}$ at any point g of the group, and thus,
2) a vector field on the tangent space $T\mathcal{L}$, and
3) by integration during unit time of the latter vector field, starting from the origin, a new point $g_1 \in \mathcal{L}$.

Item 1 above is called the *transport* of velocity $\mathbf{v}$ to g. Item 3 is called the exponential map of $\mathcal{L}$ and is denoted by $\exp$.

1) Geometric interpretations

- $\mathbb{R}$ (and by trivial generalization $\mathbb{R}^n$): the neutral element is 0. The tangent space at 0 is isomorphic to $\mathbb{R}$ and
$$\forall\theta \in \mathbb{R},\ \exp(\theta) = \theta.$$
- $SE(3)$: an element g of $SE(3)$ can be seen as the position of a moving frame in a fixed reference frame. A point $\mathbf{x} \in \mathbb{R}^3$ is mapped to $g(\mathbf{x})$. Note that $\mathbf{x}$ is also the coordinate vector of $g(\mathbf{x})$ in the moving frame g. If $\mathbf{v}$, ω are linear and angular velocities at the origin, $(\mathbf{v}, \omega)$ is *transported* to g as the same linear and angular velocities expressed in the moving frame. In other words, if
$$M = \begin{pmatrix} R & \mathbf{t} \\ 0 & 1 \end{pmatrix} \quad (1)$$
with $R \in SO(3)$ and $\mathbf{t} \in \mathbb{R}^3$ is the homogeneous matrix representing g, and $(\mathbf{v}, \omega)$ is a velocity in $T_{I_3}SE(3)$, the velocity *transported* to g corresponds to linear and angular velocities $R\mathbf{v}$ and $R\omega$ of the moving frame. Integral curves of the vector field mentioned in item 2 above correspond to screw motions of constant velocity expressed in the moving frame.

$SE(2)$, $SO(3)$, and $SO(2)$ are subgroups of $SE(3)$ and follow the same geometrical interpretation.

Lie group type	configuration	velocity
$SE(3)$	$(x_1, x_2, x_3, p_1, \cdots, p_4)$ $\in \mathbb{R}^7$	$\dot{\mathbf{q}} = (\mathbf{v}, \omega)$ $\in \mathbb{R}^6$
$SE(2)$	$(x_1, x_2, \cos\theta, \sin\theta)$ $\in \mathbb{R}^4$	$\dot{\mathbf{q}} = (\mathbf{v}, \dot{\theta})$ $\in \mathbb{R}^3$
$SO(3)$	$(p_1, p_2, p_3, p_4) \in \mathbb{R}^4$	$\dot{\mathbf{q}} = \omega \in \mathbb{R}^3$
$SO(2)$	$(\cos\theta, \sin\theta) \in \mathbb{R}^2$	$\dot{\mathbf{q}} = \dot{\theta} \in \mathbb{R}$

TABLE I

MAIN LIE GROUP TYPES AND THEIR VECTOR REPRESENTATIONS. NOTICE THAT THE DIMENSIONS OF THE CONFIGURATION REPRESENTATION AND OF THE VELOCITY REPRESENTATION MAY DIFFER. USING $(\cos\theta, \sin\theta)$ INSTEAD OF θ FOR $SO(2)$ AND $SE(2)$ MAKES THE PARAMETERIZATION CONTINUOUS WHEN θ DISCONTINUOUSLY SWITCHES FROM $-\pi$ TO π

2) Vector representations

Each Lie group element is represented by a vector. Rotations are represented by unit quaternions.

Therefore elements of $SE(3)$ are represented by a vector in $\mathbb{R}^7$ where the first three components represent the image of the origin (vector $\mathbf{t}$ in Equation 1), the last four components (x, y, z, w) represent unit quaternion $w + xi + yj + zk$.

Elements of $SO(3)$ are likewise represented by a unit vector of dimension 4.

Elements of $SE(2)$ are represented by a vector of dimension 4. The first 2 components represent the image of the origin. The last 2 components represent the cosine and sine of the rotation angle. Therefore the homogeneous matrix associated to $\mathbf{q} = (q_1, q_2, q_3, q_4)$ is

$$M = \begin{pmatrix} q_3 & -q_4 & q_1 \\ q_4 & q_3 & q_2 \\ 0 & 0 & 1 \end{pmatrix}.$$

Table I compiles this information.

3) Exponential map

As expressed earlier, following a constant velocity[1] $\dot{\mathbf{q}}$ from the neutral element of a joint configuration space leads to another configuration denoted as

$$\mathbf{q} = \exp(\dot{\mathbf{q}}).$$

In some cases, we may specify in subscript the Lie group that is used: $\exp_{SO(3)}$, $\exp_{SE(3)}$.

For all Lie groups $\mathbb{R}$, $SO(n)$, $SE(n)$, the exponential map is surjective. This means that for any $\mathbf{q} \in \mathcal{L}$, there exists $\mathbf{v} \in T_{\mathrm{n}}\mathcal{L}$, such that $\mathbf{q} = \exp(\mathbf{v})$. Although exp is not injective, choosing the smallest norm $\mathbf{v}$ uniquely defines function log from $\mathcal{L}$ to $T_{\mathrm{n}}\mathcal{L}$, up to some singularities where several candidates $\mathbf{v}$ are of equal norms. Again, we may specify the Lie group that is used: $\log_{SE(3)}$, $\log_{SO(3)}$

4) Sum and difference notations

Following a constant velocity $\dot{\mathbf{q}} \in T_{\mathrm{n}}\mathcal{L}$ starting from $\mathbf{q}_0 \in \mathcal{L}$, leads to

$$\mathbf{q}_1 = q_0.\exp(\dot{\mathbf{q}}).$$

Note that if $\mathcal{L} = \mathbb{R}$, we write

$$\mathbf{q}_1 = \mathbf{q}_0 + \dot{\mathbf{q}},$$

[1]More precisely, following the vector field generated by $\dot{\mathbf{q}} \in T_{\mathrm{n}}\mathcal{L}$ according to the Lie group structure

since the Lie group operator of $\mathbb{R}$ is $+$ and $\exp_{\mathbb{R}}$ is the identity. In order to homogenize notation, we define the following operators. For any $\mathbf{q}_0, \mathbf{q}_1 \in \mathcal{L}$ and $\dot{\mathbf{q}} \in T_{\mathrm{n}}\mathcal{L}$:

$$\mathbf{q}_0 \oplus \dot{\mathbf{q}} \triangleq \mathbf{q}_0.\exp(\dot{\mathbf{q}}) \in \mathcal{L}, \quad (2)$$

$$\mathbf{q}_1 \ominus \mathbf{q}_0 \triangleq \log(\mathbf{q}_0^{-1}.\mathbf{q}_1) \in T_{\mathrm{n}}\mathcal{L}. \quad (3)$$

C. Robot configuration space

Given a kinematic chain with joints $(J_1, \cdots, J_{njoints})$, ordered in such a way that each joint has an index bigger than its parent in the tree, the configuration space of the robot is the Cartesian product of the joint configuration spaces.

$$\mathcal{C} \triangleq \mathcal{C}_{J_1} \times \cdots \times \mathcal{C}_{J_{njoints}}.$$

$\mathcal{C}$ naturally inherits the Lie group structure of the joint configuration spaces through the Cartesian product. We denote by nq_i, nv_i the sizes of the configuration and velocity vector representations of joint J_i, as defined in Table I. The configuration and velocity of the robot can thus be represented by vectors of size nq and nv such that

$$nq = \sum_{i=1}^{njoints} nq_i, \quad nv = \sum_{i=1}^{njoints} nv_i$$

We denote by iq_i, and iv_i the starting indices of joint i in the robot configuration and velocity vectors.

$$iq_i = \sum_{j=1}^{i-1} nq_j \quad iv_i = \sum_{j=1}^{i-1} nv_j$$

With these definitions and notation, the linear interpolation between two robot configurations $\mathbf{q}_0$ and $\mathbf{q}_1$ is naturally written:

$$\mathbf{q}(t) = \mathbf{q}_0 \oplus t(\mathbf{q}_1 \ominus \mathbf{q}_0)$$

This formula generalizes the linear interpolation to robots with free-flying bases, getting rid of singularities of roll – pitch – yaw parameterization. Cartesian products of Lie groups are represented by Class `LiegroupSpace`. Elements of these spaces are represented by classes

- `LiegroupElement`
- `LiegroupElementRef`, and
- `LiegroupElementConstRef`.

IV. NONLINEAR CONSTRAINTS AND SOLVERS

Some tasks require the robot to enforce some nonlinear constraints. Foot contact on the ground for a humanoid robot, center of mass projection on a horizontal plane, gaze constraint are a few examples.

A. Nonlinear constraints

Definition 1: **nonlinear constraint.** A nonlinear constraint is defined by a piece-wise differentiable mapping h from $\mathcal{C}$ to a vector space $\mathbb{R}^m$ and is written

$$h(\mathbf{q}) = 0. \quad (4)$$

If the robot is subject to several numerical constraints, $h_1, \cdots, h_k$ with values in $\mathbb{R}^{m_1} \cdots \mathbb{R}^{m_k}$, these constraints are equivalent to a single constraint h with values in $\mathbb{R}^m$, where $m = \sum_{i=1}^{k} m_i$, such that

$$h(\mathbf{q}) \triangleq \begin{pmatrix} h_1(\mathbf{q}) \\ \vdots \\ h_k(\mathbf{q}) \end{pmatrix}.$$

It may be useful to use a non-zero right hand side for the same function h. For that we define parameterized nonlinear constraints.

Definition 2: **Parameterized nonlinear constraint.** A parameterized nonlinear constraint is defined by a piece-wise differentiable mapping h from $\mathcal{C}$ to a vector space $\mathbb{R}^m$ and by a vector $\mathbf{h}_0$ of $\mathbb{R}^m$ and is written

$$h(\mathbf{q}) = \mathbf{h}_0.$$

Piece-wise differentiable mappings are represented by abstract Class `DifferentiableFunction`.

1) Jacobian

In this paper, we will make use of the term Jacobian in a generalized way. If h is a piece-wise differentiable function from a Lie group $\mathcal{L}_1$ to a Lie group $\mathcal{L}_2$, and $\mathbf{q}_1$ an element of $\mathcal{L}_1$, we will denote by $\frac{\partial h}{\partial \mathbf{q}}(\mathbf{q}_1)$ the operator that maps velocities in $T_{\mathbf{q}_1}\mathcal{L}_1$ to the velocity in $T_{h(\mathbf{q}_1)}\mathcal{L}_2$ *transported by* h[2].

This operator is represented by a matrix with nv_2 lines and nv_1 columns, where nv_1 and nv_2 are the dimensions of the tangent spaces of $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively.

B. Newton-based solver

It is sometimes useful to produce a configuration $\mathbf{q}$ that satisfies a constraint (or a set of constraints) of type (4) from a configuration $\mathbf{q}_0$ that does not. This action is called the *projection of* $\mathbf{q}_0$ *onto the submanifold defined by the constraint* and is performed by a Gauss-Newton solver [53, Chapter 10] that iteratively linearizes the constraint as follows:

$$h(\mathbf{q}_{i+1}) \approx h(\mathbf{q}_i) + \frac{\partial h}{\partial \mathbf{q}}(\mathbf{q}_i)(\mathbf{q}_{i+1} \ominus \mathbf{q}_i) = 0$$

Iterate $\mathbf{q}_{i+1}$ is computed as follows:

$$\mathbf{q}_{i+1} = \mathbf{q}_i \ominus \alpha_i \frac{\partial h}{\partial \mathbf{q}}^{+}(\mathbf{q}_i) h(\mathbf{q}_i) \tag{5}$$

where $.^+$ denotes the Moore Penrose[3] pseudo inverse, and α_i is a positive real number called the step size. Taking $\alpha_i = 1$ solves the linear approximation, but it may not be the best choice in general.

The computation of α_i is performed by a line search algorithm. The algorithm stops when the norm of each $h_i(\mathbf{q}_{i+1})$ is below a given error threshold. Class `HierarchicalIterative` implements the above Newton method. Several line search methods are implemented:

- `Backtracking` [54],
- `ErrorNormBased`:
$$\alpha_i = C - K \tanh\left(a \frac{\|f(\mathbf{q}_i)\|}{\epsilon^2} + b\right),$$
where C, K, a, and b are constant values, and ϵ is the error threshold,
- `FixedSequence` implements a fixed sequence of α_i that converges to 1,
- and `Constant` sets α_i to 1.

Note that to define a new constraint, the user needs to derive class `DifferentiableFunction` and to implement methods `impl_compute` and `impl_jacobian`.

C. Explicit constraints

In manipulation planning applications in which robots manipulate objects, once an object is grasped, the position of the object can be explicitly computed from the configuration of the robot. In this case, some configuration variables of the system depend on other configuration variables:

$$\mathbf{q} = (\mathbf{q}_{rob}, \mathbf{q}_{obj}) \in \mathcal{C}, \qquad \mathbf{q}_{obj} = g_{grasp}(\mathbf{q}_{rob}).$$

Although this constraint may fit definition (4) by defining

$$h(\mathbf{q}) \triangleq \mathbf{q}_{obj} \ominus g_{grasp}(\mathbf{q}_{rob}), \tag{6}$$

solving this constraint possibly with other constraints using an iterative scheme (5) is obviously sub-optimal.

