Manipulation planning: addressing the crossed foliation issue

Joseph Mirabel, Florent Lamiraux

Manipulation planning: addressing the crossed foliation issue

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

***Abstract*— This paper deals with manipulation planning. First, we propose a new tool called the *constraint graph* to describe the various motion constraints relative to a manipulation planning problem. Then, we describe a problem arising for some manipulation planning problems called the *crossed foliation issue*. We propose a extension of RRT algorithm that explores the leaves of the foliations generated by motion constraints and that solves the crossed foliation problem. Finally, we show a wide variety of problem that our approach can solve.**

I. INTRODUCTION

Manipulation planning as a computational geometry problem has raised a lot of interest for the past fourty years. Pioneering works by [1] and [2] first considered low dimensional problems where robots and objects move in translation. [3] is the first work that applies random motion planning methods developed a few years earlier [4] to the manipulation planning problem. Recently, the domain has regained in interest with various variants where papers propose approaches that tackle the inherent complexity of manipulation planning. The domain is traditionally divided into several categories. Navigation among movable obstacles (NAMO) consists in planning a path for a robot that needs to move obstacles in order to reach the goal configuration [5], [6], [7]. Rearrangement planning consists in automatically finding a sequence of manipulation paths that moves several objects from initial configurations to a specified goal configuration [8], [9], [10]. Multi-arm motion planning has also given rise to a lot of papers [11], [12], [13]. From a geometric point of view, manipulation planning is a hybrid problem where discrete states (gripper A holds object B) are defined by continuous constraints on the positions of objects and robots. States are connected by manipulation trajectories that give rise to the underlying structure of a graph the nodes of which are the discrete states [14]. This structure although not expressed as such is present in various papers [15], [16], [17]. The partially discrete nature of the problem has also given rise to integration of task and motion planning techniques [18], [19], [20], [21], [22].

A. The crossed foliation issue

Surprisingly, the geometric structure of the problem, apparently well understood [23] has not been investigated in depth and some simple problems are out of reach for most of the previously cited papers that explore the configuration space using RRT-like methods.

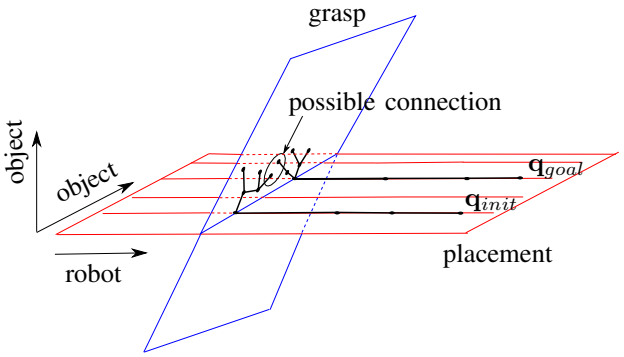

Fig. 1: If the grasp is unique, any pair of grasp configurations are connectable by a *transfer* path.

Let us consider the problem of manipulation defined by a cylindrical vertical object like a bottle, a table and a simple 6-dof manipulator robot with a gripper. The configuration space of the system is $\mathcal{C} \triangleq [-\pi, \pi]^6 \times SE(3)$. This problem is defined by two states

1) *placement*: the subspace of configurations of the system where the object is standing on the table: height of object is constant, roll and pitch of the object are equal to 0.
2) *grasp*: the subspace of configurations where the object is grasped by the robot.

Admissible motions in *placement* are called *transit* paths, while admissible motions in *grasp* are called *transfer* paths. Along *transit* paths, the position of the object on the table remains constant. This constraint defines a *foliation* of *placement*. Leaves of the foliation are parameterized by the 3 remaining degrees of freedom of the object on *placement*: horizontal translation and yaw angle of the object.

In many cases *grasp* is defined by a fixed relative position between the gripper and the object. In this case, *grasp* is not foliated (or composed of only one leaf). In this case, extending a RRT-tree from the initial configuration and from the goal configuration will solve the problem of moving the object at another configuration. The trees will connect in *grasp* space.

