HPP: a new software framework for manipulation planning

Joseph Mirabel[1,2] and Florent Lamiraux[1,2]

***Abstract*— abstract**

I. INTRODUCTION

This paper presents an approach for solving generic Manipulation Planning problems. The method can compute manipulation paths for robots and movable objects in a static environment. It handles continuous grasps and object placements. The formulation of the problem, as described in section II-E, allows the algorithm to plan for non-monotone instances, where simultaneous manipulation can be required. Practical cases of such non-monotone instances are shown in section IV. The first contribution of this paper is the representation of Manipulation Planning as a generic *Constraint Graph*, an interface bridging task and motion planning. The second contribution is an algorithm able to solve Manipulation Planning problems with few objects, with possibly simultaneous manipulation of objects, and for highly constrained environments such as humanoid robots.

A. Manipulation planning and Navigation Among Movable Obstacles

Navigation Among Movable Obstacles (NAMO) is the class of Motion Planning problems where a robot navigates in an environment and is allowed to move objects out of its way. This problem is PSPACE-hard when the final positions of objects are specified and NP-hard otherwise [1]. To our best knowledge, there exists no practical planner able to solve generic instances of this problem. The limitations are mostly the number of objects and possible interactions between objects.

Manipulation Planning is similar to NAMO. The difference lies in the purpose. Manipulation Planning aims at using the robot to act on the environment whereas NAMO aims at navigating, i.e. finding a path for the robot. Manipulation Planning Among Movable Obstacles [2] is a combination of both. The robot must achieve a specific task (e.g. moving an object) and is allowed to move other obstacles. Our focus is on Manipulation Planning problems. The formulation is given in section II-E.

In spite of its complexity, practical solutions for subclasses of NAMO have arisen. The case of regrasping has been solved particularly efficiently [3] thanks to a reduction property. This property states that the search in the intersection of the set of valid grasp and valid placement can be done regardless of the manipulation rules. Though powerful for cases where an object has to be regrasped, this property does not reduce the inherent complexity of NAMO.

Monotone instances of NAMO, i.e. when there exists a solution where each object is moved at most once, has been the focus of many works. Most of the approaches tackling this problem [4], [2], [5] uses two-level planning methods. The first, high, level is a symbolic planner, sometimes called task planner [5]. The second, low, level is a motion planner [6]. While the first level deals with discrete elements, such as actions (“pick object”, “put down object”. . .), conditions (“object is on table”,. . .) and propositions [5], the second level deals with the geometry of the problem. The latter solves motion planning problems of a specific case of the combinatorial of end-effectors, objects and their possible placements. An interface between the two levels [5] is thus required.

Two-level approaches benefit from the existence of efficient solutions to discrete problems. In some cases, reasoning on movable objects allows an early detection of the order in which objects have to be moved [2].

However, a first drawback is the difficulty of providing information from the task planner to the motion planner and vice-versa. Motion planning problem are specified in a continuous uncountable set whereas symbolic actions, conditions and propositions represents motions or continuous subspaces of the configuration space.

A second and probably more important drawback is the lack of integration between both level. When the task planner requests the motion planner to solve a subproblem, it merely waits for a boolean answer, which has false negatives - practical motion planner cannot prove a problem is infeasible. Of course, in case of success, a path is returned. In other words, from a complex and time consuming search, the motion planner is merely returning a boolean and a sequence of configurations. Motion planner are usually not able to provide a good reason when they fail to find a path.

Other approaches have tried to bring motion planning at a higher level of abstraction. For this purpose, the configuration space is segmented into modes, and motion primitives allow transition between modes [7]. Similar approaches [8], [9] represent the possible actions in a graph. In contrast with multi-level approaches, they are not specifically reasoning about blocking objects.

Our method follows this idea of a better integration of the task and motion planner. We generalize the graph of possible actions [8] and apply it to wholebody robots. We are introducing a motion planner which has the following

*This work has been partially supported by the national PSPC-Romeo 2 project, has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement n 609206 and n 608849 and from the French agency ANR under grant agreement 13-CORD-002-01.

