Constraint Graphs: Unifying task and motion planning for Navigation and Manipulation Among Movable Obstacles

Joseph Mirabel, Florent Lamiraux

Constraint Graphs: Unifying task and motion planning for Navigation and Manipulation Among Movable Obstacles

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

***Abstract*— We consider a class of advanced motion planning problems including object manipulation, navigation among movable obtacles and legged locomotion. Our approach uses rearrangement rules, encoded in a *Constraint Graph*, a formulation unifying high-level task planning and motion planning. Our method can compute manipulation paths for complex robots and movable - articulated - objects in a static environment, allowing for instance to generate continuous grasps and object placements. A new planning algorithm, *Manipulation RRT*, makes use of the *Constraint Graph* to solve Rearrangement problems and to produce rich feedbacks to higher level planners. Simulation results, for problems where movable objects must be manipulated simultaneously or several times, involving humanoid robots, are presented.**

I. INTRODUCTION

Rearrangement Planning [1] is a class of complex motion planning problems, where a robot must automatically discover a sequence of simple tasks necessary to perform a given high level task. The topic includes Navigation Among Movable Obstacles (NAMO) [2], [3] and Manipulation Among Movable Obstacles (MAMO) [4]. For instance, to swap the position of two boxes with a single robotic arm (high level task), a MAMO algorithm should automatically infer that it is first necessary to move one of the boxes to an intermediary position (task decomposition). But if two arms are available, it should directly swap the boxes.

Our first contribution is the introduction of the *Constraint Graph* formulation. Where several contributions [3], [4], [5], [6], [1] decouple task planning and motion planning, the *Constraint Graph* addresses both issues simultaneously. This formulation is not only more convenient, it also avoids the implemention of complex feedback loops resulting from hierarchical planning (i.e. a task decomposition is chosen, does not succeed because the motion is not executable, such that a new decomposition has to be chosen, and so on).

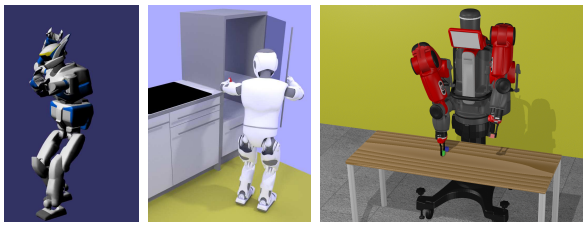

Fig. 1: Various motion planning problems handled in an unified way with a rearrangement planning formulation.

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

Our second contribution is a concrete algorithm using this formulation: the *Manipulation RRT*. This algorithm is particularly suited for rearrangement problems involving more than one object. The strength of the algorithm is that objects can be manipulated simultaneously, several times if necessary, where other formulations usually only consider an object one at the time, and only once. Furthermore, thanks to the *Constraint Graph*, the *Manipulation RRT* understands symbolic facts. This provides richer feedbacks than usual motion planners, which makes our approach suited for an integration in a task planner.

We claim that the *Manipulation RRT* is thus a significant step towards a complete solution to N/MAMO problems. The complete method presented in the paper has been implemented in a clean, generic and open source software, available as a module of the Humanoid Path Planner [1]. All the results presented in the paper can be directly reproduced by simply following the software documentation.

II. RELATED WORK

A. 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 (i.e. polynomial) when the final positions of objects are specified and NP-hard otherwise [2]. To the best of our knowledge, no existing planner is able to solve generic instances of this problem. The limitations are mostly the number of objects and possible interactions between objects.

Rearrangement Planning [1] aims at using the robot to act on the environment. On the contrary, NAMO aims at navigating, i.e. finding a path for the robot. MAMO is a combination of both. The robot must achieve a specific task (e.g. moving an object) and is allowed to move other obstacles.

In spite of its complexity, practical solutions for subclasses of NAMO have arisen. The case of regrasping has been solved efficiently [7] 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 regrasping is required, this property does not reduce the inherent complexity of NAMO.