More generally, let us denote by

- I_{nq} the set of positive integers not greater than $nq = \dim \mathcal{C}$,
- I a subset of I_{nq},
- $\bar{I}$ the complement in I_{nq} of I,
- $|I|$ the cardinal of I.

If $\mathbf{q} \in \mathcal{C}$ is a configuration, we denote by $\mathbf{q}_I \in \mathbb{R}^{|I|}$ the vector composed of the components of $\mathbf{q}$ of increasing indices in I.

1) Example

if $\mathbf{q} = (q_1, q_2, q_3, q_4, q_5, q_6, q_7)$ and $I = \{1, 2, 6\}$, then $\mathbf{q}_I = (q_1, q_2, q_6)$, $\mathbf{q}_{\bar{I}} = (q_3, q_4, q_5, q_7)$.

Similarly, if

- m and n are two integers,
- M and N are two subsets of I_m and I_n respectively,
- J is a matrix with m rows and n columns,

we denote by

$$J_{M,N} \tag{7}$$

the matrix of size $|M| \times |N|$ obtained by extracting the rows of J of indices in M and the columns of J with indices in N.

2) Example

If $m = 3$, $n = 4$, $M = \{2, 3\}$ and $N = \{1, 2, 4\}$,

$$J = \begin{pmatrix} J_{1,1} & J_{1,2} & J_{1,3} & J_{1,4} \\ J_{2,1} & J_{2,2} & J_{2,3} & J_{2,4} \\ J_{3,1} & J_{3,2} & J_{3,3} & J_{3,4} \\ J_{4,1} & J_{4,2} & J_{4,3} & J_{4,4} \end{pmatrix}$$

then

$$J_{M \times N} = \begin{pmatrix} J_{2,1} & J_{2,2} & J_{2,4} \\ J_{3,1} & J_{3,2} & J_{3,4} \end{pmatrix}$$

[2] If $\dot{\mathbf{q}} \in T_{\mathbf{q}_1}\mathcal{L}_1$ is a velocity along a time parameterized curve γ, $\frac{\partial h}{\partial \mathbf{q}}(\mathbf{q}_1)\dot{\mathbf{q}}$ is the velocity along Curve $h(\gamma)$.

[3] who has just been awarded the Nobel Prize.

Definition 3: An explicit constraint $E = (in, out, f)$ is a mapping from $\mathcal{C}$ to $\mathcal{C}$, defined by the following elements:

- a subset of input indices $in \subset \{1, \cdots, nq\}$,
- a subset of output indices $out \subset \{1, \cdots, nq\}$,
- a smooth mapping f from $\mathbb{R}^{|in|}$ to $\mathbb{R}^{|out|}$,

satisfying the following properties:

- $in \cap out = \emptyset$,
- for any $\mathbf{p} \in \mathcal{C}$, $\mathbf{q} = E(\mathbf{p})$ is defined by

$$\begin{aligned} \mathbf{q}_{\bar{out}} &= \mathbf{p}_{\bar{out}} \\ \mathbf{q}_{out} &= f(\mathbf{p}_{in}). \end{aligned}$$

D. Solver by substitution

To optimize constraint resolution, we perform variable substitution whenever possible in order to reduce the number of variables as well as the dimension of the resulting implicit constraint. Here we describe the method first published in Mirabel *et al* [3]. Unlike in the former paper, the description we give in Algorithm 1 is closer to the real implementation. Some links to the source code are indeed provided in the algorithm description. Once several compatible explicit constraints have been inserted in the solver, they behave as a single constraint. For example, if $\mathbf{q} = (\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3)$,

$$\left\{ \begin{array}{lcl} \mathbf{q}_1 &=& f_1(\mathbf{q}_2) \\ \mathbf{q}_2 &=& f_2(\mathbf{q}_3) \end{array} \right. \text{ becomes } \left[\begin{array}{c} \mathbf{q}_1 \\ \mathbf{q}_2 \end{array} \right] = \left[\begin{array}{c} f_1(f_2(\mathbf{q}_3)) \\ f_2(\mathbf{q}_3) \end{array} \right],$$

and f_2 should be evaluated before f_1.

1) Substitution

When an explicit constraint is not successfully added following Algorithm 1, it is handled as an implicit constraint. Therefore, after inserting implicit and explicit constraints, the solver stores a system of equations equivalent to one explicit and one implicit constraints that we denote by:

$$\begin{aligned} h(\mathbf{q}_{in}, \mathbf{q}_{out}) &= 0, && (8) \\ \mathbf{q}_{out} &= f(\mathbf{q}_{in}), \text{ where} && (9) \\ in \cap out = \emptyset. & && (10) \end{aligned}$$

Substituting (9) into (8), we define an implicit constraint on $\mathbf{q}_{in}$ only:

$$\tilde{h}(\mathbf{q}_{in}) \triangleq h(\mathbf{q}_{in}, f(\mathbf{q}_{in})) = 0$$

The *solver by substitution* applies iteration (5) to $\tilde{h}$, instead of h. Therefore we need to compute the Jacobian of $\tilde{h}$:

$$\frac{\partial \tilde{h}}{\partial \mathbf{q}_{in}} = \frac{\partial h}{\partial \mathbf{q}_{in}} + \frac{\partial h}{\partial \mathbf{q}_{out}} . \frac{\partial f}{\partial \mathbf{q}_{in}}.$$

As the Jacobian of h is provided with the implicit constraint, we need to compute $\frac{\partial f}{\partial \mathbf{q}_{in}}$. Let us recall that f may be the combination of several compatible explicit constraints. Let us denote by E the mapping from $\mathcal{C}$ to $\mathcal{C}$ associated to f by Definition 3. Let J denote the $nv \times nv$ Jacobian matrix of E. Then J is defined by blocks as follows:

$$\begin{array}{ll} J_{in \times in} = I_{|in|} & J_{in \times out} = 0 \\ J_{out \times in} = \frac{\partial f}{\partial \mathbf{q}_{in}} & J_{out \times out} = 0 \end{array} \quad (11)$$

Algorithm 1 Insertion of an explicit constraint in the solver. Line 1 is called once at initialization of the solver. $explicit$ is a vector that stores the constraints that are successfully added to the solver. nc is the size of the latter. $args$ is an array that, for each configuration variable stores the index in $explicit$ of the constraint that computes this configuration variable, -1 if no constraint computes the index. Procedure ADD tests whether explicit constraint E is compatible with the previously inserted constraints. Line 6 checks whether any output variable of E is already computed by a previous explicit constraint. If so the procedure returns failure and E is not inserted. The Loop at line 9 recursively checks that any element of out is not an input variable of a previously inserted constraint. If the loop ends without returning failure, line 18 stores the information that elements of out are computed by E and E is inserted in the vector of constraints. Function computeOrder at line 20 recursively computes the order in which the explicit constraints are evaluated, following the rule that the input of a constraint should be evaluated before the output.

1: **procedure** INITIALIZESOLVER
2: $explicit \leftarrow$ empty vector of explicit constraints
3: $nc \leftarrow 0$
4: $args \leftarrow$ array of size nq filled with -1
5: **function** ADD($E = (in, out, f)$)
6: **if** $args_{out}$ contains an element ≥ 0 **then**
7: return failure
8: queue $idxArg \leftarrow$ elements of in
9: **while** $idxArg$ **not** empty **do**
10: $iArg \leftarrow idxArg$ first element
11: remove $idxArgs$ first element
12: **if** $iArg \in out$ **then**
13: return failure
14: **if** $args[iArg] == -1$ **then**
15: continue
16: **else**
17: push $explicit[args[iArg]].in$ elements into $idxArg$
18: fill $args_{out}$ with nc
19: explicit.add(E); $nc \leftarrow nc + 1$
20: computeOrder()
21: **return** success

If E is the composition of several explicit constraints $E_i = (in_i, out_i, f_i)$ of Jacobian J_i, $i \in I_{nc}$, for an integer nc, then

$$J = \prod_{i=nc}^{1} J_i, \quad (12)$$

with J_i obtained by expression (11) after replacing in, out, and f by in_i, out_i, and f_i.

$\frac{\partial f}{\partial \mathbf{q}_{in}}$ is then obtained by extracting from J block $out \times in$.

Let us now detail the iterative computation of (12). Let J be the product of J_j for j from nc to $i+1$. Note that if J_i and J are square matrices of size nv, of the form (11), $J_i.J$

can be computed by block as follows:

$$(J_i.J)_{in_i \times I_{nv}} = J_{in_i \times I_{nv}}$$
$$(J_i.J)_{out_i \times I_{nv}} = \frac{\partial f_i}{\partial \mathbf{q}_{in_i}}.J_{in_i \times I_{nv}}$$

and as columns out of J are equal to 0, left multiplying J by J_i consists in modifying only the following block of J:

$$(J_i.J)_{out_i \times in} = \frac{\partial f_i}{\partial \mathbf{q}_{in_i}}.J_{in_i \times in}$$

Other coefficients of J_iJ are equal to the corresponding coefficients of J. An implementation of the aforementioned Jacobian product can be found here.

The solver by substitution described in this section is implemented by Class `SolverBySubstitution`, that stores an instance of `ExplicitConstraintSet`.

2) Important remark

As mentioned in Table I, the configuration and velocity vectors may have different sizes. As a consequence, index sets in and out in Definition 3 correspond to configuration vector indices, while in Expression (11), they correspond to velocity vector indices. To keep notation simple, we use the same notation for different sets.

E. Constrained path

Now that we are able to project configurations onto submanifolds defined by numerical constraints – up to some numerical threshold, we need to define paths on such submanifolds. The usual way of doing so is by discretizing the path and projecting each sample configuration. The shortcoming is that it requires choosing a discretization step at path construction thus losing the continuous information of the path.

Instead, we propose an alternative architecture where paths store the constraints they are subject to and apply the constraints at path evaluation (*i.e.* when computing the configuration at a given parameter). Let $P \in C^1([0,T],\mathcal{C})$ be a path without constraint defined on an interval $[0,T]$, and **proj** a projector onto a submanifold defined by numerical constraints (*i.e.* an instance of `SolverBySubstitution`).

Then the corresponding constrained path $\tilde{P}$ is defined on the same interval by

$$\forall t \in [0,T], \quad \tilde{P}(t) = \mathbf{proj}(P(t))$$

Paths are implemented by Class `Path`. Several implementations of unconstrained paths are provided:
`StraightPath` for linear interpolation generalized to Lie groups, `ReedsSheppPath`, `DubinsPath` for nonholonomic mobile robots.

1) Continuity of projection along a path

Projecting configurations at path evaluation has the advantage of not losing information. In return, the projection of a continuous path may be discontinuous. Thus, before inserting a projected path in a roadmap for example, it is necessary to detect possible discontinuities. Hauser [55] proposes a solution to this problem. In a previous paper [56], we described two algorithms to check whether a projected path is continuous. These algorithms are implemented by classes `pathProjector::Dichotomy` and `pathProjector::Progressive`. Note that when a path is not continuous, the algorithms return a continuous portion of the path starting at the beginning of the path. This enables function EXTEND in Algorithm 4 to create a new node.

V. Manipulation Problem

The previous sections have presented how we model kinematic chains, configurations and velocities for a given robotic system and how configurations and paths can be projected onto a submanifold of the configuration space defined by numerical constraints.

In this section, we will use these notions to represent a robotic manipulation problem.

Definition 4: **Prehensile manipulation problem**
A prehensile manipulation problem is defined by

- one or several robots,
- one or several objects,
- a set of possible grasps,
- environment contact surfaces,
- object contact surfaces,
- an initial configuration,
- a final configuration.

Admissible configurations of the system are configurations that satisfy the following property:

- each object is either grasped by a robot, or lies in a stable contact pose,
- the volumes occupied by the links of the robots and by the objects are pair-wise disjoint.

Admissible motions of the system are motions that satisfy the following property:

- configurations along the motion are admissible, and
- the pose of objects in stable contact is constant,
- the relative pose of objects grasped by a gripper with respect to the gripper is constant.

The solution of a prehensile manipulation problem is an admissible motion that links the initial and goal configurations.

We will now provide precise definitions for grippers, grasps and stable contact poses.

A. Grasp

1) Configuration space

The configuration space of a manipulation problem is the Cartesian product of the configuration spaces of the robots and of the objects.

$$\mathcal{C} = \mathcal{C}_{r_1} \times \cdots \times \mathcal{C}_{r_{nr}} \times SE(3)^{no}$$

where nr is the number of robots, no the number of objects, $\mathcal{C}_{r_i},\ i \in \{1,...,nr\}$ is the configuration space of robot r_i.