In the general case, objects can be grasped in a continuous way. In our example, the relative orientation of the object with respect to the gripper is free around the vertical axis of the object. In this case, *grasp* is also foliated. The foliation is parameterized by the relative angle of the object with respect to the gripper along the object axis. In this case, RRT-trees rooted in placement will never meet, since the probability

*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.

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

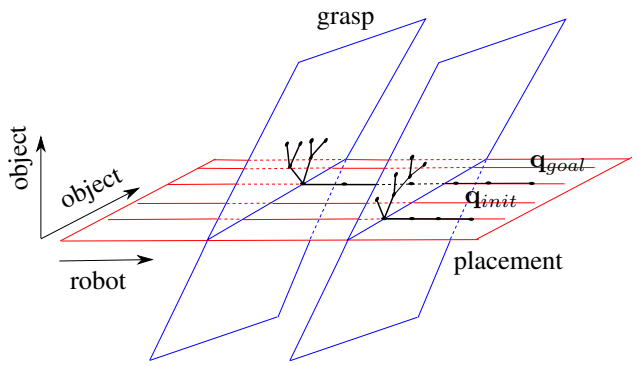

Fig. 2: The crossed foliation issue: if *grasp* and *placement* are foliated, trees rooted at initial and goal configurations will never meet.

that both trees start exploring *grasp* with the same grasp is equal to 0. We call this issue the crossed foliation issue. Figures 1 and 2 illustrate the reasoning.

To our knowledge, this issue has never been stated and solved in a general way. Note that [3], [12] and [10] overcome the issue by searching *grasp* $\cap$ *placement* and by approximating the non-feasible solution path by a sequence of admissible manipulation paths.

B. Contribution

The contributions of this paper are

1) to propose a new concept (briefly introduced in [7]) called the *constraint graph* that encodes the constraints relative to any manipulation problem,
2) to propose an algorithm that automatically builds the constraint graph of a given problem,
3) to propose a simple algorithm called *Manipulation RRT* that solves manipulation problems encoded in a constraint graph, including problems featuring the crossed foliation issue.

Experimental results are proposed at the end of the paper. They provide an example of manipulation problem where Romeo robot holds a placard with two hands and needs to perform a sequence of one hand and two hand grasps to rotate the object. This example would be very difficult to solve with other existing approaches.

II. Manipulation graph

Let $\mathcal{C}$ denote the Cartesian product of the configuration spaces of all robots and objects considered. We make use of the following concepts:

a) Numerical constraint f: C^1 mapping from $\mathcal{C}$ to $\mathbb{R}^m$ where m is an integer. $\mathbf{q} \in \mathcal{C}$ is said to satisfy the constraint iff

$$f(\mathbf{q}) = 0 \quad (\text{resp. } f(\mathbf{q}) \leq 0\ )$$

if f is an equality (resp. inequality) constraint.

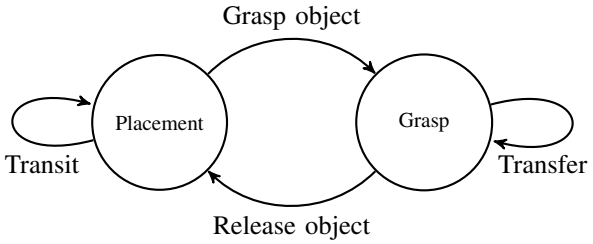

Fig. 3: Constraint graph of a robot with one gripper manipulating one object.

b) Parameterized numerical constraint g: same as equality constraint, except that the constraint is satisfied iff

$$g(\mathbf{q}) = g_0$$

where $g_0 \in \mathbb{R}^m$ is called the parameter of the numerical constraint.