Among most approaches [3], [4], [5], [6], [1], many only consider monotone solutions, i.e. where each object is moved at most once. They mostly use two-level methods composed

[1]http://projects.laas.fr/gepetto/index.php/Software/Hpp

of a symbolic or task planner and of a motion planner [8]. While the high 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. The latter solves motion planning problems of a specific case of the combinatorial of the end-effectors, objects and their possible placements. An interface between the two levels [5] is thus required.

These two-level approaches benefit from the existence of efficient solutions to discrete problems. However, a first drawback is the difficulty of sharing information between the task planner and the motion planner. A motion planning problem is specified in a continuous uncountable set whereas a task planning problem only has a discrete representation of the configuration space. A second and more important drawback is the lack of integration between both levels. 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 planners are usually not able to provide a reason for their failure.

Other approaches have tried to bring motion planning at a higher level of abstraction. To this end, the configuration space is segmented [9], [10], [11]. A graph representation of the configuration space emerges from this segmentation. In contrast with multi-level approaches, they are not specifically reasoning about blocking objects.

B. Constrained planning

Motion planning algorithms were first developed to solve problems such as the piano mover problem [8]. Randomized planners succeeded in solving motion planning problems in high dimensional configuration space.

Constrained problems concern a more generic class of problems, such as motion planning for humanoid robots [12] or closed chain kinematics [13]. Kinematic chain closure, when two arm manipulators collaborate or when an object is lifted, are also constrained problems. Constrained planning is an essential component of Rearrangement Planning.

Randomized algorithms like Randomly Exploring Random Trees (RRT) or Probabilistic Roadmap (PRM) fail to solve constrained problems [14], [8, chap. 7] because the submanifold satisfying the constraints, often denoted $\mathcal{C}_{const}$, has a the zero volume. Most methods [12], [13], [14] rely on an ability to efficiently sample $\mathcal{C}_{const}$, typically using inverse kinematics. However these approaches do not apply when the robot degrees of freedom (DOF) are correlated by stability constraints. See [15] for details of existing methods.

In this paper, we focus on differentiable constraints and we use the gradient descent method developed in [12], which is similar to the first-order retraction algorithm. It tends to have better results than other algorithms [15].

In section III we propose the first complete formal definition of the rearrangement planning problem. This formulation leads to the design of a generic algorithm, described in details in Section IV. Results involving complex manipulation scenarios and robots are then presented in Section V.

III. Formal definition of the rearrangement planning problem

A. Hybrid Robot

In the literature, a Rearrangement Planning problem usually involves a robot and movable objects [4], [5], [7]. This work considers the robot and movable objects as *parts*. Altogether, the parts form the *Hybrid Robot*, which is a kinematic chain starting with an anchor joint followed by each *part* kinematic chain. With this formulation we can consider seamlessly an under actuated legged robot, a manipulator arm manipulating static or articulated objects, two robots moving a table, a mobile manipulator crossing a door, etc.

The *Hybrid Robot* is highly under actuated. As in Motion Planning for under actuated systems [11], the configuration space is foliated and a solution path is a sequence of *transit* and *transfer* paths [7]. For instance with this definition quasi-static walking for a humanoid robot is a rearrangement planning problem. A *transit* path corresponds to the robot in equilibrium on its two feet. A *transfer* path corresponds to the robot on one foot.

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

The configuration space $\mathcal{C}$ of the *Hybrid Robot* is the Cartesian product of the configuration space of each part. In the following, the *direct path* from configuration $\mathbf{q_0}$ to $\mathbf{q_1}$ refers to the interpolation between the two configurations and is denoted $[\mathbf{q_0}, \mathbf{q_1}]$.

B. Differentiable functions and constraint solver

The *Hybrid Robot* is subject to constraints. For instance, a static object cannot move when the robot is not grasping it, and a legged robot must be in static equilibrium at all time. We consider first the standard definition of such constraints, before introducing the numerical methods that we use to sample a configuration respecting a constraint.

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

On purpose, we keep a redundancy in this formulation. Indeed, $f(\mathbf{q}) = \mathbf{b}$ and $g(\mathbf{q}) = 0$, with $g(\mathbf{q}) = f(\mathbf{q}) - \mathbf{b}$, are the same constraint. The right hand side $\mathbf{b}$ is convenient as it parameterizes the level sets of f.