Definition 5: **Gripper.** A *gripper* $\mathfrak{g}$ is defined as a frame attached to the link of a robot. $\mathfrak{g}(\mathbf{q}),\ \mathbf{q} \in \mathcal{C}$ denotes the pose of the frame when the system is in configuration $\mathbf{q}$.

Definition 6: **Handle.** A *handle* is composed of

- a frame $\mathfrak{h}$ attached to the root joint of an object,

- a list $flags = (x, y, z, rx, ry, rz)$ of 6 Boolean values.

$\mathfrak{h}(\mathbf{q})$, $\mathbf{q} \in \mathcal{C}$ denotes the pose of the frame when the system is in configuration $\mathbf{q}$.

Definition 7: **Grasp.** A *grasp* is a numerical constraint h over $\mathcal{C}$, defined by

- a gripper $\mathfrak{g}$,
- a handle $\mathfrak{h}$.

Let $\bar{h}$ be the smooth mapping from $\mathcal{C}$ to $\mathbb{R}^6$ defined by

$$\bar{h}(\mathbf{q}) = \log_{\mathbb{R}^3 \times SO(3)} \left(\mathfrak{g}^{-1}(\mathbf{q})\mathfrak{h}(\mathbf{q})\right). \quad (13)$$

$h(\mathbf{q})$ is obtained by extracting from $\bar{h}$ the components the values of which are `true` in the handle flag.

Note that $\mathbb{R}^3 \times SO(3)$ and $SE(3)$ have different group operators, exponential maps and logarithms. Constant velocity motions in $SE(3)$ are screw motions while constant velocity motions in $\mathbb{R}^3 \times SO(3)$ consist of linear interpolation of the center of the frame and constant angular velocity.

Definition 8: **Grasp complement.** Given a grasp constraint defined by gripper $\mathfrak{g}$, handle $\mathfrak{h}$ and some flag vector, the grasp complement is a parameterized nonlinear constraint defined by

$$h_{comp}(\mathbf{q}) = \mathbf{h}_0$$

where h_{comp} is composed of the components of $\bar{h}$ that are not in h and $\mathbf{h}_0$ is a vector with the same size as h_{comp} output.

2) Geometric interpretation and examples

The first three components of $\bar{h}(\mathbf{q})$ in equation (13) correspond to the position of the center of $\mathfrak{h}(\mathbf{q})$ in the frame of $\mathfrak{g}(\mathbf{q})$. The last three components of $\bar{h}(\mathbf{q})$ are a vector representing the relative orientation of $\mathfrak{h}(\mathbf{q})$ with respect to $\mathfrak{g}(\mathbf{q})$. The direction of the vector represents the axis of rotation, the norm of the vector represents the angle of rotation.

- If $flags = (\texttt{true}, \texttt{true}, \texttt{true}, \texttt{true}, \texttt{true}, \texttt{true})$ the grasp is satisfied iff $\mathfrak{g}(\mathbf{q})$ and $\mathfrak{h}(\mathbf{q})$ coincide:

$$h = \bar{h},$$

  h_{comp} is an empty constraint;
- if $flags = (\texttt{true}, \texttt{true}, \texttt{true}, \texttt{true}, \texttt{true}, \texttt{false})$ the grasp is satisfied iff the centers and z axes of $\mathfrak{g}(\mathbf{q})$ and $\mathfrak{h}(\mathbf{q})$ coincide (free rotation around z). This is useful for cylindrical objects:

$$h = (\bar{h}_1, \bar{h}_2, \bar{h}_3, \bar{h}_4, \bar{h}_5),$$
$$h_{comp} = (\bar{h}_6);$$

- if $flags = (\texttt{true}, \texttt{true}, \texttt{false}, \texttt{true}, \texttt{true}, \texttt{false})$ the grasp is satisfied iff the center of $\mathfrak{h}(\mathbf{q})$ is on the z axis of $\mathfrak{g}(\mathbf{q})$ and if the z axes of $\mathfrak{g}(\mathbf{q})$ and $\mathfrak{h}(\mathbf{q})$ coincide (free translation and rotation around z). This is also useful for cylindrical objects:

$$h = (\bar{h}_1, \bar{h}_2, \bar{h}_4, \bar{h}_5),$$
$$h_{comp} = (\bar{h}_3, \bar{h}_6),$$

  however inequality constraints need to be added manually on $\bar{h}_3$ to limit the translation.
- if $flags = (\texttt{true}, \texttt{true}, \texttt{true}, \texttt{false}, \texttt{false}, \texttt{false})$ the grasp is satisfied iff the centers of $\mathfrak{g}(\mathbf{q})$ and $\mathfrak{h}(\mathbf{q})$ coincide (free rotation). This is useful for spherical objects.

$$h = (\bar{h}_1, \bar{h}_2, \bar{h}_3),$$
$$h_{comp} = (\bar{h}_4, \bar{h}_5, \bar{h}_6).$$

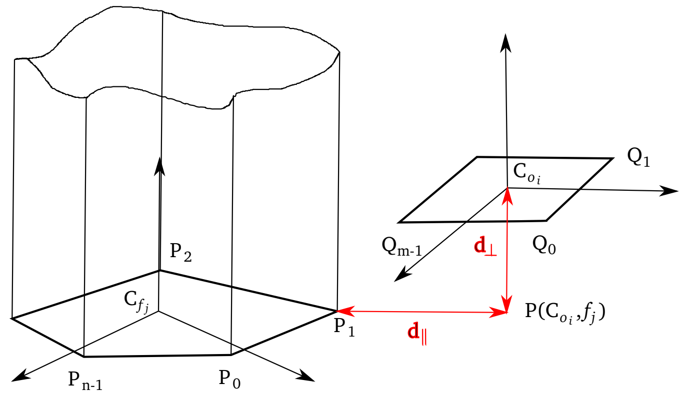

Fig. 2. Distance defined by two convex polygons.

If $\mathbf{q}_0$ is a configuration satisfying the grasp constraint: $h(\mathbf{q}_0)$=0, then the submanifold defined by

$$\{\mathbf{q} \in \mathcal{C}, \quad h(\mathbf{q}) = 0 \quad h_{comp}(\mathbf{q}) = h_{comp}(\mathbf{q}_0)\}$$

contains all the configurations that are reachable from $\mathbf{q}_0$ while maintaining the grasp. Note that this representation of relative pose constraints has been used in the Stack of Task software, although it is not described in the corresponding paper [57]. It is different from Task Space Regions [17] where open domains of $SE(3)$ are defined.

B. Stable contact pose

When an object is not grasped, it should lie in a stable pose. There are two simple methods to enforce that:

1) defining virtual grippers in the environment and virtual handles on the object, implicitly defines a discrete set of poses,
2) defining a virtual gripper on a horizontal plane and a virtual handle on the object, and using a grasp with flags $(\texttt{false}, \texttt{false}, \texttt{true}, \texttt{true}, \texttt{true}, \texttt{false})$ constrains the object to move along an infinite horizontal plane.

Here we propose a third method that enables users to define contact surfaces in a more flexible way. To this end, we denote by:

- $(o_i)_{i \in I}$ a set of convex polygons attached to an object,
- $(f_j)_{j \in J}$ a set of convex polygons attached to the environment or to a mobile part of a robot that can receive objects (mobile robot for example),
- respectively C_{o_i}, $\mathbf{n}_{o_i}$ the barycenter of o_i and the normal to the plane containing o_i,
- C_{f_j}, $\mathbf{n}_{f_j}$ the barycenter of f_j and the normal to the plane containing f_j,
- $P(C_{o_i}, f_j)$, the orthogonal projection of C_{o_i} onto the plane containing f_j.

Then we define the distance between polygons o_i and f_j as the distance of C_{o_i} to the cylindrical volume of generatrix $\mathbf{n}_{f_j}$ and of directrix f_j:

$$d(f_j, o_i) = \sqrt{d_{\parallel}^2 + d_{\perp}^2} \tag{14}$$

where

i, j are the indices that minimize the above distance,

$$d_{\parallel} = \begin{cases} d(f_j, P(C_{o_i}, f_j)) & \text{if} \quad P(C_{o_i}, f_j) \text{ outside } f_j \\ 0 & \text{otherwise} \end{cases}$$

$$d_{\perp} = \mathbf{n}_{f_j}.\vec{C_{f_j}C_{o_i}}.$$

Figure 2 illustrates this definition.

We denote by $o_i(\mathbf{q})$ and $f_j(\mathbf{q})$ the poses in configuration $\mathbf{q}$ of frames with respective centers C_{o_i} and C_{f_j} and with x-axis normal to each polygon. Similarly as in Definition 7, we define

$$\bar{h}(\mathbf{q}) = \log_{\mathbb{R}^3 \times SO(3)} \left(f_j(\mathbf{q})^{-1} o_i(\mathbf{q}) \right) \tag{15}$$

The contact constraint is defined by the following piece-wise differentiable function:

$$h(\mathbf{q}) = \begin{cases} (\bar{h}_1, 0, 0, \bar{h}_5, \bar{h}_6) & \text{if} \quad P(C_{o_i}, f_j) \text{ inside } f_j \\ (\bar{h}_1, \bar{h}_2, \bar{h}_3, \bar{h}_5, \bar{h}_6) & \text{if} \quad P(C_{o_i}, f_j) \text{ outside } f_j \end{cases} \tag{16}$$

It is straightforward that this function vanishes if and only if two convex polygons o_i and f_j are in contact and if C_{o_i} is inside f_j.

As for grasps, we need to define a parameterized complement constraint for the contact constraint in order to specify the submanifold of configurations reachable from one configuration while keeping the object in a constant stable pose. The naive way consists in defining

$$h_{comp}(\mathbf{q}) = \begin{cases} (\bar{h}_2, \bar{h}_3, \bar{h}_4) & \text{if} \quad P(C_{o_i}, f_j) \text{ inside } f_j \\ (0, 0, \bar{h}_4) & \text{if} \quad P(C_{o_i}, f_j) \text{ outside } f_j \end{cases}$$

$\bar{h}_2$, $\bar{h}_3$, $\bar{h}_4$ respectively represent the translation in $y - z$ plane and the rotation around x-axis of frame o_i with respect to frame f_j. Let $\mathbf{q}_0$ be a configuration such that $h(\mathbf{q}_0) = 0$. The submanifold defined by

$$\{\mathbf{q} \in \mathcal{C}, \quad h(\mathbf{q}) = 0 \quad h_{comp}(\mathbf{q}) = h_{comp}(\mathbf{q}_0)\} \tag{17}$$

contains one object pose for each pair of polygons (o_i, f_j), and there are $|I|.|J|$ possible combinations. Thus this constraint is not suitable to enforce object immobility along a path since the object may jump from one pose to another.

To disambiguate the various combinations of convex polygons that can be in contact, we define

$$h_{comp}(\mathbf{q}) = \begin{cases} (\bar{h}_2 + 2jM, \bar{h}_3 + 2iM, \bar{h}_4) \\ \text{if } P(C_{o_i}, f_j) \text{ inside } f_j \\ (2jM, 2iM, \bar{h}_4) \\ \text{if } P(C_{o_i}, f_j) \text{ outside } f_j \end{cases} \tag{18}$$

where

- i and j are the indices that minimize distance (14),
- M is a positive real number such that for any $\kappa \in J$, all vertices of f_κ are included in the disk of center C_{f_κ} and of radius M.

With this definition, the submanifold defined by (17), (16), (18) contains configurations where the object is in a unique stable pose. The polygon indices i and j, as well as their relative position can indeed be recovered from (18):

$$\begin{aligned} i &= \left\lfloor \frac{\bar{h}_3}{2M} + \frac{1}{2} \right\rfloor \\ j &= \left\lfloor \frac{\bar{h}_2}{2M} + \frac{1}{2} \right\rfloor \\ \bar{h}_1 &= \bar{h}_5 = \bar{h}_6 = 0 \\ \bar{h}_2 &= h_{comp\ 1} - 2jM \\ \bar{h}_3 &= h_{comp\ 2} - 2iM \\ \bar{h}_4 &= h_{comp\ 3} \end{aligned}$$

and from (15),

$$f_j(\mathbf{q})^{-1} o_i(\mathbf{q}) = \exp_{\mathbb{R}^3 \times SO(3)} \left(\bar{h} \right).$$

Uniqueness comes from the fact that when two convex polygons are in contact, necessarily, $|\bar{h}_2| \leq M$,$|\bar{h}_3| \leq M$.

C. Merging constraint and complement into an explicit constraint

Note that when a grasp constraint and its complement are combined, they constitute an explicit constraint since the pose of the object grasped uniquely depends on the configuration of the robot that grasps the object.

Similarly, when a placement constraint and its complement are combined, they constitute an explicit constraint since the pose of the object placed uniquely depends on the pose of the contact surface on which the object is placed. This latter pose

- either depends on the configuration of the robot the contact surface belongs to,
- or is constant if the contact surface belongs to the environment.

In any case, the explicit expression of the object pose depends on the right hand side of the complement constraint that is constant along transition paths.

During the construction of the constraint graph (described in Section V-D), grasp and placement constraints, their complements and the associated explicit constraints are created together and registered using method `registerConstraint` of Class `ConstraintGraph`.