[1]CNRS, LAAS, 7 avenue du colonel Roche, F–31400 Toulouse, France
[2]Univ de Toulouse, LAAS, F–31400 Toulouse, France

properties:

- it understands symbolic facts, similar to [5], making it easy to integrate with a higher level planner, this is just a guess
- it provides useful information on the current state of the search, so that a task planner get more indication of what to try next, Emphasize the fact that it can try several "actions" at the same time.
- it does not reason on objects, so that it can handle problems which do not look, at first, like Manipulation Planning problems.

B. *Constrained planning*

Motion planning algorithms were first developed to solve problems such as the piano's mover problem [6]. Randomized planners succeeded in solving motion planning problems in high dimensional configuration space.

Constrained problems concern a more generic class of problems, where the previous solutions does not work. Examples of such problems are motion planning for humanoid robots [11] or closed chain kinematics [12].

It is a known fact that standard randomized algorithms like Randomly Exploring Random Trees (RRT) or Probabilistic Roadmap (PRM) fail to solve constrained problems, because of $\mathcal{CS}_{const}$, the zero volume submanifold satisfying the constraints [13], [6, chap. 7]. Constrained planning is an essential component of Manipulation Planning. When a robot is lifting an object, the end-effector and the object motion are constrained. Inverse kinematic does not allow to solve cases where the robot degrees of freedom (DOF) are correlated by stability constraints. Another example of constrained planning is the temporary closed kinematic chain created by two arm manipulators when they collaborate to achieve a task. Most methods [11], [12], [13] relies on the ability to generate configuration in $\mathcal{CS}_{const}$. See [14] for details of existing methods. In this paper, we focus on differentiable constraints and we use the gradient descent method developed in [11]. The method is similar to the First-Order Retraction algorithm which tends to have better results than other algorithms [14].

II. Background

A. Hybrid Robot

Say a word on operation between configuration done later, as we are not working in a vector space.

1) Kinematic chain: In the literature, a Manipulation Planning problem usually involves a robot and movable objects [2], [5], [3]. This work considers the robot and movable objects as the same level. The term *part* refers either to the robot or to a movable object, *robot* keeping its usual meaning. Altogether, the parts form a kinematic chain defined as follow: Figure?

- the root joint is an anchor joint,
- each movable parts is inserted as a subtree which is a child of the root joint.

The kinematic chain thus built defines the *Hybrid Robot*. It has several particularities deserving to be highlighted. Firstly, It seemlessly handles several robots and articulated objects. Moreover, this *Hybrid Robot* is highly under actuated. This is not surprising in Manipulation Planning. This remark relates Manipulation Planning and Motion Planning for under actuated systems [9]. In both cases, indeed, the configuration space is foliated and a solution path will be a sequence of *transit* and *transfer* paths [3]. For example, computing a quasi-static walking path for a humanoid robot is exactly the same problem as computing a manipulation path. In the former problem, a *transit* path corresponds to the robot in equilibrium on its two feet. A *transfer* path corresponds to the robot on one foot.

Eventually, it standardizes interactions between a robot and objects. Indeed, in our framework, interactions between a robot and an object, between two robots and between two objects are considered as interaction between parts. There is no difference in finding a configuration with a contact between two surfaces of different parts, two surfaces of the same parts, or a surface of a part and one of the environment.

The non actuation of some DOFs is not explicitly handled by the motion planner. Instead, it is abstracted in the *Constraint Graph*, described in section II-D.

2) Configuration space: The configuration space $\mathcal{CS}$ of the *Hybrid Robot* is the cross product of the configuration space of each parts. In the following, we will refer to interpolation between two configurations as *direct path* or *direct motion*. We denote $[q_0, q_1]$ the direct path from configuration q_0 to q_1. This interpolation method is fully specified by the joint types. We represented $SO(3)$ with quaternions and are using SLERP to interpolate between quaternions.

B. *Differentiable functions and constraint solver*

1) Constraints definition: In the framework of this work, a *constraint* is defined by $f(q) = b$, where $f \in C^1\left(\mathcal{CS}, \mathbf{R}^d\right)$ is a differentiable function and $b \in \mathbf{R}^d$ is a reference vector. d is the constraint dimension. We say that configuration q satisfies the constraint iif $f(q) = b$. Two constraints of dimension d_1 and d_2 can be merged into one constraint of dimension $d_1 + d_2$ by stacking element-wise.