We also define the *error function* e, associated with a constraint $f(\mathbf{q}) = \mathbf{b}$ by

$$e(\mathbf{q}) = f(\mathbf{q}) - \mathbf{b}$$

The error function is a differentiable function from $\mathcal{C}$ to $\mathbb{R}^d$, and we write $\dot{e}(\mathbf{q}) = \dot{f}(\mathbf{q})$.

Many constraints can be implemented in this way: for instance our framework handles end-effector position and orientation, relative position and orientation of a joint with respect to another joint, the robot center of mass and placement constraints.

For a function f, a level set is defined as

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

2) Constraint solver: Our constraint solver is based on the Newton-Raphson method (see [12, Alg. 2]). For robot configuration q_0, the algorithm iteratively finds a sequence $(\mathbf{q_n})$ such that $||e(\mathbf{q_{i+1}})|| < ||e(\mathbf{q_i})||$. $\mathbf{q_{n+1}}$ is given by $\mathbf{q_n}$:

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

where $A^\dagger$ denotes the Moore-Penrose pseudo-inverse of the matrix A. The algorithm succeeds when $||e(\mathbf{q_n})|| \leq \epsilon$ and fails after a maximum number of iterations or if it could not find $\mathbf{q_{n+1}}$ such that the error decreases.

C. Grasping and placement constraints

As our focus is on Rearrangement Planning, essential constraints are grasping constraints (to specify that an effector grasps an object), and placement constraint (to make the position of a body coincide with an objective, e.g. to create a contact). Furthermore, we assume an affordance step provides relevant information about objects - movable or not - relatively to grasping and placement. This step is out of the scope of this work. In our framework, grasps and contact surfaces specification is using the Semantic Robot Definition Format (SRDF).

For more details about how this can be done automatically, the reader may refer to [16]. We refer to affordance as the object documentation.

The grasping and placements constraints are each characterized by two functions:

- a function $P_f(\mathbf{q}) = \mathbf{0}$, which defines the set of configurations that verify the constraint;
- a complement function $\overline{P_f}(\mathbf{q})$, which gives a parametrization of the constraint.

1) Placement constraints: Possible contact surfaces are defined as convex planar polygons. This representation is generic enough to represent most objects (object meshes being typically composed of convex triangles). We only consider planar contact between polygons though more complex contact models could be considered.

Two parts A and B (B can be the environment) are in contact if their associated contact polygons are in contact, that is the distance between the surfaces is lesser than a threshold ϵ. The difficulty is to define the distance, which includes the relative orientation of the surface normals, and the minimal distance between the surfaces. A possible distance is defined in Appendix VI-A.

We assume that A is moving (along with its moving polygon M), and B is static (as its support polygon S), without loss of generality. We make the simplification assumption that S is flat (its normal pointing in the direction opposite to the gravity). Let $I, J = \arg\min_{i,j} d(M_i, S_j)$ be the points of each polygon that are the closest according to the distance function d and

$$T(i,j) = (\mathcal{R}^{M_i})^{-1}\mathcal{R}^{S_j} = [x, y, z, \Omega_x, \Omega_y, \Omega_z]$$

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:

$$P_f(\mathbf{q}) = \begin{cases} [x, 0, 0, \Omega_y, \Omega_z] & \text{if } Q_{M,S} \in S \\ [x, y, z, \Omega_y, \Omega_z] & \text{otherwise} \end{cases} \tag{3}$$

$$\overline{P_f}(\mathbf{q}) = \begin{cases} [y, z, \Omega_x] & \text{if } Q_{M,S} \in S \\ [0, 0, \Omega_x] & \text{otherwise} \end{cases} \tag{4}$$

with $Q_{M,S}$ the projection of the centroid of M onto S.

If $P_f(\mathbf{q}) = \mathbf{0}$, the distance function is minimal, so we are in contact. Using these two functions to constrain a motion is equivalent to constraining the relative transformation between the closest polygons. Indeed, by adding zero on unconstrained dimensions, we get

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

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

These functions are only piecewise continuous and differentiable. Though this may cause instability in the solver, we encountered none in all the cases and the efficiency of our solver was not altered. A more complex approach can provide continuous gradient [17].