D. Constraint graph

According to Definition 4, the set of admissible configurations of a manipulation problem is the union of submanifolds of the configuration space of the system. Each submanifold is defined by grasp and/or stable contact constraints. We call each submanifold a *state* of the problem.

A state can be defined by a subset of active grasps, any object not grasped being in a stable contact pose. Let $n_{\mathfrak{g}}$, $n_{\mathfrak{h}}$, and n_o respectively denote the number of grippers, handles, and objects.

We denote by

- $grasp_{ij}$ $i \in \{1, \cdots, n_{\mathfrak{g}}\}$ $j \in \{1, \cdots, n_{\mathfrak{h}}\}$ the grasp constraint of handle j by gripper i,

- $grasp_{ij}/comp$ $i \in \{1, \cdots, n_\mathfrak{g}\}$ $j \in \{1, \cdots, n_\mathfrak{h}\}$ the complement constraint of the latter,
- $place_i$ $i \in \{1, \cdots n_o\}$ the placement constraint of object i,
- $place_i/comp$ $i \in \{1, \cdots n_o\}$ the complement constraint of the latter.

A state $\mathcal{S}$ is denoted by a vector of size $n_\mathfrak{g}$:

$$\mathcal{S} = \left(h_1, \cdots, h_{n_\mathfrak{g}}\right) \quad (19)$$

where $h_i \in \{\emptyset, 1, \cdots, n_\mathfrak{h}\}$ denotes the index of the handle grasped by gripper i; $h_i = \emptyset$ means that gripper i does not grasp any handle.

1) Number of states

note that for $i \in \{1, \cdots, n_\mathfrak{h}\}$ the number of occurrences of i in $\mathcal{S}$ is at the most 1: a handle cannot be grasped by several grippers. Note also that the number of occurrences of $\emptyset$ is not limited: several grippers may hold nothing. Let m be a non-negative integer not greater than $\min(n_\mathfrak{g}, n_\mathfrak{h})$ and let us count the number of states with m handles grasped. The number of subset of m handles among $n_\mathfrak{h}$ is equal to $\frac{n_\mathfrak{h}!}{(n_\mathfrak{h}-m)!m!}$. And the number of ways of dispatching them among the $n_\mathfrak{g}$ grippers is equal to $\frac{n_\mathfrak{g}!}{(n_\mathfrak{g}-m)!}$. Thus, the total number of states is equal to

$$\sum_{m=0}^{\min(n_\mathfrak{g}, n_\mathfrak{h})} \frac{n_\mathfrak{h}!}{(n_\mathfrak{h}-m)!m!} \frac{n_\mathfrak{g}!}{(n_\mathfrak{g}-m)!}$$

Definition 9: **adjacent states**

Two states $\mathcal{S}_1 = (h_{11}, \cdots, h_{n_\mathfrak{g}1})$ and $\mathcal{S}_2 = (h_{12}, \cdots, h_{n_\mathfrak{g}2})$ are adjacent to each other if they differ by only one grasp and the grasp is empty in one of the states:

$$\exists i \in \{1, \cdots, n_\mathfrak{g}\},\ h_{i1} \neq h_{i2} \text{ and } (h_{i1} = \emptyset \text{ or } h_{i2} = \emptyset), \text{ and}$$
$$\forall j \in \{1, \cdots, n_\mathfrak{g}\},\ j \neq i, h_{j1} = h_{j2}.$$

Definition 10: **Constraint graph**

The constraint graph related to a manipulation problem as defined in Definition 4 is a graph

- the nodes of which are states defined by subsets of grasps (19),
- two edges (back and forth) connect two states if they are adjacent to each other,
- one edge connects each state to itself.

Edges are also called *transitions*. Nodes contain

- the grasp constraints that are active in the corresponding state,
- a placement constraint for each object that is not grasped by any handle.

Transitions contain

- the constraints of the node they connect with the least active grasps,
- the parameterized complement constraint of each of the latter.

2) Example

To illustrate the notions expounded in the previous sections, let us consider an example of two UR3 robots manipulating a cylinder illustrated in Figure 1. The robot is fitted with one gripper attached to the end-effector. The cylinder

state	active constraints
$(\emptyset, \emptyset)$	$place_1$
$(j, \emptyset)$, $j \in \{1, 2\}$	$grasp_{1j}$
$(\emptyset, j)$, $j \in \{1, 2\}$	$grasp_{2j}$
(i, j) $i, j \in \{1, 2\}$	$grasp_{1i}, grasp_{2j}$

TABLE II
STATE CONSTRAINTS.

transition	belongs to	additional constraints
$(\emptyset, \emptyset) \to (i, \emptyset)$	$(\emptyset, \emptyset)$	$place_1/comp$
$(\emptyset, \emptyset) \to (\emptyset, i)$	$(\emptyset, \emptyset)$	$place_1/comp$
$(i, \emptyset) \to (i, j)$	$(i, \emptyset)$	$grasp_{1i}/comp$
$(\emptyset, j) \to (i, j)$	$(\emptyset, j)$	$grasp_{2j}/comp$

TABLE III
TRANSITION CONSTRAINTS: i, j ARE EITHER 1 OR 2. COLUMN "BELONGS TO" MEANS THAT PATHS ALONG THE TRANSITION BELONG TO THE STATE, *i.e.* THE TRANSITION CONTAINS THE STATE CONSTRAINTS.

is equipped with two handles and with two square contact surfaces corresponding to the top and bottom sides of the cylinder. $n_\mathfrak{g} = 2, n_\mathfrak{h} = 2, n_o = 1$. The flag of the handles are

$$(\texttt{true}, \texttt{true}, \texttt{true}, \texttt{false}, \texttt{true}, \texttt{true}).$$

Therefore grasp constraints are of dimension 5 and keep the rotation of the gripper around the cylinder axis free. Table II indicates which constraints are active for each state and Table III for each transition.

3) Automatic construction

Given a set of grippers, handles and objects, the constraint graph can be constructed automatically. Here is an implementation in python. Algorithm 2 describes this implementation. Functions

- GRASPCONSTRAINT,
- GRASPCONSTRAINTCOMP, build grasp constraint and complement constraint as defined in Section V-A,
- PLACECONSTRAINT,
- PLACECONSTRAINTCOMP build placement constraints and complement as defined in Section V-B,
- EXISTSTATE(Gr) returns `true` if a state has already been created for the set of grasps given as input,
- STATE(Gr) returns the state created with the set of grasps given as input,
- OBJECTINDEX(h) returns the index of the object handle h belongs to.

VI. MANIPULATION PLANNING

In this section, we show how the constraint graph defined in the previous section is used to plan collision-free manipulation paths. Although we are working on an extension of the RMR* algorithm [58] to several grippers, objects and handles, the only manipulation planning algorithm available so far in HPP is an extension of the RRT algorithm described in the next section.

A. Manipulation-RRT

Manipulation Randomly exploring Random Tree is an extension of the RRT algorithm [59] that grows trees in the

Algorithm 2 Recursive Construction of the constraint graph. The construction starts by the state with no grasp. Call to RECURSE function loops over the available grippers and handles and creates states with one more grasp, and a transition to these new states. In each state, a placement constraint is added for each object of which no handle is grasped. Variables $\mathcal{G}$ and $\mathcal{H}$ contain the indices of the free grippers and handles. Variable Gr stores the current set of grasps following Expression (19). Lines 5 to 9 compute which objects are not grasped. Lines 20 to 23 insert placement constraints in the state for those objects. Line 24 recurses only if the latest node reached is new. Functions CREATESTATE and CREATETRANSITION are given in Algorithm 3.

1: **global variables**
2: n_o ▷ number of objects
3: $n_\mathfrak{g}$ ▷ number of grippers
4: $n_\mathfrak{h}$ ▷ number of handles
5:
6: **function** BUILDCONSTRAINTGRAPH
7: $\mathcal{G} \leftarrow [0, \cdots, n_\mathfrak{g} - 1]$ ▷ list of gripper indices
8: $\mathcal{H} \leftarrow [0, \cdots, n_\mathfrak{h} - 1]$ ▷ list of handle indices
9: $Gr \leftarrow [\emptyset, \cdots, \emptyset]$ ▷ list of size $n_\mathfrak{g}$
10: RECURSE($\mathcal{G}$,$\mathcal{H}$,Gr)
11: **function** RECURSE($\mathcal{G}$,$\mathcal{H}$,Gr)
12: CREATESTATE(Gr)
13: **if** $\mathcal{G} = \emptyset$ or $\mathcal{H} = \emptyset$ **then**
14: **return**
15: **for** g in $\mathcal{G}$ **do**
16: $\mathcal{G}' \leftarrow \mathcal{G} \setminus \{g\}$
17: **for** h in $\mathcal{H}$ **do**
18: $\mathcal{H}' \leftarrow \mathcal{H} \setminus \{h\}$
19: $Gr' \leftarrow Gr$
20: $Gr'[g] \leftarrow h$
21: $isNewState \leftarrow$ **not** EXISTSTATE(Gr')
22: CREATESTATE(Gr')
23: CREATETRANSITION(Gr, Gr')
24: **if** $isNewState$ **then** RECURSE($\mathcal{G}'$,$\mathcal{H}'$,Gr')

Algorithm 3 Method CREATESTATE builds the constraints relative to a state: one grasp constraint for each grasp, and one placement constraint for each object not grasped. CREATETRANSITION builds the constraints relative to a transition: the constraints of the initial state (with the fewest grasps) and their complements.

function CREATESTATE(Gr)
if EXISTSTATE(Gr) **then**
return
$\mathcal{S} \leftarrow$ new state
$\mathcal{S}.Pl \leftarrow$ [true, ..., true] ▷ list of size n_o
for g in $[0, \cdots, n_\mathfrak{g} - 1]$ **do**
$h \leftarrow Gr[g]$
$\mathcal{S}.Pl$[OBJECTINDEX(h)] $\leftarrow$ false
$\mathcal{S}$.ADD(GRASPCONSTRAINT(g,h))
for o in $[0, \cdots, n_o]$ **do**
if $\mathcal{S}.Pl[o]$ **then**
$\mathcal{S}$.ADD(PLACECONSTRAINT(o))
function CREATETRANSITION(Gr_1, Gr_2)
$\mathcal{T} \leftarrow$ new transition(Gr_1, Gr_2)
$\mathcal{S}_1 \leftarrow$STATE(Gr_1) ▷ Recover state for this set of grasps
for g in $[0, \cdots, n_\mathfrak{g} - 1]$ **do**
$h \leftarrow Gr_1[g]$
$\mathcal{T}$.ADD(GRASPCONSTRAINT(g,h))
$\mathcal{T}$.ADD(GRASPCONSTRAINTCOMP(g,h))
for o in $[0, \cdots, n_o]$ **do**
if $\mathcal{S}_1.Pl[o]$ **then**
$\mathcal{T}$.ADD(PLACECONSTRAINT(o))
$\mathcal{T}$.ADD(PLACECONSTRAINTCOMP(o))
$\mathcal{T}_1 \leftarrow$ new transition(Gr_2, Gr_1)
$\mathcal{T}_1$.SETCONSTRAINTS($\mathcal{T}$.CONSTRAINTS())

free configuration space, exploring the different states of the manipulation problem. Algorithm 4 describes the algorithm implemented in C++ here.

After initializing the roadmap with the initial and goal configurations, the algorithm iteratively calls method ONESTEP until a solution path is found or the maximum number of iterations is reached. This method picks a random configuration (line 6) and for each connected component of the roadmap and each state of the constraint graph, extends the nearest node in the direction of the random configuration (lines 7–10). For each successful extension, the end of the extension path is stored for subsequent connections (line 11). After the extension step, the algorithm tries to connect new nodes to other connected components using two strategies:

1) function TRYCONNECTNEWNODES calls method CONNECT between all pairs of new nodes,
2) function TRYCONNECTTOROADMAP tries to connect each new node to the nearest nodes in other connected components of the roadmap also using function CONNECT.

Function CONNECT attempts to connect two configurations in two states. First, it checks whether there exists a transition between the states. If so, it checks that the right hand side of the parameterized constraints of the transition is the same for both configurations (up to the error threshold). Then it returns the linear interpolation between the configurations, projected onto the submanifold defined by the transition constraints. If the path is in collision or discontinuous, only the continuous collision-free part at the beginning of the path is returned.

Function EXTEND attempts to generate a path from a configuration in a state to another state following a random transition. Similarly as for function CONNECT, the path is projected onto the submanifold defined by the transition constraints. The end configuration is obtained by applying to the random configuration the constraints of the transition and of the goal state.