On purpose, we keep a redundancy in this formulation. Indeed, $f(q) = b$ and $g(q) = 0$, with $g(q) = f(q) - b$, are the same constraint. The right hand side b will be convenient as it can be used to parameterize level set of the function f. The notion and usefulness of function level sets is detailed at Section II-D.

We also define the *error function*, e, associated with a constraint $f(q) = b$, as

$$e_i(q) = f_i(q) - b_i$$

where the i subscript correspond to the index in a vector. The error function is a differentiable function of the robot configuration space. The tangent application of e gives

$$\dot{e_i}(q) = \dot{f_i}(q)$$

The current framework provides constraints for end-effector position and orientation, relative position and orientation of a joint with respect to another joint, the robot center of mass and a placement constraints (see Section II-C.1).

For a function f, we also define its level sets as

$$L_{q_0}(f) = \{q \in \mathcal{CS} | f(q) = f(q_0)\} \tag{1}$$

2) Constraint solver: Our constraint solver is based on a Newton-Raphson method (see [11, Alg. 2]). Given a robot configuration q_0, the algorithm iteratively find a sequence (q_n) such that $||e(q_{i+1})|| < ||e(q_i)||$. q_{n+1} is drawn from q_n using 2. $A^\dagger$ denotes the Moore-Penrose pseudo-inverse of the matrix A. The algorithm succeeds when $||e(q_n)|| \leq \epsilon$ and fails after a maximum number of iterations or if it could not find q_{n+1} such that the error decreases.

$$q_{n+1} = q_n - \alpha \left(e(\dot{q_n}) \right)^\dagger e(q_n), \tag{2}$$

C. Grasping and placement constraints

In the flow of data, from sensors to robot motion, our work goes after an affordance step. We assume the existence of a tool able to extract information about objects - movable or not - relative to grasping and placement. At the moment, specifying grasps and contact surfaces is done by hand in SRDF files.

For more details about how this can be done automatically, the reader may refer to [15]. We refer to this extra-information as the object documentation.

1) Placement constraints: We define surfaces suitable for contact on objects by dividing them by convex planar polygons. A mesh for instance is a collection of convex planar polygons so this representation is generic enough to represent most objects. We will only allow to consider planar contact between polygons though more complex contact could be considered. For a polygon P, we define its normal n^P, its center[1] C^P, its jth edge e_j^P and:

- $\mathcal{R}^P = (C_P, n^P, e_0^P, n^P \wedge e_0^P)$ a reference frame associated with P,
- $Q_{P,S}$ the projection of C^P onto the plane containing the planar polygon S.

Three distances between a moving polygon M and a support polygon S are defined in 3. S is assumed to be orthogonal to the gravity to ensure stability. Note that if $Q_{M,S} \notin S$ then, $d(M,S)$ is merely the 2-norm of the respective centers, otherwise, it is the distance orthogonal to S. Strictly speaking, d is not a distance function as it is not symmetric.

$$\begin{aligned} d_\perp(M,S) &= C^S C^M . n^S \\ d_\parallel(M,S) &= \begin{cases} ||C^S Q_{M,S}||_2 & \text{if } Q_{M,S} \notin S \\ 0 & \text{otherwise} \end{cases} \\ d(M,S) &= \sqrt{d_\perp(M,S)^2 + d_\parallel(M,S)^2} \end{aligned} \tag{3}$$

Let us consider two parts A and B. Assume we want to put A on B (B can also be the environment). Their polygons are denoted respectively (M_i) and (S_j).Let $I, J = \arg\min_{i,j} d(M_i, S_j)$ and

$$T(i,j) = (\mathcal{R}^{M_i})^{-1} \mathcal{R}^{S_j} = [x, y, z, \theta, \phi, \psi]$$

be the relative transformation between the reference frame of $\mathcal{R}^{M_i}$ and $\mathcal{R}^{S_j}$.