2) Grasping constraint: We model grasping with a gripper/handle model between two parts in an arbitrary configuration. Again without loss of generality, we consider that the gripper is moving while the handle is not.

Grasping is in fact a placement problem, but the constraints apply to predefined reference frames rather than surfaces. New constraints are given by the grasping type:

- in "solid" grasping, a reference frame from the gripper must match exactly a frame from the handle. This can be used to model a hand grasping a door handle;
- in "axial" grasping, the rotation about the z axis of the handle is free (as for grasping a glass in the subway);
- in "long" grasping, the translation about the z axis of the handle is free (as for grasping a pen)...

Using the same development as for the placement constraints, we can obtain the following methods for the solid and axial cases:

$$G_f^{solid} = [x, y, z, \Omega_x, \Omega_y, \Omega_z] \tag{5}$$

$$\overline{G_f}^{solid} = [] \tag{6}$$

$$G_f^{axial} = [x, y, z, \Omega_x, \Omega_y] \tag{7}$$

$$\overline{G_f}^{axial} = [\Omega_z] \tag{8}$$

Similar constraints can be defined for other handle types.

D. Constraint Graph

Manipulation rules, when expressed in $\mathcal{C}$, induce a foliation [7], [11]. We assume all grasps and placements are continuous. In this section, $\mathbf{q} \in \mathcal{C}$ is a configuration of the *Hybrid Robot* and the reachability set in $\mathbf{q}$ represents the accessible subset of $\mathcal{C}$ by a direct path. We represent the manipulation rules in the form of a *Constraint Graph*.

In Figure 2, the vertices A and B represent submanifolds of $\mathcal{C}$. For instance, the set of valid placements of an object $S_a = \{\mathbf{q} \in \mathcal{C} | P_f(\mathbf{q}) = 0\}$ and of valid grasps $S_b = \{\mathbf{q} \in \mathcal{C} | G_f(\mathbf{q}) = 0\}$.

The manipulation rules state that:

- an object is either at a valid placement or grasped, i.e. $\mathbf{q} \in S_a \cup S_b$,
- an object cannot move if not grasped. It must stay in the same placement so $\overline{P_f}(\mathbf{q})$ must be constant when the object is not grasped. The placement reachability set at $\mathbf{q_0}$ is
$$\mathcal{RS}_{place}(\mathbf{q_0}) = S_a \cap L_{\mathbf{q_0}}(\overline{P_f})$$
- an object cannot move wrt to an end-effector when grasped so, similarly, we must have $\overline{G_f}(\mathbf{q})$ constant when an object is being grasped. The reachability set for grasp in $\mathbf{q_0}$ is
$$\mathcal{RS}_{grasp}(\mathbf{q_0}) = S_b \cap L_{\mathbf{q_0}}(\overline{G_f})$$
- when the object is grasped and at a valid placement, the reachability set is the intersection, $\mathcal{RS}_i = \mathcal{RS}_g \cap \mathcal{RS}_p$. With the reduction property [7], it becomes $\mathcal{RS}_i \{\mathbf{q} \in \mathcal{C} | G_f(\mathbf{q}) = 0 \text{ and } P_f(\mathbf{q}) = 0\}$.

Figure 3 represents the configuration space. $\mathcal{RS}_{place}(\mathbf{q})$, resp. $\mathcal{RS}_{grasp}(\mathbf{q})$, are included in manifolds parallel to the $(L_{f_i}(f))_i$, resp. $(L_{g_i}(g))_i$. The lines show a sequence of valid direct paths. Direct paths stay in one particular leaf of the foliation, that is $\mathcal{RS}_{place}(\mathbf{q})$ or $\mathcal{RS}_{grasp}(\mathbf{q})$.

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.

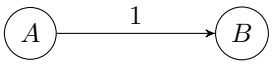

Fig. 2: A *Constraint Graph* with two nodes and an edge.