B. Examples

In this section, the algorithm described in the previous section is depicted with two examples. Figure 3 shows function EXTEND defined in the previous section applied to the example of Figure 1. The system considered is composed of two

Algorithm 4 Manipulation RRT algorithm iteratively calls method ONESTEP until a solution path is found or the maximum number of iterations is reached. Function CONNECT is described in Algorithm 5

function INITIALIZEROADMAP($\mathbf{q}_{init}$,$\mathbf{q}_{goal}$)
  $\Gamma \leftarrow$ new roadmap
  Γ.ADDNODE($\mathbf{q}_{init}$); Γ.ADDNODE($\mathbf{q}_{goal}$)
function ONESTEP(Γ)
  $newNodes \leftarrow$ empty list
  $\mathbf{q}_{rand} \leftarrow$SHOOTRANDOMCONFIG( )
  for cc in connected components of Γ **do**
    for s in constraint graph states **do**
      $\mathbf{q}_{near} \leftarrow$NEARESTNODE($cc$,$s$,$\mathbf{q}_{rand}$)
      $p \leftarrow$EXTEND(s, $\mathbf{q}_{near}$, $\mathbf{q}_{rand}$)
      if p **then** $newNodes \leftarrow newNodes \cup$ {end of p}
  $nc \leftarrow$TRYCONNECTNEWNODES(Γ, $newNodes$)
  if $nc = 0$ **then**
    TRYCONNECTTOROADMAP(Γ,$newNodes$)
function TRYCONNECTNEWNODES(Γ, $nodes$)
  for $\mathbf{q}_1, \mathbf{q}_2$ in $nodes$, $\mathbf{q}_1 \neq \mathbf{q}_2$ **do**
    $s_1 \leftarrow$ STATE($\mathbf{q}_1$); $s_2 \leftarrow$ STATE($\mathbf{q}_2$)
    $p \leftarrow$ CONNECT($\mathbf{q}_1, s_1, \mathbf{q}_2, s_2$)
    if p **then**
      Γ.ADDEDGE($\mathbf{q}_1$,$\mathbf{q}_2$,p)
function TRYCONNECTTOROADMAP(Γ,$nodes$)
  for $\mathbf{q}_1$ in $nodes$ **do**
    $s_1 \leftarrow$ STATE($\mathbf{q}_1$)
    for cc in connected components of Γ **do**
      if $\mathbf{q}_1 \notin cc$ **then**
        $near \leftarrow K$ nearest neighbors of $\mathbf{q}_1$ in cc
        for $\mathbf{q}_2$ in $near$ **do**
          $s_2 \leftarrow$ STATE($\mathbf{q}_2$)
          $p \leftarrow$ CONNECT($\mathbf{q}_1, s_1, \mathbf{q}_2, s_2$)
          if p **then** Γ.ADDEDGE($\mathbf{q}_1, \mathbf{q}_2$)
function EXTEND(s, $\mathbf{q}_{near}$, $\mathbf{q}_{rand}$)
  $solver \leftarrow$SOLVERBYSUBSTITUTION
  $\mathcal{T} \leftarrow$ random edge getting out of s
  $g \leftarrow$ state $\mathcal{T}$ points to
  for c in g.CONSTRAINTS( ) **do**
    $solver$.ADDCONSTRAINT($c(\mathbf{q}) = 0$)
  for c in $\mathcal{T}$.CONSTRAINTS( ) **do**
    $solver$.ADDCONSTRAINT($c(\mathbf{q}) = c(\mathbf{q}_{near})$)
  $\mathbf{q}_{target} \leftarrow solver$.SOLVE($\mathbf{q}_{rand}$)
  if $\mathbf{q}_{target}$ **then**
    $p \leftarrow$ linear interpolation from $\mathbf{q}_{near}$ to $\mathbf{q}_{target}$
    p.ADDCONSTRAINTS($\mathcal{T}$.CONSTRAINTS())
  if p collision-free and continuous **then return** p
  else return collision-free continuous portion of p starting at $\mathbf{q}_{near}$

Algorithm 5 Function CONNECT of M-RRT algorithm

function CONNECT($\mathbf{q}_1, s_1, \mathbf{q}_2, s_2$)
  parameter $\epsilon > 0$
  $p \leftarrow$ linear interpolation from $\mathbf{q}_1$ to $\mathbf{q}_2$
  $\mathcal{T} \leftarrow$TRANSITION(s_1,s_2)
  if not $\mathcal{T}$ **then return** $\emptyset$
  for c in $\mathcal{T}$.CONSTRAINTS( ) **do**
    if $\|c(\mathbf{q}_2) - c(\mathbf{q}_1)\| \geq \epsilon$ **then return** $\emptyset$
    else
      p.ADDCONSTRAINT($c(\mathbf{q}) = c(\mathbf{q}_1)$)
  if p in collision **then return** $\emptyset$
  return p

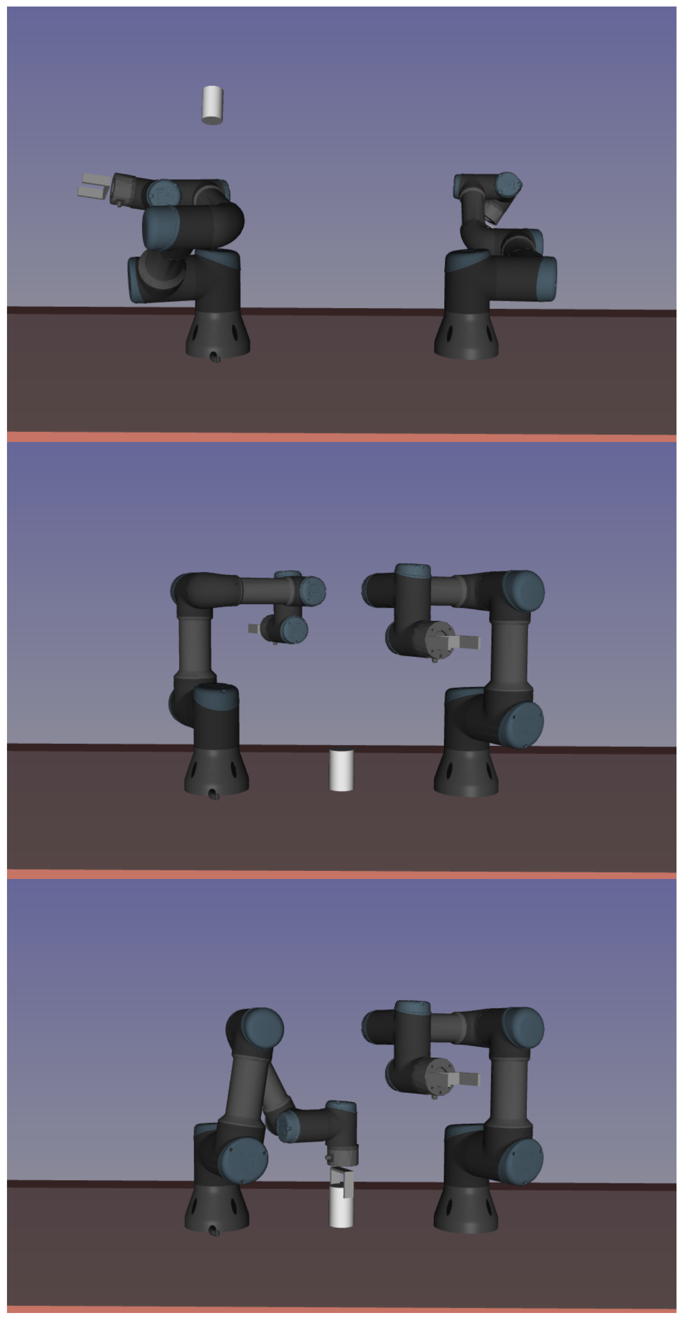

Fig. 3. Example of extension along a transition of the constraint graph. Top $\mathbf{q}_{rand}$, middle $\mathbf{q}_{near}$, bottom $\mathbf{q}_{target}$.

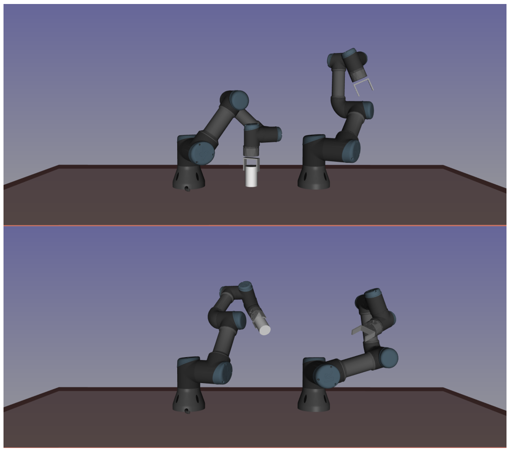

Fig. 4. Method CONNECT applied to two configurations.

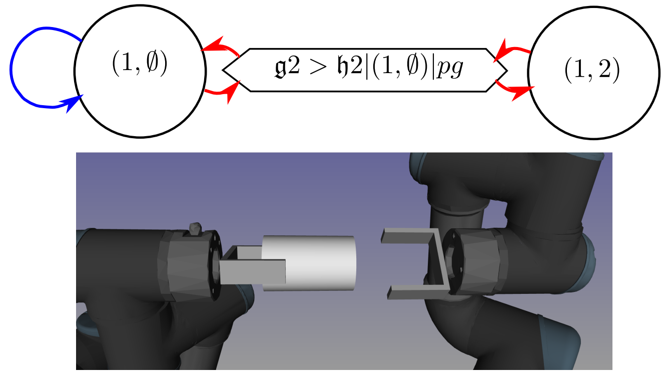

Fig. 5. Along a transition where an object already grasped is grasped a second time, an intermediate waypoint state called *pregrasp* (pg) is added. This intermediate state is represented by an hexagonal box. $\mathfrak{g}2 > \mathfrak{h}2|(1, \emptyset)$ means that gripper 2 is going to grasp handle 2 from the state where gripper 1 grasps handle 1. The constraints associated to this waypoint state are those of the state with the least active grasps (here $(1, \emptyset)$) and the pregrasp constraint corresponding to the new grasp (here $pregrasp_{22}$). The transition constraints are the same for all transitions (in red) and identical to the loop transition of the state with the least active grasps (in blue: here $grasp_{11}$ and $grasp_{11}/comp$).

robots and a cylinder with two handles. The picture at the top displays $\mathbf{q}_{rand}$. The picture in the middle displays $\mathbf{q}_{near}$ that belongs to state $(\emptyset, \emptyset)$. The transition that is randomly selected (Algorithm 4, line 33) is $(\emptyset, \emptyset) \to (1, \emptyset)$, meaning that robot 1 will try to grasp handle 1. According to tables II and III, the transition constraints are $(place_1, place_1/comp)$. The first one is of type (16), the second of type (18) and is parameterized: the right hand side uniquely defines the contact surfaces and the position of the object on the contact surface. $\mathbf{q}_{target}$ is obtained by projecting $\mathbf{q}_{rand}$ onto the manifold defined by the following constraints (Algorithm 4, lines 35–39):

- $place_1$, $place_1/comp$ that belong to the transition,
- $grasp_{11}$ that belongs to the goal state.

According to Section V-C, the first two constraints can be replaced by an explicit constraint: the position of the object can be derived from the right hand side of $place_1/comp$ that is initialized with configuration $\mathbf{q}_{near}$.

After substitution, the set of constraints is reduced to an implicit constraint on the configuration variables of robot 1 (6 variables). The solution found by the solver, $\mathbf{q}_{target}$ (line 39) is displayed in Figure 3 at the bottom. Notice that as expected, the position of the object is the same in $\mathbf{q}_{target}$ as in $\mathbf{q}_{near}$.

The path returned by function EXTEND is the linear interpolation between $\mathbf{q}_{near}$ and $\mathbf{q}_{target}$ constrained with $place_1$, $place_1/comp$ with right hand side initialized with $\mathbf{q}_{near}$. As explained earlier, this constraint is replaced by an explicit constraint. Let us notice that the linear interpolation already satisfies the constraint, but this is not always the case.

If the latter path is in collision, the collision-free part of the path starting at $\mathbf{q}_{near}$ is returned.

Figure 4 illustrates method CONNECT applied to two configurations $\mathbf{q}_1$ (top) and $\mathbf{q}_2$ (bottom). Both configurations belong to state $(1, \emptyset)$[4]. The transition between those states $(1, \emptyset) \to (1, \emptyset)$ contains the following constraints (tables II and III):

- $grasp_{11}/comp$, $grasp_{11}$.

Method CONNECT checks that the right hand side of $grasp_{11}/comp$ is the same for $\mathbf{q}_1$ and $\mathbf{q}_2$, up to the error threshold (Algorithm 4, line 51). From a geometrical point of view, this means that the orientation of the cylinder along its axis, with respect to the gripper is the same in both configurations. Let us recall that the right hand side of $grasp_{11}$ is 0. If the condition is satisfied, the method builds the linear interpolation between $\mathbf{q}_1$ and $\mathbf{q}_2$ with the explicit constraint equivalent to $\{grasp_{11}/comp,\ grasp_{11}\}$ and returns this path if it is collision-free.

[4]Note that $\mathbf{q}_1$ is at the intersection between states $(\emptyset, \emptyset)$ and $(1, \emptyset)$.

C. Waypoint transitions

By definition, a prehensile manipulation motion contains configurations that are in contact:

- between gripper and object during grasp,
- between object and contact surface when the object lies in a stable pose.