We can now define the placement function and its complement as:

$$P_f(q) = \begin{cases} [x+\epsilon, 0, 0, \phi, \psi] & \text{if } Q_{M,S} \in S \\ [x+\epsilon, y, z, \phi, \psi] & \text{otherwise} \end{cases} \tag{4}$$

$$\overline{P_f}(q) = \begin{cases} [y, z, \theta] & \text{if } Q_{M,S} \in S \\ [0, 0, \theta] & \text{otherwise} \end{cases} \tag{5}$$

ϵ is a value ensuring there will be no collision. It was set to 1mm in the simulations.

The presented method is compatible with other primitives such as cylinder, resp. sphere, which would define linear, resp. punctual contacts. With a bit of additional mathematical calculations, one can similarly define distances for those primitives.

Using these two functions to constraint a motion is equivalent as constraining the relative transformation between the closest polygons. Indeed, by adding zero on unconstrained dimensions, we get

$$P_f(q) + \overline{P_f}(q) = T(I, J)$$

These functions are only piecewise differentiable and piecewise continuous. Though this may cause instability in the solver, no instability were encountered in all the cases and the efficiency of our solver was not altered.

2) Grasping constraint: We model grasping with a gripper/handle model. The gripper documentation contains:

- a center C and a reference frame, wrt to a joint frame, such that its x-axis is an axis of approach and its z-axis is an axis around which you may rotate.
- a clearance value d such that a plane orthogonal to x at a distance d of C would not collide with the gripper. It gives an idea of how big the gripper is.

The handle documentation contains:

- a type of handle, which defines the space of grasp parameters. Possibilities are solid, axial (rotation around z), long (translation along z)...
- a center C and a reference frame, wrt to a joint frame, such that a gripper reference frame aligned with it represent a valid grasp. It might not be the only valid grasp, depending of its type.
- a clearance value d with a similar definition to gripper clearance.

From the grippers and handles definition, we draw the relative transformation between the reference frames of handle i and gripper j, which we denote $T(i,j) = [x, y, z, \theta, \phi, \psi]$. For a solid and axial grasp, the grasp constraints and their

[1]We define the center of a polygon as the centroid of its vertices. We could also use convex optimization method to compute a better geometric center.

complements are:

$$G_f^{solid} = [x, y, z, \theta, \phi, \psi] \quad (6)$$

$$\overline{G_f}^{solid} = [] \quad (7)$$

$$G_f^{axial} = [x, y, z, \theta, \phi] \quad (8)$$

$$\overline{G_f}^{axial} = [\psi] \quad (9)$$

Similar constraints can be defined for other types of handle.

D. Constraint Graph

Manipulation rules, when expressed in the configuration space, induce a foliation [3], [9]. We assume all grasps and placements are continuous. In this section, $q \in \mathcal{CS}$ is a configuration of the *Hybrid Robot* and the reachability set in q represents the accessible subset of $\mathcal{CS}$ by a direct path. We represent the manipulation rules in the form of a graph, called *Constraint Graph*.

Consider Figure 1. Vertices A and B represent submanifolds of $\mathcal{CS}$. For instance, the set of valid placements of an object $S_A = \{q \in \mathcal{CS} | P_f(q) = 0\}$ and of valid grasps $S_B = \{q \in \mathcal{CS} | G_f(q) = 0\}$.

The manipulation rules state that:

- an object is either at a valid placement or grasped, i.e. $q \in S_A \cup S_B$,
- an object cannot move if not grasped, i.e. it must stay in the same placement so $\overline{P_f}(q)$ must be constant when object is not grasped. The reachability set for placement in q_0 is
$$\mathcal{RS}_{place}(q_0) = S_A \cap L_{q_0}(\overline{P_f})$$
- an object cannot move wrt to an end-effector when grasped so, similarly, we must have $\overline{G_f}(q)$ constant when an object is being grasped. The reachability set for grasp in q_0 is
$$\mathcal{RS}_{grasp}(q_0) = S_B \cap L_{q_0}(\overline{G_f})$$
- when the object is grasped and at a valid placement, the reachability set is the union, $\mathcal{RS}_i = \mathcal{RS}_g \cap \mathcal{RS}_p$. With the reduction property [3], it becomes $\mathcal{RS}_i \, \{q \in \mathcal{CS} | G_f(q) = 0 \text{ and } P_f(q) = 0\}$.