On Figure 2, edge 1 represents the set of direct paths from A to B. All the configurations along such motions must be in S_a or S_b. The particularity of the graph is that an edge is associated to a vertex. Let $\tau : [0, 1] \mapsto \mathcal{C}$ be a 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:

$$S_1 = \left\{(\mathbf{q_0}, \mathbf{q_1}) \in S_a \times S_a \cap S_b | \overline{P_f}(\mathbf{q_0}) = \overline{P_f}(\mathbf{q_1})\right\} \quad (9)$$

and, if edge 1 is in vertex B:

$$S_1 = \left\{(\mathbf{q_0}, \mathbf{q_1}) \in S_a \cap S_b \times S_b | \overline{G_f}(\mathbf{q_0}) = \overline{G_f}(\mathbf{q_1})\right\} \quad (10)$$

Note that this holds if A and B are the same vertex. Figure 6 provides the *Constraint Graph* for one example.

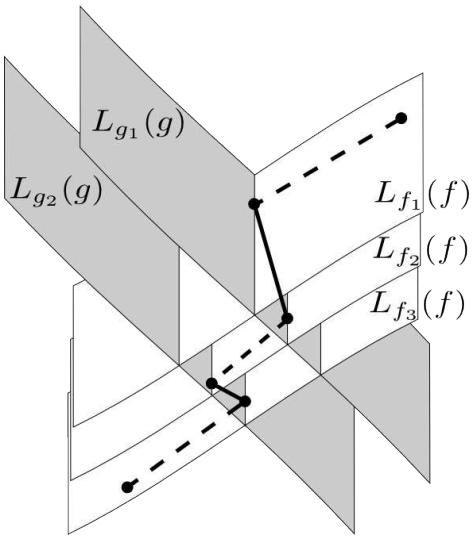

Fig. 3: Example of level sets of two constraints f and g in $\mathcal{C}$. A manipulation path leading from one level set to another is represented, with its elementary paths. A dashed line, resp. solid, represents an elementary path in a level of f, resp. g. Note that every elementary path lie in a unique level set.

E. Rearrangement Planning problem

We define the Rearrangement Planning problem in the following terms:

- a *Hybrid Robot* with grippers, handles and contact surfaces,
- environment with contact surfaces,
- a *Constraint Graph*, built automatically in most cases,
- an initial and one or several goal(s) configurations.

Next section presents an algorithm to solve this problem.

IV. *Manipulation RRT*

Algorithm 1 Manipulation RRT

```
1: function EXPLORETREE(q_0)
                ▷ RRT from q_0 using the constraint graph
2:     T.init(q_0)
3:     for i = 1 → K do
4:         q_rand ← RAND(C)
5:         q_near ← NEAREST(q_rand, T)
6:         e ← CHOOSEEDGE(q_near)
7:         path ← CONSTRAINEDEXTEND(q_near, q_rand, e)

8:         if last step succeeded then
9:             T .INSERTPATH(path)
```

The *Manipulation RRT* is introduced by a basic example where Baxter manipulates a box.

A. Example

Figure 4 shows the *Constraint Graph* for this simple case. To keep the example minimal, the box can be grasped by at

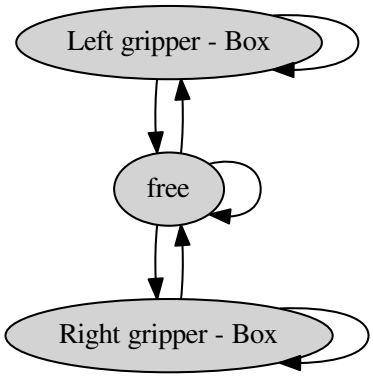

Fig. 4: *Constraint Graph* for the Baxter robot manipulating a box

most one gripper and the grasps are rigid. The constraints are $P_f(\mathbf{q}) = 0$, $\overline{P_f}(\mathbf{q}) = \text{ref}$, $G_f^{left}(\mathbf{q}) = 0$, $G_f^{right}(\mathbf{q}) = 0$.