Contacts are difficult to handle using classical collision detection libraries and are often considered as collisions. To overcome this issue, we keep the gripper open during grasp, and objects slightly above contact surfaces in stable poses.

However even with these simple tricks, solution paths to a manipulation problem need to come close to collision, raising the well-known issue of narrow passages.

To cope with this, we define intermediate states in the constraint graph called waypoint states. These states are inserted between the regular states of the constraint graph. They require some prior definitions.

Definition 11: **pregrasp** A *pregrasp* is a numerical constraint h over $\mathcal{C}$, defined by

- a gripper $\mathfrak{g}$,
- a handle $\mathfrak{h}$.
- a non-negative real number Δ.

Let $\bar{h}$ be the smooth mapping from $\mathcal{C}$ to $\mathbb{R}^6$ defined by

$$\bar{h}(\mathbf{q}) = \log_{\mathbb{R}^3 \times SO(3)} \left(\mathfrak{g}^{-1}(\mathbf{q})\mathfrak{h}(\mathbf{q})\right) - (\Delta\, 0\, 0\, 0\, 0\, 0)^T. \quad (20)$$

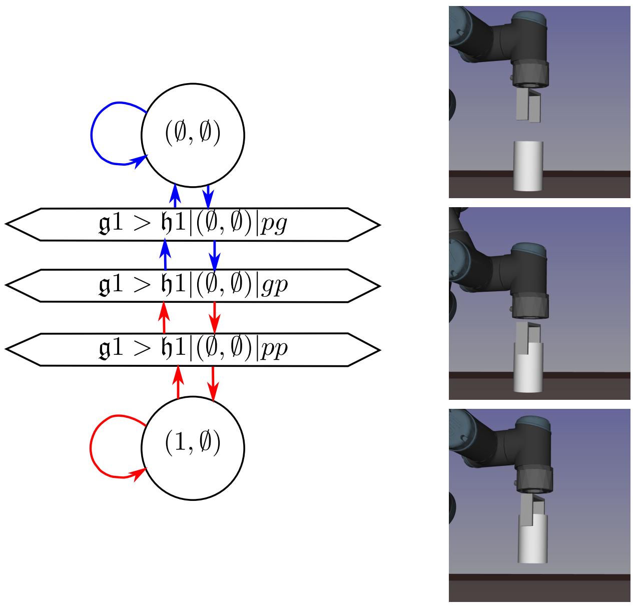

Fig. 6. Along a transition where an object in placement is grasped by a gripper, we add three waypoint states called *pregrasp* (pg) where the gripper is above the object, *grasp-placement* (gp) where the object is grasped but still in placement and *preplacement* (pp) where the object is grasped above the contact surface. All transitions between the state with the least active grasps and the waypoint *gp* have the same constraints as the loop transition of the state with the least active grasps (here: $place_1$ and $place_1/comp$ in blue). All transitions between the waypoint state *gp* and the state with the most active grasps have the same constraints as the loop transition of the state with the most active grasps (here: $grasp_{11}$ and $grasp_{11}/comp$ in red).

$h(\mathbf{q})$ is obtained by extracting from $\bar{h}$ the components the values of which are `true` in the handle flag.

Note that when this constraint is satisfied, the handle is translated along x axis over a distance Δ compared to a configuration satisfying the grasp constraint. The value of Δ depends on the geometry of the gripper and object. Clearance values are associated to the handle: cl_o and to the gripper: cl_g. Δ is defined as $cl_o + cl_g$. The clearance parameters are part of the definition of the gripper and handle and are stored in SRDF files.

Definition 12: **preplacement** A preplacement is a numerical constraint h over $\mathcal{C}$, defined by

- $(o_i)_{i\in I}$ a set of convex polygons attached to an object,
- $(f_j)_{j\in J}$ a set of convex polygons attached to the environment or to a mobile part of a robot that can receive objects (mobile robot for example),
- a non-negative real number Δ.

with the same notation as in Section V-B, we define i and j as the indices that minimize $d(f_j, o_i)$ (Equation (14)), and

$$\bar{h}(\mathbf{q}) = \log_{\mathbb{R}^3\times SO(3)}\left(f_j(\mathbf{q})^{-1}o_i(\mathbf{q})\right) + (\Delta\, 0\, 0\, 0\, 0\, 0)^T \quad (21)$$

The left hand side of the preplacement constraint is defined by Equation (16).

Note that when this constraint is satisfied, the object is translated over a distance Δ along the normal to the contact surface.

We denote by

- $pregrasp_{ij}$ $i \in \{1, \cdots, n_{\mathfrak{g}}\}$ $j \in \{1, \cdots, n_{\mathfrak{h}}\}$ the pregrasp constraint of handle j by gripper i,

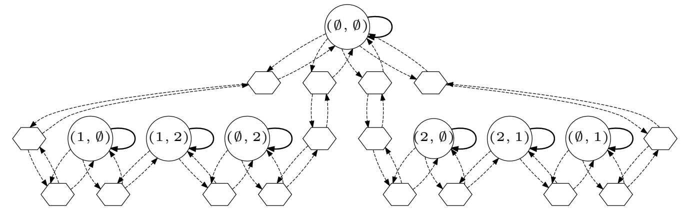

Fig. 7. Structure of the constraint graph corresponding to the system in Section V-D2 after inserting waypoint transitions. Waypoint transitions starting from/going to $(\emptyset, \emptyset)$ contain three waypoint states. All other waypoint transitions contain one waypoint state.

- $preplace_i$ $i \in \{1, \cdots n_o\}$ the preplacement constraint of object i.

We replace the transitions of the constraint graph defined in Section V-D by a sequence of intermediate states and transitions: given Definition 9, if two states $\mathcal{S}_1$ and $\mathcal{S}_2$ are adjacent to each other, one of them contains an additional grasp with respect to the other. Without loss of generality, consider that $\mathcal{S}_2$ contains the additional grasp $gr(\mathfrak{g}_i, \mathfrak{h}_j)$, $i \in \{1, \cdots n_{\mathfrak{g}}\}$, $j \in \{1, \cdots n_{\mathfrak{h}}\}$. Let us denote by o the object to which handle $\mathfrak{h}_j$ belongs. Then either

1) o is already grasped in state $\mathcal{S}_1$, or
2) o is in placement in state $\mathcal{S}_1$.

In case 1, we replace the transitions between $\mathcal{S}_1$ and $\mathcal{S}_2$ by an additional waypoint state and four waypoint transitions as explained in Figure 5.

In case 2, we replace the transitions between $\mathcal{S}_1$ and $\mathcal{S}_2$ by three additional waypoint states and eight waypoint transitions as explained in Figure 6.

1) Construction of a path along a waypoint transition

Function EXTEND in Algorithm 4 builds a path along a transition from an initial configuration by projecting the configuration onto the submanifold defined by the goal state constraints and by the transition constraints. The right hand side of the transition constraint is first initialized with the initial configuration.

A waypoint transition builds a path by defining a sequence of configurations that belong to the intermediate waypoint states, each configuration being obtained by projecting the previous configuration onto the corresponding manifold. Figure 8 proposes an example of extension along edge $(\emptyset, \emptyset) \rightarrow (1, \emptyset)$ from configuration $\mathbf{q}_{near}$ (Figure 3 middle). The edge contains three waypoints. The random configuration $\mathbf{q}_{rand}$ is displayed in Figure 3, top. Table IV lists the waypoint configurations that are produced when extending $\mathbf{q}_{near}$ toward $\mathbf{q}_{rand}$, and the constraints applied to compute these configurations.

2) Implementation

From an implementation point of view, Class `WaypointEdge` derives from class `Edge`. The waypoint configurations are computed by method `generateTargetConfig` that is specialized in Class `WaypointEdge`.

Note that waypoint states are internal to waypoint edges and thus not known by the constraint graph when determining to which state a configuration belongs (Algorithm 4 lines 17

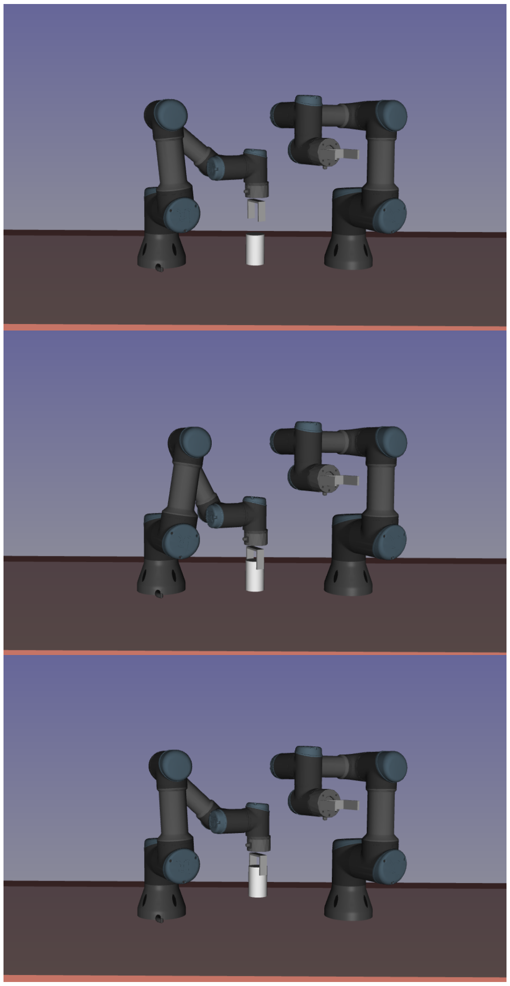

Fig. 8. Example of extension along the waypoint transition between states $(\emptyset, \emptyset)$ and $(1, \emptyset)$. Each picture represents a waypoint. The last waypoint is in state $(1, \emptyset)$. $\mathbf{q}_{near}$ and $\mathbf{q}_{rand}$ are the same as in Figure 3.

$\mathbf{q}_{near}$	**in state** $(\emptyset, \emptyset)$
$\mathbf{q}_1$	**waypoint state** $\mathfrak{g}1 > \mathfrak{h}1\|(\emptyset, \emptyset)\|pg$ **in state** $(\emptyset, \emptyset)$ **constraints** $place_1(\mathbf{q}) = 0$ $place_1/comp(\mathbf{q}) = place_1/comp(\mathbf{q}_{near})$ $pregrasp_{11}(\mathbf{q}) = 0$ **solver initialized with** $\mathbf{q}_{rand}$
$\mathbf{q}_2$	**waypoint state** $\mathfrak{g}1 > \mathfrak{h}1\|(\emptyset, \emptyset)\|gp$ **in states** $(\emptyset, \emptyset)$ and $(1, \emptyset)$ **constraints** $place_1(\mathbf{q}) = 0$ $place_1/comp(\mathbf{q}) = place_1/comp(\mathbf{q}_{near})$ $grasp_{11}(\mathbf{q}) = 0$ **solver initialized with** $\mathbf{q}_1$
$\mathbf{q}_{target}$	**waypoint state** $\mathfrak{g}1 > \mathfrak{h}1\|(\emptyset, \emptyset)\|pp$ **in state** $(1, \emptyset)$ **constraints** $grasp_{11}(\mathbf{q}) = 0$ $grasp_{11}/comp(\mathbf{q}) = grasp_{11}/comp(\mathbf{q}_2)$ $preplace_1(\mathbf{q}) = 0$ **solver initialized with** $\mathbf{q}_2$

TABLE IV
WAYPOINT CONFIGURATIONS COMPUTED ALONG EDGE $(\emptyset, \emptyset) \rightarrow (1, \emptyset)$. THE RESULTING PATH BETWEEN $\mathbf{q}_{near}$ AND $\mathbf{q}_{target}$ IS A CONCATENATION OF CONSTRAINED LINEAR INTERPOLATION. CONSTRAINTS APPLIED BETWEEN A WAYPOINT AND ITS PREDECESSOR ARE SHOWN IN BLUE.

and 23) and when visiting the states of the constraint graph (Algorithm 4 line 8).

VII. HUMANOID PATH PLANNER

In this section, we describe in greater details the software platform Humanoid Path Planner that implements the concepts and algorithms of the previous sections.

Humanoid Path Planner is a collection of standard software packages that depend on each other. The main packages are the following:

- `hpp-fcl` a modified version of `fcl`. The main additional features are:
  - computation of a lower bound of the distance when testing collision between two objects. This is required for continuous collision detection,
  - security margins in collision checking,
- `pinocchio` [60] a library computing forward kinematics and dynamics for multi-body kinematic chains,
- `hpp-constraints` a library that implements numerical constraints and solvers,
- `hpp-core` a library that implements most of the concepts relative to motion planning. The main features are:
  - abstraction of paths in configuration spaces and some implementations,
  - abstraction of path planning and path optimization and some implementations,
  - abstraction of steering methods and some implementations,
  - roadmaps,
  - validation of configurations and paths, notice that this includes an implementation of continuous collision checking first proposed by Schwarzer *et al* [61].
- `hpp-manipulation` a library that implements manipulation problems and manipulation planning with
  - composite kinematic chains composed of the robots and objects,
  - the constraint graph,
  - M-RRT algorithm,
- `hpp-manipulation-urdf` an extension of the SRDF parser to retrieve information relative to objects, like the definition of grippers, handles, and contact surfaces.