c) Projection $proj$ on a numerical constraint f: mapping from a subspace of $\mathcal{C}$ to $\mathcal{C}$ such that for any $\mathbf{q}$ in the input subspace, $proj(\mathbf{q})$ satisfies the numerical constraint. In our implementation, projection is obtained by using Newton-Raphson algorithm.

d) Constraint graph: graph the nodes of which are called *States* and the edges of which are called *Transitions*. The space of admissible configurations of a manipulation problem is the union of the states.

e) State: a state contains a numerical constraint. A configuration belongs to the state iff it satisfies the numerical constraint.

f) Transition: a transition contains a parameterizable numerical constraint g. The constraint graph contains a transition T between two states S_0 and S_1 if there exists

1) $\mathbf{q}_0 \in S_0$, $\mathbf{q}_1 \in S_1$
2) a path γ from $[0,1]$ to $\mathcal{C}$ such that

$$\gamma(0) = \mathbf{q}_0 \quad \text{and} \quad \gamma(1) = \mathbf{q}_1$$
$$\forall t \in [0,1],\ g(\gamma(t)) = g(\mathbf{q}_0)$$

Note that:

1) the existence of a transition does not imply the existence of a path between the corresponding states, (for instance *placement* on a table unreachable by a robot may be connected to *grasp* by a transition),
2) the constraint graph is not unique and depends on the partition of the admissible configuration space into numerical constraints,
3) for most manipulation problems, it is easy to produce an effective constraint graph.

Figure 3 displays the constraint graph of the simple example described in Section I-A

A. Construction of the constraint graph

In this section, we explain how a constraint graph is automatically built from a set of objects and robot grippers.

1) Placement: they are defined by contact between convex polygons attached to objects or to the environment. the contact constraint between two polygons P_0 and P_1 is satisfied iff the center of P_1 (projection of the center of mass of object 1 onto P_1 plane) lies inside P_0 and if P_0 and P_1 normals are colinear. We denote by

$$\text{CONTACT}(L_1, L_2)$$

the numerical constraint that enforces contact between either polygon of two sets L_1 (attached to the object) and L_2 (attached to the environment). This single constraint is defined by evaluating the constraint defined by the closest pair of polygons in L_1 and L_2.

2) Grasp: We attach to each gripper a frame centered at the center of the gripper (between the jaws), and to each object one or several frames called *handles*. A gripper can grasp an object when both frames coincide (up to possible degrees of freedom for symmetric objects, like free rotation around axis for cylindrical objects, free translation for long and thin objects,...). We denote by

$$\text{GRASP}(g, h)$$

the numerical constraint defined by the grasp of handle h by gripper g. The dimension of the constraint can be less than 6 if the grasp does not constrain all the degrees of freedom (5 for cylindrical objects for instance).

Grippers, handles and contact surfaces are part of the description of the robots, objects, environment and need to be provided.

As explained above grasp and contact constraints may partially constrain the relative position of an object with respect to a gripper or to the environment. Once an object is grasped, however or once it is released in contact, the relative position becomes fully constrained. We thus define

$$\text{FIXED}(obj, g)$$

where obj is an object and g is either a robot gripper or the environment as the parameterizable numerical constraint that keeps the relative position of the object with g constant.

B. Algorithm

Algorithm 1 describes the construction of the constraint graph relative to a problem of manipulation where some objects are grasped by robot grippers. One or several handles are attached to each object. $obj.handles$ denotes the set of handles of object obj (line 19). The algorithm loops over all possible combinations of "some grippers hold some handles" (Lines 7, 8) and creates a state for each combination. Method GETSTATE returns the state built with the combination of grasps f_1 given as input (Line 14). Then transitions are created from the new state to each state defined by the same set of grasps minus one (lines 11-15).

Function MAKESTATE loops over all ungrasped objects and creates constraints so that those objects stay in stable contact pose (Line 19). Then it loops over each grasp defined by f_1 (Line 23) and creates the corresponding constraint. Lines 22 and 27 store foreach state a set of parameterized numerical constraints that will be inserted in edges by function MAKETRANSITIONS.