The *Manipulation RRT* (Alg. 1) incrementally builds two trees of configurations from an initial configuration and from a goal configuration. These two trees necessarily contain configurations in one of the node of the *Constraint Graph*. It works as follow:

- a random configuration $\mathbf{q}_{rand}$ is shot, regardless of any constraints.
- for each tree, the configuration $\mathbf{q}_{near}$ nearest to $\mathbf{q}_{rand}$ is found (function NEAREST).
- $\mathbf{q}_{near}$ is in a vertex of the *Constraint Graph*, for instance "free". This vertex has three outgoing edges, one for each type of path: put left gripper in grasp position, put right gripper in grasp position, unconstrained Baxter path. In all cases, the box cannot move. One of these types is selected. For clarity, at this stage, this choice is made at random (function CHOOSEEDGE).
- the reachability set in $\mathbf{q}_{near}$ is built using the selected type. If "put left gripper in grasp position" is chosen, then this set is $\left\{\mathbf{q} | P_f(\mathbf{q}) = 0, \overline{P_f}(\mathbf{q}) = \overline{P_f}(\mathbf{q}_{near}) \text{ and } G_f^{left}(\mathbf{q}) = 0\right\}$ - unchanged placement and left grasp (functions GETEDGESET and SETLEFTREFERENCE).
- $\mathbf{q}_{rand}$ is projected into the reachability set (APPLYCONSTRAINTS)
- $\mathbf{q}_{near}$ is extended towards $\mathbf{q}_{proj}$, the projection of $\mathbf{q}_{rand}$ onto the manifold satisfying the constraints, up to collision (function GETCOLLISIONFREELEFTPART).

When a configuration is added to a tree, a connection between the new configuration and the other tree is tried. When this connection succeeds, the algorithm succeeds.

B. Manipulation RRT

The *Manipulation RRT* algorithm integrates the manipulation rules via the *Constraint Graph*. A pseudo-code is given in Algorithm 1. The algorithm selects and applies constraints in order to respect the manipulation rules. It is based on RRT, with the following noticeable changes:

- Line 6: an outgoing edge of the vertex of the *Constraint Graph* containing $\mathbf{q}_{near}$ 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 $\mathbf{q}_{near}$ towards $\mathbf{q}_{rand}$ while respecting the manipulation rules encoded by the selected edge. This procedure is presented in Algorithm 2.

Algorithm 2 first generates $\mathbf{q}_{new}$ by projecting $\mathbf{q}_{rand}$ into the set $\{\mathbf{q} | (\mathbf{q}_{near}, \mathbf{q}) \in E_{set}\}$ where E_{set} refers to the edge set in Eq. 9 or 10. The function finally checks for collisions and returns the longest direct sub path of $(\mathbf{q}_{near}, \mathbf{q}_{new})$ starting at $\mathbf{q}_{near}$.

This approach allows us to transform numerical information into symbolic information. This may constitute a very helpful feedback to a task planner, to refine the edge selection strategy, apdated online. Possible feedbacks are:

- The success rate of the projection onto the edge set. This reflects the difficulty of performing an action. The harder the action, the lower the success rate;
- Statistics on which geometries are colliding and prevent full extension, highlights blocking objects.

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

Algorithm 2 Constrained extension

1: **function** CONSTRAINEDEXTEND($\mathbf{q}_{near}$,$\mathbf{q}_{rand}$,$edge$)
   ▷ Extend $\mathbf{q}_{near}$ towards $\mathbf{q}_{rand}$ while staying in the subset of S_{edge} starting at $\mathbf{q}_{near}$
2: $f \leftarrow edge$.GETEDGESET( )
3: f.SETLEFTREFERENCE($\mathbf{q}_{near}$)
4: $\mathbf{q}_{new} \leftarrow f$.APPLYCONSTRAINTS($\mathbf{q}_{rand}$)
5: $p \leftarrow (\mathbf{q}_{near}, \mathbf{q}_{new})$
6: $p \leftarrow$ GETCOLLISIONFREELEFTPART(p)
7: **return** p

1) Connected component visibility: 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 permute the position of two boxes $b_{1,2}$. All the solutions have an intermediate valid placement for one of the boxes. There is no monotone solution. Running the *Manipulation RRT* 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 and to 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:

$$\left\{\overline{P_f}(\mathbf{q}) | \mathbf{q} \in T_i, P_f(\mathbf{q}) = 0\right\} \tag{11}$$

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 Eq. 11 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.