An HPP session consists of a standalone executable `hppcorbaserver` that implements CORBA services. These services can be extended via a plugin system. The application can then be controlled with python scripts or C++ code. CORBA clients are provided in python and C++. The packages implementing CORBA clients and servers are

- `hpp-corbaserver` for canonical path planning problems, and
- `hpp-manipulation-corba` for manipulation problems. This package also provides an implementation of the automatic constraint graph construction in python.

The environment used for path planning as well as the paths computed can be displayed using `gepetto-gui` through packages

- `gepetto-viewer`,

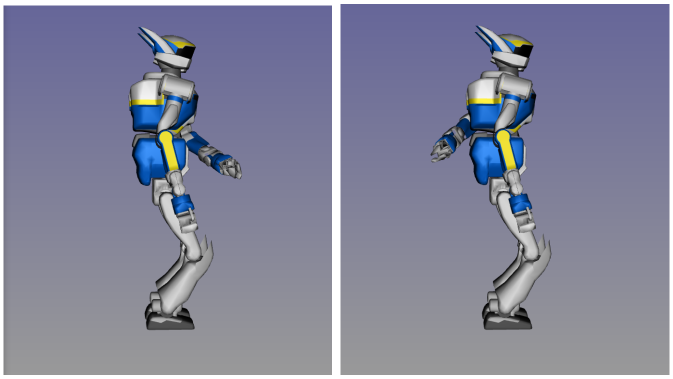

Fig. 9. Constrained motion planning for HRP-2 humanoid robot sliding on the ground in quasi-static equilibrium: the feet should stay horizontal with a fixed relative position and the center of mass should project between the feet. The initial configuration is shown on the left. The goal configuration is shown on the right. The algorithm is a constrained RRT close to the one described in Dalibard *et al* [16]

	min	max	mean	std dev
time (s)	0.03	11.64	1.32	2.55
nodes	4	136	32.40	30.72

TABLE V
EXPERIMENTAL RESULTS FOR HRP-2 SLIDING ON THE GROUND (36 DEGREES OF FREEDOM): TIME OF COMPUTATION AND NUMBER OF NODES.

- `gepetto-viewer-corba`, and
- `hpp-gepetto-viewer`.

A. Virtual machine

A virtual docker image can be downloaded to run, test and replicate the examples described in the next sections. An archive is provided with this paper. Decompress the archive and follow instructions in the README file.

B. Experimental results

In this section, we report on several experimental results obtained with HPP software on constrained motion planning and on manipulation planning problems. The raw data can be found in `hpp_benchmark` package. Here we only present a few test cases. The benchmarks are run 20 times each on an Intel Core i7 at 2.60 GHz, with 32 Gigabytes of RAM and 9 Megabytes of cache memory. For each test case, we report the minimum, maximum, mean and standard deviation of the time of computation on the one hand, and of the number of nodes in the roadmap built to solve the problem, on the other.

1) Constrained motion planning

One test case concerns constrained motion planning. The robot is an HRP-2 humanoid robot in quasi-static equilibrium that can slide on the ground (Figure 9). This type of motion can be post-processed into a walking motion using the method described in Dalibard *et al* [62]. The results are displayed in Table V.

2) Manipulation planning

In this section, we present some experimental results of manipulation planning problems obtained with M-RRT algorithm described in Section VI.

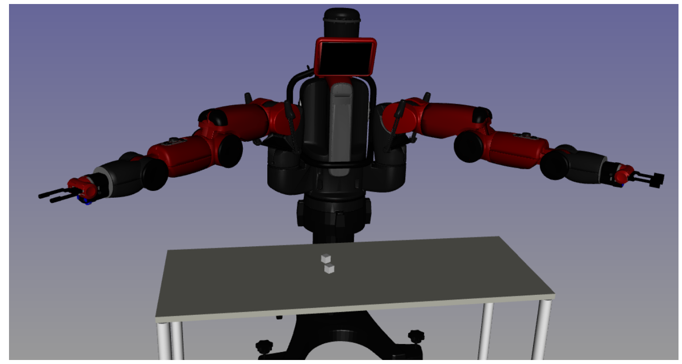

Fig. 10. Manipulation problem with Baxter robot manipulating two small boxes. The robot is requested to swap the boxes.

	min	max	mean	std dev
time (s)	0.84	15.60	7.60	4.48
nodes	23	375	176.15	108.54

TABLE VI
EXPERIMENTAL RESULTS FOR BAXTER ROBOT MANIPULATING TWO BOXES ON A TABLE (31 DEGREES OF FREEDOM).

The first test case features robot Baxter manipulating two boxes on a table (see Figure 10). The boxes are swapped between the initial and final configurations. The robot has two grippers and each box is equipped with a handle. Thus the constraint graph contains seven nodes. The experimental results are displayed in Table VI.

The second test case features robot PR-2 manipulating a box on a table. The robot is requested to flip the box upside down from an initial pose to a goal pose as represented in Figure 11. The robot is equipped with two grippers and the box with two handles. The constraint graph contains seven nodes. Table VII shows the experimental results.

The third test case features Humanoid Robot Romeo manipulating a placard. The robot is requested to rotate the placard by 180 degrees. It is equipped with two grippers and the placard with two handles. Each handle is associated to a single gripper. The number of states of the constraint graph is thus three.

In the three previous test cases, the constraint graph was automatically built by Algorithm 2. If the number of grippers and handles increases, the number of states in the constraint

	min	max	mean	std dev
time (s)	0.92	9.62	3.30	2.47
nodes	6	111	32.90	31.63

TABLE VII
EXPERIMENTAL RESULTS FOR PR-2 ROBOT MANIPULATING A BOX ON A TABLE (39 DEGREES OF FREEDOM).

	min	max	mean	std dev
time (s)	4.64	554.18	151.49	158.64
nodes	27	2448	610.45	662.83

TABLE VIII
EXPERIMENTAL RESULTS FOR ROMEO ROBOT MANIPULATING A PLACARD (67 DEGREES OF FREEDOM).

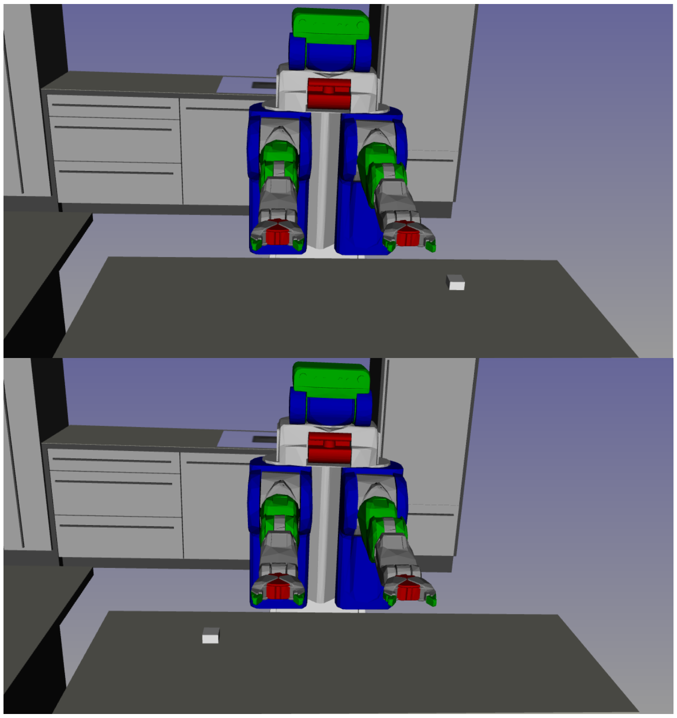

Fig. 11. Manipulation planning problem with PR-2 robot manipulating a box. The robot needs to flip the box upside down from an initial pose (top) to a goal pose (bottom).

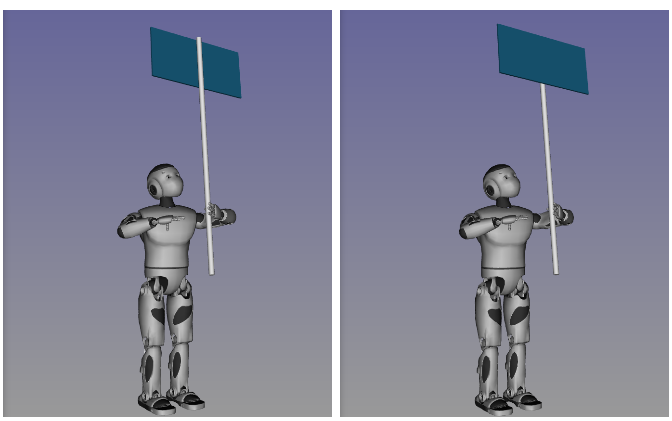

Fig. 12. Manipulation planning problem with Romeo robot manipulating a placard. The robot needs to flip the placard by 180 degrees from an initial pose (left) to a goal pose (right), keeping balance.

	min	max	mean	std dev
time (s)	0.20	214.22	17.11	45.84
nodes	10	39	14.70	6.48

TABLE IX
EXPERIMENTAL RESULTS FOR CONSTRUCTION SET ASSEMBLY (36 DEGREES OF FREEDOM).

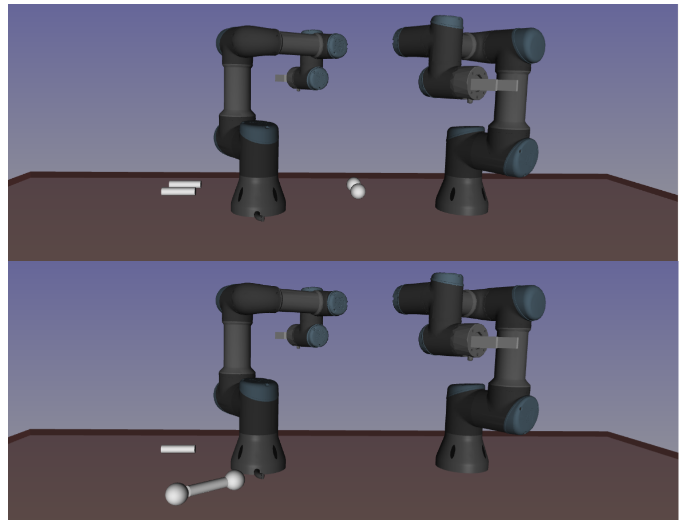

Fig. 13. Construction set: two robots are requested to assemble magnetic spheres on a cylinder from an initial configuration (top) to a goal state (bottom).

	min	max	mean	std dev
with waypoints				
time (s)	0.01	0.49	0.12	0.14
nodes	4	30	9.75	7.26
without waypoints				
time (s)	4.15	73.53	26.24	17.07
nodes	97	1609	711.55	407.98

TABLE X
UR-5 MANIPULATING A BALL WITH AND WITHOUT WAYPOINT TRANSITIONS

graph may increase very quickly. However using python bindings, it is possible to define constraint graphs with only the necessary states. We now present a test case that illustrates this possibility. The system is depicted in Figure 13.

In this example, an operator provides the sequence of actions (transitions) the system needs to perform:

1) robot 1 grasps sphere 1,
2) robot 2 grasps cylinder 1,
3) robot 1 sticks sphere 1 to cylinder 1,
4) robot 1 releases sphere 1,
5) robot 1 grasps sphere 2,
6) robot 1 sticks sphere 2 to cylinder 1,
7) robot 1 releases sphere 2,
8) robot 2 puts cylinder 1 on the ground.

From this sequence of actions, the sequence of states visited is computed and only those states (nine in total) are built in the constraint graph. Then, a sequence of subgoals in the successive states is computed, in such a way that each subgoal is accessible by the previous one (on the same leaf of the corresponding transition foliation). The subgoals are then linked by running a constrained visibility PRM algorithm [63] on each leaf. The python code can be found at github.com.

Figure 13 displays the initial configuration and the goal state. Table IX shows the experimental results.

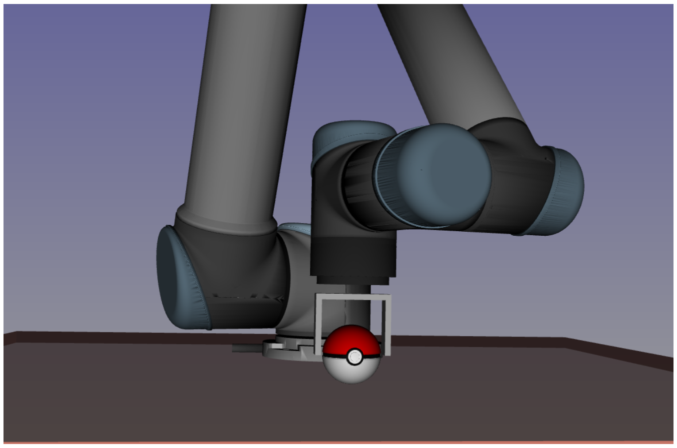

Fig. 14. UR-5 robot manipulating a ball lying on a plane. The robot is requested to pick the ball and place it a few centimeters aside.