Algorithm 1 Build the constraint graph

```
1:  G ← set of grippers,
2:  H ← set of handles,
3:  O ← set of objects (with contact polygons),
4:  P ← set of contact polygons of the environment
5:  function BUILDCONSTRAINTGRAPH
6:      states ← ∅
7:      for all subset G' of G by increasing cardinal do
8:          for all injective mapping f_1 from G' to H do
9:              S_1 ← MAKESTATE(f_1, G')
10:             states ← states ∪ {S_1}
11:             for all g_1 ∈ G' do
12:                 G'' ← G' \ {g_1}
13:                 f_0 ← f_1 restricted to G''
14:                 S_0 ← states.GETSTATE(f_0)
15:                 MAKETRANSITIONS(S_0, S_1)
16:             MAKETRANSITIONS(S_1, S_1)
17: function MAKESTATE(f_1, G')
18:     S ← EMPTYSTATE
19:     for all obj ∈ O | f_1(G') ∩ obj.handles = ∅ do
20:         C_contact ← CONTACT(obj.polygons, P)
21:         S.constraints.ADD(C_contact)
22:         S.transConstr.ADD(FIXED(obj, env))
23:     for all g_1 ∈ G' do
24:         C_grasp ← GRASP(g_1, f_1(g_1))
25:         S.constraints.ADD(C_grasp)
26:         if C_grasp.dimension < 6 then
27:             S.transConstr.ADD(FIXED(f_1(g_1).obj, g_1))
28:     return S
29: function MAKETRANSITIONS(S_0, S_1)
30:     T_0 ← EMPTYTRANSITIONBETWEEN(S_0, S_1)
31:     T_1 ← EMPTYTRANSITIONBETWEEN(S_1, S_0)
32:     T_0.constraints.ADD(S_0.transConstr)
33:     T_1.constraints.ADD(S_1.transConstr)
```

III. MANIPULATION PLANNING

In this section, we explain how we use the previously built contraint graph to plan manipulation paths in order to solve the crossed foliation issue.

The number of nodes of the constraint graph may quickly grow when the number of objects and grippers increases. The user may provide as input only a subset of relevant grasps to algorithm 1 so that the result remains tractable. This subset can be provided as a *grasp-placement table* as in [24], [10].

A. Exploration using the constraint graph

For each state of the constraint graph, we define a probability distribution over all transitions starting from this state. By default the uniform distribution is a reasonable option.

Then our manipulation planning algorithm consists in exploring the transitions of the constraint graph as follows.

1) shoot a random configuration $\mathbf{q}_{rand}$,

2) find the closest node $\mathbf{q}_{near}$ in the current roadmap,
3) find the state of this node in the constraint graph,
4) sample a transition getting out of this state,
5) extend $\mathbf{q}_{near}$ along the transition up to $\mathbf{q}_{new}$ following Algorithm 2,
6) try to connect $\mathbf{q}_{new}$ to other connected components of the roadmap (Algorithm 3).

Algorithm 2 Constrained extension

```
1: function CONSTRAINEDEXTEND(q_near, q_rand, T)
2:     S ← T.TARGETNODE
3:     g_t ← T.CONSTRAINT
4:     f_s ← S.CONSTRAINT
5:     proj ← PROJECTOR(f_s = 0, g_t = g_t(q_near))
6:     q_proj ←proj(q_rand)
7:     p_1 ← INTERPOLATE(q_near, q_proj)
8:     pathProj ← PROJECTOR(g_t = g_t(q_near))
9:     p_2 ← PROJECTPATH(p1, pathProj)
10:    p_3 ← TESTCOLLISION(p_2)
11:    q_new ← FINALCONFIGURATION(p_3)
12:    return q_new, p_3
```