2) Waypoints: Motion Planning problems with *narrow passage* are still very challenging for today's motion planner. In Rearrangement Planning, this issue is encountered when performing a grasp or a put-down, because, in both cases, *Hybrid Robot* is close to collision.

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.

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: **for all** $o \in$ objects not grasped **do**
6: $constraints \leftarrow constraints + P_f(o, S)$
7: **return** $constraints$
8: **function** GRAPHEDGE($grasps, (g, h)$)
Constraint set for edge from vertex $grasps$ to $grasps + (g, h)$
9: $constraints \leftarrow \{\}$
10: **for all** $o \in$ objects not grasped **do**
11: $constraints \leftarrow constraints + \overline{P_f}(o, S)$
12: **for all** $o \in$ objects grasped **do**
13: $constraints \leftarrow constraints + \overline{G_f}(o, S)$
14: **return** $constraints$

3) 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. 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. To build the graph, one has to call these procedures on each possible pair of handles and grippers. However, adding infeasible states to the *Constraint Graph* affects the solving time.

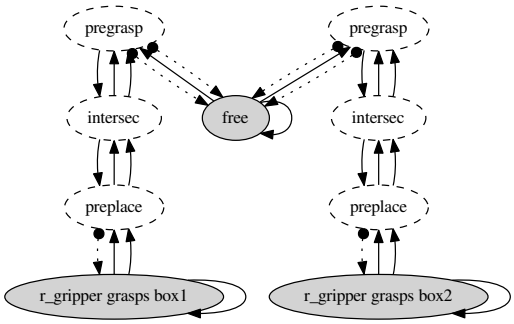

Fig. 6: *Constraint Graph* for case 1 with the Baxter robot: 2 boxes and considering only the right arm

V. RESULTS

We are able to compute manipulation paths using our method for a wide range of examples. This section presents these examples and shows some benchmarks.

A. Manipulation

On these simulations, the Baxter robot moves boxes on a table, in 4 different cases. In cases 1 and 2, we explicitly specified, by removing the right part of the *Constraint Graph*, that only the left arm is used. In cases 3 and 4, both arms are used. The *Constraint Graph* is given in Figure 6. In the first case, the box positions are only shifted and the problem is monotone. In the other 3 cases, the boxes are to be permuted so the solutions to these 3 problems are not monotone.

Table I, the accompanying video and Figure 5 summarizes the results. The planner is run on a standard 2.4GHz, RAM 4Go, 4 cores, desktop computer with no parallelization. The solver is able to find solutions in all the four cases. For cases 1 and 2, the problem is not difficult and the solution comes quickly. Cases 3 and 4 corresponds to artificially-hard toy problems, yet the planner is able to discover a solution in a reasonable amount of time.

The first case allows a comparison with other approaches of the state of the art. As we are not using any task planner, we are not performing as good on monotone cases. In contrast, we can solve non-monotone instances, as shown in cases 2, 3, 4. These cases shows the ability to discover new common valid placement. Case 3 and 4 also show the ability of the planner to consider simultaneous manipulation.

B. Grasping behind a door

We demonstrate here the implementation of a more complex motion planning. A mobile (humanoid) robot has to

Cases	1	2	3	4
Nb boxes	2 (monotone)	2	2	3
Arms	Right	Right	Both	Both
Time (s)	7.6 ± 7.2	27 ± 19	38 ± 24	1959 ± 1045

TABLE I: Mean solving time, $\pm$ standard deviation, on 100 runs, for various cases with Baxter robot.