Figure 2 represents $\mathcal{CS}$. The $\mathcal{RS}_{place}(q)$, resp. $\mathcal{RS}_{grasp}(q)$, are included in manifolds parallel to the $(L_{f_i}(f))_i$, resp. $(L_{g_i}(g))_i$. The lines shows a series of valid direct paths. Direct paths will stay in one particular leaf of the foliation, that is $\mathcal{RS}_{place}(q)$ or $\mathcal{RS}_{grasp}(q)$.

The function $\overline{G_f}$ parametrizes the foliation induced by the continuous grasp. Similarly, $\overline{P_f}$ parametrizes the foliation induced by the object placement.

$$\begin{tikzcd} A \arrow[r, "1"] & B \end{tikzcd}$$

Fig. 1. A *Constraint Graph* with two nodes and an edge.

On Figure 1, edge 1 represents the set of direct motions from A to B. All the configurations along such motions must

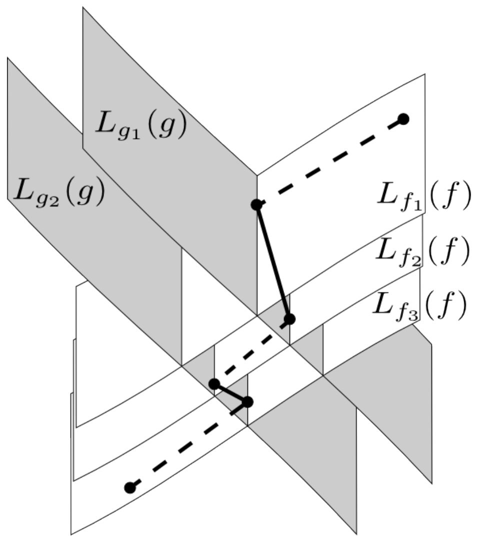

Fig. 2. Example of level sets of two constraints f and g in the configuration space. A manipulation path leading from one level set to another is represented, with its elementary paths. A dashed line, resp. solid, represents an elementary path in a level of f, resp. g. Note that every elementary path lie in a unique level set.

be in S_A or S_B. The particularity of the graph is that an edge is associated to a vertex. Let $\tau : [0, 1] \mapsto \mathcal{CS}$ be an direct path. We must have $\forall s \in]0, 1[\, , \tau(s) \in V$, V being either S_A or S_B, independently of s.

So, if edge 1 is in vertex A, edge 1 is:

$$S_1 = \left\{(q_0, q_1) \in S_A \times S_A \cap S_B | \overline{P_f}(q_0) = \overline{P_f}(q_1)\right\} \quad (10)$$

and, if edge 1 is in vertex B:

$$S_1 = \left\{(q_0, q_1) \in S_A \cap S_B \times S_B | \overline{G_f}(q_0) = \overline{G_f}(q_1)\right\} \quad (11)$$

Note that this holds if A and B are the same vertex. It can be extended to more states, as shown in the examples.

E. Manipulation Planning problem

We define the Manipulation Planning problem in the following terms:

- a *Hybrid Robot* with grippers, handles and contact surfaces,
- environment with contact surfaces,
- ○ a *Constraint Graph*, that can be built automatically in most cases,
- an initial and one or several goal(s) configurations. Goal configuration defined via constraints?

III. Algorithm

This section present an RRT-based algorithm to solve Manipulation Planning problems as defined in II-E, by using the concepts detailled in previous section.

A. Manipulation RRT

The *Manipulation RRT* algorithm integrates the manipulation rules via the *Constraint Graph*. A pseudo-code is given in Algorithm 1. The algorithm is very close to the constrained RRT. The difference are:

Algorithm 1 Manipulation RRT

```
function EXPLORETREE($q_0$)
    ▷ Randomly exploring Random Tree from $q_0$ using the constraint graph
    $\mathcal{T}$.init($q_0$)
    for $i = 1 \to K$ do
        $q_{rand} \leftarrow$ RAND($\mathcal{CS}$)
        $q_{near} \leftarrow$ NEAREST($q_{rand}$,$\mathcal{T}$)
        $e \leftarrow$ CHOOSEEDGE($q_{near}$)
        $path \leftarrow$ CONSTRAINEDEXTEND($q_{near}$,$q_{rand}$,$e$)
        if last step failed then Continue
        end if
        $\mathcal{T}$ .INSERTPATH($path$)
    end for
end function
```