Algorithm 2 describes Step 5 above. g_t denotes the *parameterized numerical constraint* of transition T (Line 3). f_s denotes the *numerical constraint* of target state S (Line 4). g right hand side is initialized with $\mathbf{q}_{near}$ (Line 5). $\mathbf{q}_{proj}$ is obtained by projecting $\mathbf{q}_{rand}$ (Line 6). Line 9, the path is projected, by discretizing and projecting on the transition constraint. The resulting path is tested for collision. In case of collision or projection failure, p_3 is the part of the path that was successfully projected and collision-free. After calling the constrained extension algorithm, $\mathbf{q}_{new}$ is added as a new roadmap node, and p_3 is added as a new roadmap edge starting from $\mathbf{q}_{near}$.

Algorithm 3 Connect configurations along a transition

```
1: function CONNECT(q_1, q_2)
2:     S_1 ←STATE(q_1)
3:     S_2 ←STATE(q_2)
4:     T ←TRANSITION(S_1, S_2)
5:     if T is None then return failure
6:     g ← T.CONSTRAINT
7:     if g(q_2) ≠ g(q_1) then return failure
8:     proj ← PROJECTOR(g = g(q_1))
9:     p_1 ← INTERPOLATE(q_near, q_proj)
10:    p_2 ← PROJECTPATH(p1, proj)
11:    p_3 ← TESTCOLLISION(p_2)
12:    if no collision and projection success then return p_3
13:    else return failure
```

Algorithm 3 tries to connects new nodes created by Algorithm 2 to other connected components of the roadmap, as in any classical implementation of RRT algorithm. First the function looks for a transtion between the configurations. If a transition exists, the function checks that $\mathbf{q}_1$ and $\mathbf{q}_2$ are on the same leaf of the transition foliation. If not the function returns failure.

As explained in the introduction, if states S_1 and S_2 are foliated, Line 7 will always return failure.

B. Crossed foliation transition

The crossed foliation issue arises when two transitions connecting two nodes back and forth are both foliated. In Algorithm 1, lines 32 and 33, it corresponds to the parameterized constraints $S_1.transConstr$ and $S_2.transConstr$ to be both non-empty.

In this case, as explained in Figure 2, the exploration algorithm described in Section III-A will never succeed in connecting configurations from different trees.

To solve this issue, we add between S_1 and S_2 a new type of transition called *crossed foliation transition*. This type of transition stores the parameterized numerical constraints of transition $S_1 \rightarrow S_2$ as g_1, and the parameterized numerical constraints of transition $S_2 \rightarrow S_1$ as g_2.

When extending a node belonging to S_1 through this transition, the extension algorithm picks a random configuration $\mathbf{q}_1$ in another connected component of the current roadmap and lying in state S_1. Constraint $g_2 = q_2(\mathbf{q}_1)$ is then added to the projector (Line 5 of Algorithm 2). As a consequence, $\mathbf{q}_{proj}$ is on the same leaf as $\mathbf{q}_1$ for the transition linking S_2 to S_1. As a consequence, Algorithm 3 may connect these configurations.

When adding crossed foliation transition to the graph, we need to redefine the probability distribution of edges getting out of each node. Thus, when extending a node of a state, the algorithm alternates between regular extension and crossed foliation extension.

IV. Experimental results

In this section, we show some experimental results demonstrating the effectivity of our algorithm. Algorithms are implement in an open source software called HPP [14]: https://humanoid-path-planner.github.io/hpp-doc. Experiments are run on a standard 2.4GHz, RAM 4Go, 4 cores, desktop computer.

A. Romeo holding a placard

In this example, Romeo holds a placard. The manipulation planning problem consists in rotating the placard around the vertical axis. To do so, the robot needs to go through a sequence of states where the robot holds the placard by the right hand, the left hand and both hands. Applying a task planning based approach for this problem seems difficult since the minimal sequence of tasks highly depends on the workspace of the robot arms and is very difficult to precompute.