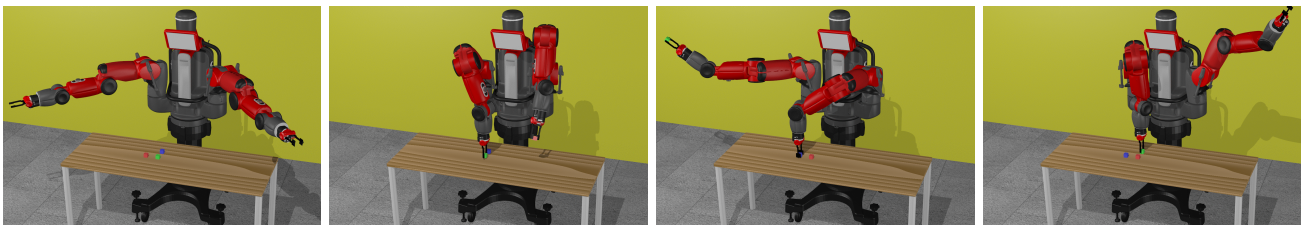

Fig. 5: Baxter permutes the position of 3 boxes.

Fig. 7: Romeo puts a box in a fridge.

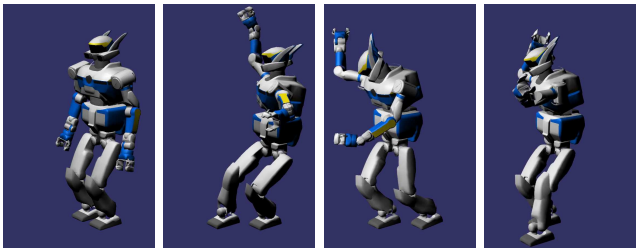

Fig. 8: HRP-2 walking quasi-statically.

grasp an object and place it inside a fridge while opening the fridge door. Once more, the sequence of high level actions is not given. We also demonstrate that this approach, valid for mobile robots, can simply be extended to humanoid robots. First, we generate a path for the sliding robots. Then the path can be post processed [12], [18] to generate a walking trajectory. We successfully planned a path where the Romeo robot take an object and put it in a fridge. The solution found is included in the video and summarized in Figure 7.

C. Solving locomotion problems

In this example, we show our ability to model locomotion problems as Rearrangement Planning problems. With our approach, we are able to compute a quasi-static walking path for HRP-2. This example shows the ability of finding common valid foot placement and that our approach is generic. Foot placement are considered in the same way as object placement. The exact same planner, used for manipulation, is then used the locomotion trajectory. The planner was able to find a suitable common foot step for both expansion trees. The result is shown in the video and summarized in Figure 8.

VI. Conclusion

Future works concern the integration of our motion planner into a higher level task planner and a collision detection that would make use of the *Constraint Graph*. We believe both can reduce solving time. The former helps guiding the search while the latter avoids checking for collision between object not moving relative to one another.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[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] 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.

[7] T. Simon, 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.

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

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.
[17] A. Escande, S. Miossec, M. Benallegue, and A. Kheddar, "A Strictly Convex Hull for Computing Proximity Distances With Continuous Gradients," *IEEE Transactions on Robotics*, vol. 30, no. 3, pp. 666–678, 2014.
[18] J. Carpentier, S. Tonneau, M. Naveau, O. Stasse, and N. Mansard, "A Versatile and Efficient Pattern Generator for Generalized Legged Locomotion," Sept. 2015, this paper has been submitted to the IEEE International Conference on Robotics and Automation (ICRA) 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01203507

APPENDIX: DISTANCE FUNCTIONS

A. Placement distance functions

For a polygon P, $\mathbf{n^P}$ denotes its normal, C^P its center[2], $\mathbf{e_j^P}$ its jth edge and:

- $\mathcal{R}^P = (C_P, \mathbf{n^P}, \mathbf{e_0^P}, \mathbf{n^P} \wedge \mathbf{e_0^P})$ 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 12. 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 euclidian distance between of the centers, otherwise, it is the distance orthogonal to S. Strictly speaking, d is not a distance function as it is not symmetric.

$$\begin{aligned} d_\perp(M,S) &= \mathbf{C^S C^M}.\mathbf{n^S} \\ d_\parallel(M,S) &= \begin{cases} ||\mathbf{C^S Q_{M,S}}||_2 & \text{if } Q_{M,S} \notin S \\ 0 & \text{otherwise} \end{cases} \\ d(M,S) &= \sqrt{d_\perp(M,S)^2 + d_\parallel(M,S)^2} \end{aligned} \quad (12)$$

[2]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.