Manipulation planning: addressing the crossed foliation issue
Joseph Mirabel, Florent Lamiraux

To cite this version:
Joseph Mirabel, Florent Lamiraux. Manipulation planning: addressing the crossed foliation issue. IEEE International Conference on Robotics and Automation 2017, May 2017, Singapour, Singapore. hal-01358767

HAL Id: hal-01358767
https://hal.archives-ouvertes.fr/hal-01358767
Submitted on 4 Sep 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Manipulation planning: addressing the crossed foliation issue

Joseph Mirabel\textsuperscript{1,2} and Florent Lamiraux\textsuperscript{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 an 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.

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 $C \triangleq [-\pi, \pi]^6 \times SE(3)$. 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.

\textsuperscript{1}CNRS, LAAS, 7 avenue du colonel Roche, F–31400 Toulouse, France

\textsuperscript{2}Univ de Toulouse, LAAS, F–31400 Toulouse, France
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 ∩ 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 $C$ 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 $C$ to $\mathbb{R}^m$ where $m$ is an integer. $q \in C$ is said to satisfy the constraint iff

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

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

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

$$g(q) = g_0$$

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 $C$ to $C$ such that for any $q$ in the input subspace, $proj(q)$ 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) $q_0 \in S_0$, $q_1 \in S_1$

2) a path $\gamma$ from $[0, 1]$ to $C$ such that

$$\gamma(0) = q_0 \quad \text{and} \quad \gamma(1) = q_1$$

$$\forall t \in [0, 1], \ g(\gamma(t)) = g(q_0)$$

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 \( P_1 \) and \( P_2 \) 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

\[
\text{FIXED}(\text{obj}, g)
\]

where \( \text{obj} \) 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. \( \text{obj.handles} \) denotes the set of handles of object \( \text{obj} \) (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 \( \text{GETSTATE} \) 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 \( \text{MAKESTATE} \) 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 \( \text{MAKETRANSITIONS} \).

III. Manipulation planning

In this section, we explain how we use the previously built constraint 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 grasps-configuration 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 \( \text{q.rand} \).
2) find the closest node \( q_{\text{near}} \) 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 \( q_{\text{near}} \) along the transition up to \( q_{\text{new}} \) following Algorithm 2,
6) try to connect \( q_{\text{new}} \) to other connected components of the roadmap (Algorithm 3).

Algorithm 2 Constrained extension

1: function \text{CONstrainedEXTEND}(q_{\text{near}}, q_{\text{rand}}, T)
2: \( S \leftarrow T.\text{TARGET NODE} \)
3: \( g_t \leftarrow T.\text{CONSTRAINT} \)
4: \( f_s \leftarrow S.\text{CONSTRAINT} \)
5: \( \text{proj} \leftarrow \text{PROJECTOR}(f_s = 0, g_t = g_t(q_{\text{near}})) \)
6: \( q_{\text{proj}} \leftarrow \text{proj}(q_{\text{rand}}) \)
7: \( p_1 \leftarrow \text{INTERPOLATE}(q_{\text{near}}, q_{\text{proj}}) \)
8: \( \text{pathProj} \leftarrow \text{PROJECTOR}(g_t = g_t(q_{\text{near}})) \)
9: \( p_2 \leftarrow \text{PROJECTPATH}(p_1, \text{pathProj}) \)
10: \( p_3 \leftarrow \text{TESTCOLLISION}(p_2) \)
11: \( q_{\text{new}} \leftarrow \text{FINALCONFIGURATION}(p_3) \)
12: return \( q_{\text{new}}, p_3 \)

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). The right-hand side is initialized with \( q_{\text{near}} \) (Line 5). \( q_{\text{proj}} \) is obtained by projecting \( q_{\text{rand}} \) (Line 6). 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, \( q_{\text{new}} \) is added as a new roadmap node, and \( p_{3} \) is added as a new roadmap edge starting from \( q_{\text{near}} \).

Algorithm 3 Connect configurations along a transition

1: function \text{CONNECT}(q_1, q_2)
2: \( S_1 \leftarrow \text{STATE}(q_1) \)
3: \( S_2 \leftarrow \text{STATE}(q_2) \)
4: \( T \leftarrow \text{TRANSITION}(S_1, S_2) \)
5: if \( T \) is None then \text{return} failure
6: \( g \leftarrow T.\text{CONSTRAINT} \)
7: if \( g(q_2) \neq g(q_1) \) then \text{return} failure
8: \( \text{proj} \leftarrow \text{PROJECTOR}(g = g(q_1)) \)
9: \( p_1 \leftarrow \text{INTERPOLATE}(q_{\text{near}}, q_{\text{proj}}) \)
10: \( p_2 \leftarrow \text{PROJECTPATH}(p_1, \text{proj}) \)
11: \( p_3 \leftarrow \text{TESTCOLLISION}(p_2) \)
12: if no collision and projection success then \text{return} \( p_3 \)
13: else \text{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 transition between the configurations. If a transition exists, the function checks that \( q_1 \) and \( q_2 \) 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 12 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 \( S_1.\text{transConstr} \) and \( S_2.\text{transConstr} \) 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 \( q_1 \) in another connected component of the current roadmap and lying in state \( S_1 \). Constraint \( g_2 = g_2(q_2) \) is then added to the projector (Line 5 of Algorithm 2). As a consequence, \( q_{\text{proj}} \) is on the same leaf as \( q_1 \) 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 effectiveness of our algorithm. Algorithms are implemented in an open-source software called HPP [14]: https://humanoid-path-planner.github.io/hpp-docs/

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...
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.

<table>
<thead>
<tr>
<th>Case</th>
<th>Nb</th>
<th>Arm</th>
<th>Time (s)</th>
<th>Nb nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Left</td>
<td>0.15</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td>1.1</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Both</td>
<td>3.7</td>
<td>103</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>Both</td>
<td>7.6</td>
<td>664</td>
</tr>
</tbody>
</table>

**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.

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...
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 inefficient. Better handling collision-checking by not testing collision for objects that are fixed together will also improve performance.

REFERENCES