3) Influence of waypoint transitions

All the previous experimental results have been obtained using waypoint transitions as described in Section VI-C. We now empirically show the positive effect of waypoints on the efficiency of manipulation planning. To do that, we run 20 times Algorithm M-RRT on the same problem with and without waypoint transitions. The problem is defined by a UR-5 robot manipulating a ball as shown in Figure 14. The results are reported in Table X. We can notice in this example, that waypoint transitions decrease the computation time and the number of nodes by two orders of magnitude. This is because in grasp configurations, the gripper is very close to the object and only a small part of the approaching directions of the gripper toward the object leads to collision-free paths. On the contrary waypoint states are away from obstacles and easier to reach. The transition between the pregrasp waypoint and the grasp $\cap$ placement waypoint is almost always collision-free.

4) Analysis

The experimental results show that M-RRT is able to solve a variety of manipulation problems including that of a legged robot in quasi-static equilibrium. No parameter tuning is required between the different problems. All parameters are set to a default value for all test cases.

As in any random motion planning method, we observe a large standard deviation between the 20 runs of each test case, for the number of nodes as well as for the time of computation.

We have also observed experimentally that the efficiency of M-RRT decreases when

1) the number of states to visit to solve a problem increases,
2) the number of foliated states increases.

Thus, M-RRT is not able to solve the construction set problem within a reasonable amount of time. However, to our knowledge it is the only algorithm in the literature capable of solving a variety of problems as large as those presented in this section.

VIII. Conclusion and Future Work

This paper presents a software platform aimed at prototyping and solving a large number of prehensile manipulation planning problems. The platform provides an original algorithm M-RRT that is an extension of RRT exploring the leaves of the foliations defined by the manipulation constraints. The automatic insertion of waypoint states makes the resolution more efficient and the resulting paths more natural.

It is the authors' opinion that this platform is perfect for researchers who want to develop and benchmark new manipulation planning algorithms. Note that some of the ongoing work in humanoid locomotion [64] is based on HPP.

To show the maturity of the project, we provide a docker image embarking the software.

As a future work, we aim at working on general manipulation planning algorithms that can handle use cases as diverse as those proposed in the benchmark section. A good candidate is a generalization of RMR* [58]. Also we intend to focus on manipulation path optimization since paths computed by random algorithms are too long to be applied to real robots as such. Finally, we would like to generalize the reduction property proposed by Siméon *et al* [30] . The constraint graph representation is a perfect tool for that.

Acknowledgment

This work has been partially supported by Airbus S.A.S. within the framework of the common laboratory Rob4Fam.

References

[1] C. Eppner, S. Höfer, R. Jonschkowski, R. Martín-Martín, A. Sieverling, V. Wall, and O. Brock, "Four aspects of building robotic systems: lessons from the amazon picking challenge 2015," *Autonomous Robots*, vol. 42, no. 7, pp. 1459–1475, October 2018, https://link.springer.com/article/10.1007/s10514-018-9761-2. [Online]. Available: https://link.springer.com/article/10.1007/s10514-018-9761-2
[2] J. Mirabel, S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiraux, "HPP: a new software for constrained motion planning," in *International Conference on Intelligent Robots and Systems (IROS 2016)*, Daejeon, South Korea, Oct. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01290850
[3] J. Mirabel and F. Lamiraux, "Handling implicit and explicit constraints in manipulation planning," in *Robotics: Science and Systems 2018*, Pittsburg, United States, Jun. 2018, p. 9p. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01804774
[4] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm," in *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'05. AAAI Press, 2005, p. 262–271.
[5] F. Lamiraux and L. E. Kavraki, "Planning paths for elastic objects under manipulation constraints," *The International Journal of Robotics Research*, vol. 20, no. 3, pp. 188–208, 2001. [Online]. Available: https://doi.org/10.1177/02783640122067354
[6] O. Roussel, P. Fernbach, and M. Taïx, "Motion Planning for an Elastic Rod using Contacts," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 2, pp. 670–683, Apr. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01954894
[7] H. Choset, "Coverage for robotics – a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1, pp. 113–126, 2001.
[8] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258 – 1276, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092188901300167X
[9] L. J. Guibas, J.-C. Latombe, S. M. Lavalle, D. Lin, and R. Motwani, "A visibility-based pursuit-evasion problem," *International Journal of Computational Geometry & Applications*, vol. 09, no. 04n05, pp. 471–493, 1999. [Online]. Available: https://doi.org/10.1142/S0218195999000273

[10] J. T. Schwartz and M. Sharir, "On the "piano movers" problem. ii. general techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*, vol. 4, no. 3, pp. 298 – 351, 1983. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0196885883900143

[11] J. Canny, "The complexity of robot motion planning," Ph.D. dissertation, Massachuset Institute of Technology, 1983.

[12] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for fast path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[13] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry and Applications*, vol. 9, no. 4–5, pp. 495–512, 1999.

[14] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *International Conference on Robotics and Automation*. San Francisco, (USA): IEEE, Apr. 2000, pp. 473–479.

[15] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: http://arxiv.org/abs/1105.1186

[16] S. Dalibard, A. Nakhaei, F. Lamiraux, and J.-P. Laumond, "Whole-Body Task Planning for a Humanoid Robot: a Way to Integrate Collision Avoidance," in *IEEE International Conference on Humanoid Robots*, Paris, France, Dec. 2009, pp. 1–6. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00450897

[17] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011. [Online]. Available: https://doi.org/10.1177/0278364910396389

[18] J. Cortés, T. Simeon, and J.-P. Laumond, "A Random Loop Generator for Planning the Motions of Closed Kinematic Chains using PRM Methods," in *2002 IEEE International Conference on Robotics and Automation (ICRA 2002)*. Washington, United States: IEEE, May 2002, pp. 2141–2146. [Online]. Available: https://hal.laas.fr/hal-01988698

[19] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 105–117, 2013.

[20] B. Kim, T. T. Um, C. Suh, and F. Park, "Tangent bundle rrt: A randomized algorithm for constrained motion planning," *Robotica*, vol. 34, pp. 202–225, 2016.

[21] M. Cefalo and G. Oriolo, "A general framework for task-constrained motion planning with moving obstacles," *Robotica*, vol. 37, pp. 575–598, 2019. [Online]. Available: http://www.dis.uniroma1.it/~labrob/pub/papers/Robotica19.pdf

[22] Z. Kingston, M. Moll, and L. E. Kavraki, "Exploring implicit spaces for constrained sampling-based planning," *The International Journal of Robotics Research*, vol. 38, no. 10-11, pp. 1151–1178, 2019. [Online]. Available: https://doi.org/10.1177/0278364919868530

[23] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *IEEE Transactions On Robotics And Automation*, vol. 14, no. 4, pp. 549–565, August 1998.

[24] J. Z. Woodruff and K. M. Lynch, "Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4066–4073.

[25] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *The Int. Journal of Robot. Research (IJRR)*, vol. 25, no. 4, pp. 317–342, Apr. 2006. [Online]. Available: http://dx.doi.org/10.1177/0278364906063979

[26] S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar, "Generation of whole-body optimal dynamic multi-contact motions," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1104–1119, 2013. [Online]. Available: https://doi.org/10.1177/0278364913478990

[27] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 586–601, 2018.

[28] R. Alami, T. Siméon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks (3). the case of discrete placements and grasps," LAAS-CNRS, Tech. Rep., 1989.

[29] M. Vendittelli, J.-P. Laumond, and B. Mishra, "Decidability in robot manipulation planning," arXiv.org, Tech. Rep., 2018. [Online]. Available: https://arxiv.org/abs/1811.03581

[30] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *International Journal of Robotics Research*, vol. 23, no. 7/8, July 2004.

[31] G. Wilfong, "Motion planning in the presence of movable obstacles," in *Proceedings of the fourth annual symposium on Computational geometry*. ACM, 1988, pp. 279–288.

[32] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1295–1307, 2008.

[33] J. 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*, vol. 2. IEEE, 2004, pp. 1962–1967.

[34] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "Sample-based methods for factored task and motion planning," in *Robotics: Science and Systems (RSS)*, 2017. [Online]. Available: http://lis.csail.mit.edu/pubs/garrett-rss17.pdf

[35] A. Krontiris and K. Bekris, "Dealing with difficult instances of object rearrangement," in *Robotics Science and Systems*, Roma, Italy, 2015.

[36] P. Lertkultanon and Q.-C. Pham, "A single-query manipulation planner," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 198–205, 2015.

[37] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 3327–3332.

[38] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[39] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars, "An effective framework for path planning amidst movable obstacles," in *Algorithmic Foundation of Robotics VII*. Springer, 2008, pp. 87–102.

[40] S. Cambon, R. Alami, and F. Gravot, "A Hybrid Approach to Intricate Motion, Manipulation and Task Planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, Jan. 2009. [Online]. Available: https://hal.laas.fr/hal-01976081

[41] L. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *International Journal on Robotics Research*, vol. 32, no. 9–10, pp. 1194—-1227, 2013. [Online]. Available: https://lis.csail.mit.edu/pubs/tlp/IJRRBelFinal.pdf

[42] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 3684–3691. [Online]. Available: https://lis.csail.mit.edu/pubs/tlpk-iros14.pdf

[43] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proc. of Robotics: Science and Systems (R:SS 2018)*, 2018, *Best Paper Award*.

[44] M. Gharbi, J. Cortés, , and T. Siméon, "Roadmap composition for multi-arm systems path planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Saint-Louis, USA, 2009.

[45] 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*, Hongkong, China, 2014, pp. 928—934.

[46] A. Dobson and K. Bekris, "Planning representations and algorithms for prehensile multi-arm manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015.

[47] Z. Xian, P. Lertkultanon, and Q. Pham, "Closed-chain manipulation of large objects by multi-arm robotic systems," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 1832–1839, 2017.

[48] P. S. Schmitt, F. Wirnshofer, K. M. Wurm, G. V. Wichert, and W. Burgard, "Modeling and planning manipulation in dynamic environments," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019. [Online]. Available: http://ais.informatik.uni-freiburg.de/publications/papers/schmitt19icra.pdf

[49] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011. [Online]. Available: http://motion.pratt.duke.edu/papers/ijrr2011-MultiModal-preprint.pdf

[50] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[51] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010. [Online]. Available: http://www.programmingvision.com/rosen_diankov_thesis.pdf

[52] R. M. Murray, S. S. Sastry, and L. Zexiang, *A mathematical introduction to robotic manipulation*. CRC Press, 1994.
[53] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.
[54] "Backtracking line search." [Online]. Available: https://en.wikipedia.org/wiki/Backtracking_line_search
[55] K. Hauser, "Fast interpolation and time-optimization on implicit contact submanifolds," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.
[56] J. Mirabel and F. Lamiraux, "Manipulation planning: building paths on constrained manifolds," Jul. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01360409
[57] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, "A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: the Stack of Tasks," in *ICAR'09: International Conference on Advanced Robotics*, Munich, Germany, Jun. 2009, pp. 1–6. [Online]. Available: https://hal-lirmm.ccsd.cnrs.fr/lirmm-00796736
[58] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3426–3432. [Online]. Available: http://ais.informatik.uni-freiburg.de/publications/papers/schmitt17icra.pdf
[59] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.
[60] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*, 2019, pp. 614–619.
[61] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact collision checking of robot paths," in *Algorithmic Foundations of Robotics V, STAR 7*, J.-D. B. et al., Ed. Springer, 2004, pp. pp 25–41.
[62] S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J.-P. Laumond, "Dynamic Walking and Whole-Body Motion Planning for Humanoid Robots: an Integrated Approach," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. pp.1089–1103, Aug. 2013. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00654175
[63] T. Simeon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Journal of Advanced Robotics*, vol. 14, no. 6, pp. 477–494, 2000.
[64] S. Tonneau, A. D. Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, pp. 1–16, 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01267345/document

Florent Lamiraux graduated from the Ecole Polytechnique Paris in 1993. He received the Ph-D degree in Computer Science from the Institut National Polytechnique de Toulouse in 1997 for his research on Mobile Robots. Between 1997 and 1999, he worked at Rice University as a postdoctoral Research Associate on motion planning for deformable objects. Since 2005, he has been working in humanoid robots. He spent two years in AIST Tsukuba in Japan in 2008-2009. He is currently Directeur de Recherche at LAAS-CNRS. His research interests include manipulation planning and control for humanoid and industrial robots.

Joseph Mirabel graduated from the Ecole Polytechnique Paris and the Royal Institute of Technology, Stockholm in 2013. He received the Ph-D degree in Robotics from the Institut National Polytechnique de Toulouse in 2017 for his research on motion and manipulation planning. Between 2017 and 2021, he worked at LAAS-CNRS as a researcher on reactive manipulation planning and robot control with visual feedback. He recently joined Eureka Robotics as a Senior Scientist.