The results obtained after 20 runs of our algorithms are displayed in Table I. The variance of the computation time is surprisingly high. We chose not to interrupt path planning after a threshold time as usually done for difficult problems. Note that the model of the robot is very accurate (each hand has four fingers with 3 segments) and a lot of time

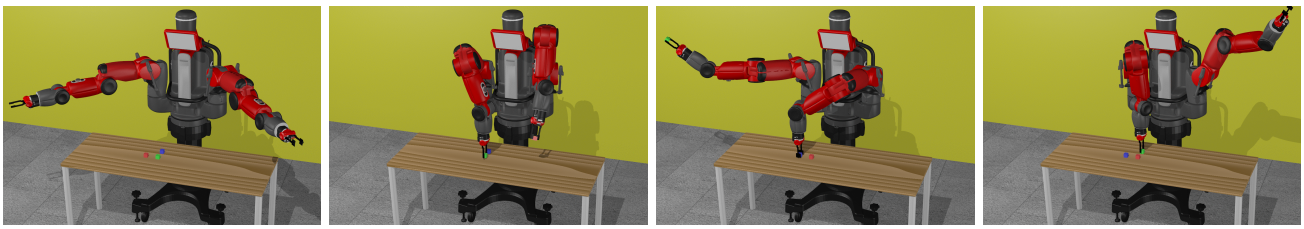

Fig. 4: Baxter permutes the position of 3 boxes.

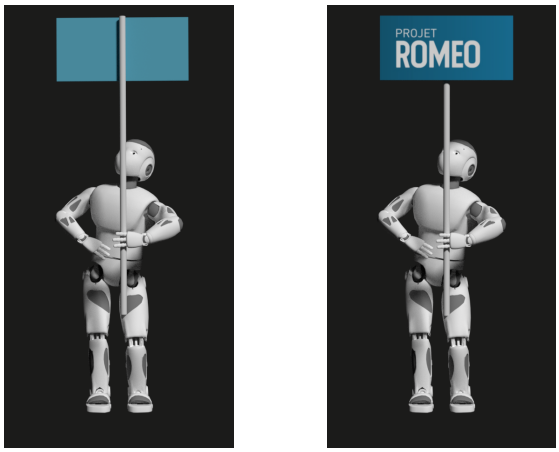

Fig. 5: Humanoid robot Romeo holding a placard. In final configuration, the placard is rotated by 180 degrees. The manipulation planning algorithm needs to explore manifolds defined by right, left and both hand grasps.

	Min	Median	Max
Number of nodes	42	1370	7002
Time of computation (s)	19	880	6500

TABLE I: Romeo holding a placard

is spent in collision checking. Moreover, the solution path goes through several "narrow passages" since grasping the pole requires to avoid a lot of collisions between the object and the fingers. Half of the time, the problem is solved in less than 15 minutes. The accompanying video shows a path solving the manipulation problem.

B. Rearrangement planning

On these 4 test cases, Baxter robot moves boxes on a table. In cases 1 and 2, 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 II, the accompanying video and Figure 4 summarizes the results. 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.

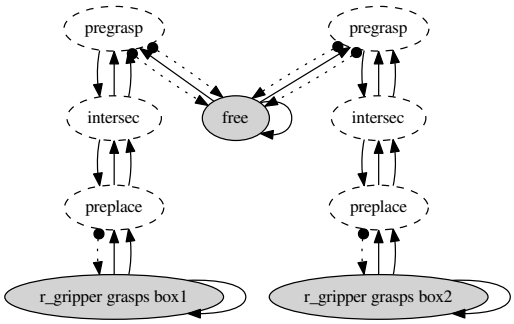

Fig. 6: Constraint graph for Case 1 with Baxter robot: 2 boxes and considering only the right arm. The constraint graph is produced by an extension of Algorithm 1 that inserts waypoint corresponding to approaching positions of the gripper in front of the object.

Case 1 shows that our approach is not as efficient as task planning based approaches 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.