- Line 6: an outgoing edge of the vertex of the *Constraint Graph* containing q_{near} is chosen. Several strategies are possible for this choice. The strategy we use is random choice, with a probability law proportional to user defined edge weights.
- Line 7: Extend q_{near} towards q_{rand} while respecting the manipulation rules encoded by the selected edge.

Algorithm 2 is a pseudo-code for the constrained extension mentionned above.

It first generates q_{new} by projecting q_{rand} into the set $\{q | (q_{near}, q) \in EdgeSet\}$ where $EdgeSet$ refers to either 10 or 11. Eventually, the function checks for collisions and returns the longuest direct subpath of (q_{near}, q_{new}) starting in q_{near}.

This approach allows us to transform numerical into symbolic information. This may constitue a very helpful feedback to a task planner and could enable a task planner to refine its the edge selection strategy, even a strategy that could evolve while the algorithm running. Possible feedbacks are:

- The success rate of the projection onto the edge set. This gives an indication of how hard it is to perform an action, the represented action being harder when a success rate is low.
- At the collision detection step, when the path is not fully valid, recording what geometry where colliding. This may highlight blocking objects.

Currently, we are computing the success rate of the projection but are not using any of feedback. We believe the interpretation of these feedbacks must be done by a task planner.

B. The Rendez-Vous problem

Care has to be taken for cases with continuous grasps and/or placements. Consider the case of an arm manipulator to which we ask to invert the position of two boxes $b_{1,2}$. A solution to this problem would look like:

- move b_1 to an intermediate valid placement,
- move b_2 to b_1 initial position,
- move b_1 to b_2 initial position,

This is not a monotone solution, since b_1 is manipulated twice. It requires to find an intermediate valid placement for b_1. Running the previous algorithm would grow two trees, T_1, from the initial *Hybrid Robot* configuration, and T_2, from the goal *Hybrid Robot* configuration. When extending a tree, always choosing new valid placements from random configurations leads to a zero probability of finding a common valid placement. The problem is thus unsolvable.

To overcome this issue, for each tree, we must try to:

- discover new valid placements,
- reach valid placements found by the other tree.

This can simply be done by memorizing, for each tree, the reached placements, i.e. the set:

$$\left\{\overline{P_f}(q) | q \in T_i, P_f(q) = 0\right\} \tag{12}$$

We also memorize the number of times a given placement has been reached. When an attempt, from T_1, to reach a placement reached by T_2 is made, an element of the set defined in Equation 12 is randomly chosen. We use a probability law which is proportional to the frequency of each element because, if this trial succeeds, there will be more possibilities of linking the two trees.

Naturally, the same remark holds for continuous grasps.

Algorithm 2 Constrained extention

```
    ▷ Extend $q_{near}$ towards $q_{rand}$ while staying in the subset of $S_{edge}$ starting at $q_{near}$
function CONSTRAINEDEXTEND($q_{near}$,$q_{rand}$,$edge$)
    $f \leftarrow edge$.GETEDGESET( )
    $f$.SETLEFTREFERENCE($q_{near}$)
    $q_{new} \leftarrow f$.APPLYCONSTRAINTS($q_{rand}$)
    $p \leftarrow (q_{near}, q_{new})$
    $p \leftarrow$ GETCOLLISIONFREELEFTPART($p$)
    return $p$
end function
```

C. Waypoints

Motion Planning problems with *narrow passage* are still very challenging for today's motion planner. In Manipulation Planning, we encounter this issue when performing a graps or a put-down. The former because of *Hybrid Robot* is close to self-collision, the latter because of *Hybrid Robot* is close to colliding with the environment.

To overcome this issue and help the planner finding a solution, we also define pre-grasping and pre-placement tasks. These tasks do not require more information than grasp and placement. For pre-grasps, we shift the grasp position of the sum of the clearance of the gripper and handle, along the x-axis. For pre-placements, we shift the shape position of a constant value along the supporting shape normal.

D. *Graph builder*

It may be cumbersome to write the *Constraint Graph* by hand as it exponentially increases with the number of grippers and handles. In most cases however, it is possible to build it automatically. Let us say we have a robot with grippers (g_i), objects (o_p), with handles (h_j^p) and contact polygons M^p and environment support polygons S.

Algorithm 3 Graph builder

```
1: function GraphVertex($grasps$)
                                   ▷ Constraint set for vextex $grasps$
2:     $constraints \leftarrow \{\}$
3:     for $(g, h) \in grasps$ do
4:         $constraints \leftarrow constraints + G_f(g, h)$
5:     end for
6:     for all $o$ not grasped do
7:         $constraints \leftarrow constraints + P_f(o, S)$
8:     end for
9:     return $constraints$
10: end function
11: function GraphEdge($grasps, (g, h)$)
    Constraint set for edge from vertex $grasps$ to $grasps + (g, h)$
12:     $constraints \leftarrow \{\}$
13:     for all $o$ not grasped do
14:         $constraints \leftarrow constraints + \overline{P_f}(o, S)$
15:     end for
16:     for all $o$ grasped do
17:         $constraints \leftarrow constraints + \overline{G_f}(o, S)$
18:     end for
19:     return $constraints$
20: end function
```

Algorithm 3 gives two procedures creating the constraint sets for respectively vertices and edges. These procedures can be extended to pre-grasp and pre-placement, as done in the simulations.

IV. Simulations

We were able to compute manipulation paths using our methods on a wide range of examples. This section presents these examples and shows some benchmarks.

A. *Examples*

a) Baxter: We used the Baxter robot to invert the position of two boxes, first by using only its left arm, then by using both. The first case is interesting because it shows the ability to solve non-monotone cases and illustrate the theoretical aspects of section III-B. The second is interesting because the robot has the ability to manipulate the two boxes at the same time.

b) PR2: Do this with Baxter

c) Romeo: In this example, PR2 manipulates a box. The problem has been designed such that it must pass the box from its left hand to its right hand without releasing the box. This example shows that, with a well-designed *Constraint Graph*, the algorithm can generate a manipulation path with a narrow passage - the set of configuration where PR2 holds the box with its two hands is highly constrained.

d) HRP-2: In this example, HRP-2 makes a few steps in a quasi-static equilibrium. This example illustrates the theoretical aspect of section III-B and shows the genericity of the method. Foot placement are considered in the same way as object placement. The planner was able to find a suitable common foot step for both expansion trees.

The input of the problem is:

- Robot: HRP-2, its two feet can be on the ground, with three possible balance constraints;
- Environment: None;
- Constraint Graph: shown in Figure **??**;
- Initial configuration: HRP-2 is on its two feet, the COM between its feet.
- Goal configuration: the same as the initial configuration, shifted 50cm forward.

B. *Benchmark*

Baxter with two boxes in the two cases. Baxter with 3 boxes. Give solving time for HRP2, ROMEO and PR2 (if any)

V. Conclusion

Future work: - collision detection in a smarter way, - integration into a task planner that would use feedback information from the planner.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[7] 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.

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

[9] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *The International Journal of Robotics Research*, 2009.

[10] M. Stilman, K. Nishiwaki, S. Kagami, and J. J. Kuffner, "Planning and executing navigation among movable obstacles," *Advanced Robotics*, vol. 21, no. 14, pp. 1617–1634, 2007.

[11] 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. 1089–1103, 2013. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00654175

[12] S. M. LaValle, J. H. Yakey, and L. E. Kavraki, "A probabilistic roadmap approach for systems with closed kinematic chains," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 3. IEEE, 1999, pp. 1671–1676.

[13] D. Berenson, S. S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *The International Journal of Robotics Research*, p. 0278364910396389, 2011.

[14] M. Stilman, "Global manipulation planning in robot joint space with task constraints," 2010.

[15] A. Stoytchev, "Behavior-grounded representation of tool affordances," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, April 2005, pp. 3060–3065.

[16] S. M. Lavalle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.