C. Grasping behind a door

We demonstrate here the implementation of a more complex manipulation planning problem. A mobile (humanoid) robot has to 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 robot. Then the path can be post processed [25], [26] to

			Time (s)			Nb nodes		
Case	Nb	Arm	Min	Med	Max	Min	Med	Max
1	2	Right	0.15	1.4	3.5	11	55	141
2			1.1	4.6	10.6	42	199	482
3		Both	3.7	18	60	103	273	832
4	3		76	397	1028	664	3659	8830

TABLE II: Minimum / median / maximum solving time and number of nodes, over 20 runs, for various cases with Baxter robot. "Nb" represents the number of boxes.

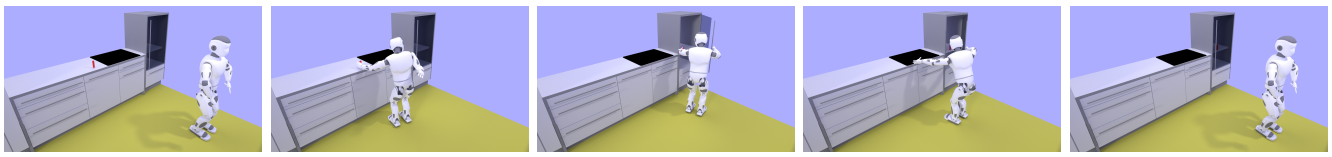

Fig. 7: Romeo puts a box in a fridge.

generate a walking trajectory. We successfully planned a path where Romeo robot takes an object and puts it in a fridge. The solution found is included in the video and summarized in Figure 7.

V. Conclusion

The main contribution of this paper is the introduction of the *constraint graph* as a tool to model and solve manipulation planning problems. We propose an algorithm using this tool to explore the configuration space of the system, but we truly think that many existing manipulation planning approaches (task planning based, multi-arm manipulation, navigation among movable obstacles,...) could be implemented using this tool.

The purely geometric algorithm that we propose in this paper is certainly not the most efficient on any benchmark, but it is the most general that we know about.

Future improvements include solving explicit constraints in an explicit way: when an object is rigidly fixed to a gripper, the position of the object depends on the position of the gripper. Expressing this as a numerical constraint is unefficient. Better handling collision-checking by not testing collision for objects that are fixed together will also improve performance.

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] R. Alami, T. Simon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps," in *5th International Symposium on Robotics Research*, Tokyo, Japan, 1989.

[3] 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.

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

[5] 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.

[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] 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.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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 *IEEE/RSJ Intelligent Robots and Systems*, October 2016.

[15] 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.

[16] S. Jentzsch, A. Gaschler, O. Khatib, and A. Knoll, "MOPL: A multi-modal path planner for generic manipulation tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, September 2015, http://youtu.be/1QRvjBw58bU.

[17] 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.

[18] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *International Journal of Robotics Research*, vol. 28, 2009.

[19] J. Barry, L. Kaelbling, and T. Lozano-Pérez, "A hierarchical approach to manipulation with diverse actions," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

[20] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *Workshop on the Algorithmic Foundations of Robotics*, 2014.

[21] 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.

[22] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu, "Hybrid reasoning for geometric rearrangement of multiple movable objects on cluttered surfaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, 2014.

[23] M. Vendittelli, J.-P. Laumond, and B. Mishra, *Algorithmic Foundations of Robotics XI*, ser. Tracks in Advanced Robotics. Springer, 2015, vol. 107, ch. Decidability of Robot Manipulation Planning: Three Disks in the Plane, pp. 641–657.

[24] P. Tournassoud, T. Lozano-Prez, and E. Mazer, "Regrasping," in *IEEE International Conference on Robotics and Automation*, vol. 4, 1987, pp. 1924–1928.

[25] 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.

[26] J. Carpentier, S. Tonneau, M. Naveau, O. Stasse, and N. Mansard, "A versatile and efficient pattern generator for generalized legged locomotion," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 3555–3561.