A Single-Query Manipulation Planner

Lertkultanon, Puttichai; Pham, Quang-Cuong

2015

Lertkultanon, P., & Pham, Q.-C. (2016). A Single-Query Manipulation Planner. IEEE Robotics and Automation Letters, 1(1), 198-205.

https://hdl.handle.net/10356/84985

https://doi.org/10.1109/LRA.2015.2513731

A Single-Query Manipulation Planner

Puttichai Lertkultanon and Quang-Cuong Pham

***Abstract*— In manipulation tasks, a robot interacts with movable object(s). The configuration space in manipulation planning is thus the Cartesian product of the configuration space of the robot with those of the movable objects. It is the complex structure of such a "composite configuration space" that makes manipulation planning particularly challenging. Previous works approximate the connectivity of the composite configuration space by means of discretization or by creating random roadmaps. Such approaches involve an extensive pre-processing phase, which furthermore has to be re-done each time the environment changes. In this paper, we propose a high-level Grasp-Placement Table similar to that proposed by Tournassoud et al. (1987), but which does not require any discretization or heavy pre-processing. The table captures the potential connectivity of the composite configuration space while being specific only to the movable objects: in particular, it does not require to be re-computed when the environment changes. During the query phase, the table is used to guide a tree-based planner that explores the space systematically. Our simulations and experiments show that the proposed method enables improvements in both running time and trajectory quality as compared to existing approaches.**

I. Introduction

Automated robotic assembly requires the ability to plan *pick-and-place* motions : pick an object from a given configuration (position and orientation) and place it in a desired configuration, possibly with a desired grasp. A grasp refers to a relative transformation between the gripper and the object. When none of the possible grasps for picking up the object at its initial configuration is compatible with subsequent operations, the robot needs to change its grasp, possibly several times, until the desired grasp can be attained. Unlike multi-fingered hands, parallel grippers (which are the most common and robust grippers in the industry) cannot realize in-hand manipulations. To change a grasp with a parallel gripper, the robot must *transfer* the object to some intermediate stable placement, then *transit* to a new grasp. Those operations are collectively termed "regrasping" [1].

Since a manipulation planning problem involves both the robot and the object configurations, searching for a sequence of transit and transfer paths, so-called a *manipulation path*, has to be done in the *composite configuration space* $\mathcal{C}$ [2], which is the Cartesian product of the robot configuration space $\mathcal{C}_{\text{robot}}$ and the object configuration space $\mathcal{C}_{\text{object}}$. A transit path, along which the robot moves alone while the object remains at a stable placement, lies in $\mathcal{P}$, the subset of $\mathcal{C}$ corresponding to stable placements of the object. Similarly, a transfer path, along which the robot moves while grasping the object with a valid grasp, lies in $\mathcal{G}$, the subset of $\mathcal{C}$ corresponding to valid grasps. The subset $\mathcal{G} \cap \mathcal{P}$ is central in solving manipulation planning problems since it is the place where transitions between transit and transfer paths may occur [2], [3]. Manipulation planners differ mainly in the way they explore and capture the connectivity of $\mathcal{G} \cap \mathcal{P}$.

For a parallel gripper grasping a polyhedral object, all the valid grasps and stable placements can be categorized into a finite number of classes. A grasp class corresponds to e.g. a pair of object surfaces the fingers are touching. A placement class corresponds to e.g. a surface (of the object or of its convex hull) in contact with the table. Each grasp and placement class may be further parameterized by a set of continuously varying parameters.

In a pioneering work, Tournassoud *et al.* [1] started by discretizing $\mathcal{G} \cap \mathcal{P}$[1] and then searched for a feasible regrasping sequence by backward chaining from the goal grasp. However, because of the high dimensionality of $\mathcal{C}$ and the complexity arising from the discretization, the authors had to constrain each grasp and placement class to have only single varying parameter. Their method is therefore limited and much dependent on the particular choice of the parameters. Nevertheless, they did introduce the important notion of *Grasp-Placement Table*, which captures part of the connectivity of $\mathcal{G} \cap \mathcal{P}$. This Grasp-Placement Table can be seen as an instance of a Manipulation Graph [3] that is particularly adapted for polyhedral objects. The Grasp-Placement Table is a grid where each vertical line represents a placement class while each horizontal line represents a grasp class. Intersections of vertical lines with horizontal lines then correspond to subsets of $\mathcal{G} \cap \mathcal{P}$. A transfer path appears as a connection between intersections on the same horizontal line, whereas a transit path appears as a connection between intersections on the same vertical line.

Here we propose a method to construct a *high-level* Grasp-Placement Table (or graph). In contrast with [1], our graph does not require any discretization or heavy pre-processing. Moreover, it is specific only to the movable object, and not to the environment or to the robot : it does not therefore require to be re-computed when the environment changes. At the query phase, the graph is used as a high-level task planner to guide the manipulation planner in exploring $\mathcal{C}$.

Specifically, the edges of the graph are generated by ignoring all the robot kinematics. Verification of kinematic feasibility along each edge of the graph is postponed to the

This work was partially supported by Tier 1 grant RG109/14 awarded by Singapore MOE to the second author.

Puttichai Lertkultanon and Quang-Cuong Pham are with the School of Mechanical and Aerospace Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798 `L.Puttichai@gmail.com`

[1]The notion of composite configuration space was not introduced at the time. However, their idea of grasp and placement spaces bears a close resemblance to the formulation presented in [2].

planning phase. In doing so, our method enables handling the full parameterization of $\mathcal{G} \cap \mathcal{P}$. Although the idea of delaying IK computation has been independently explored in a recent work [4] to construct a manipulation graph, the authors' graph still requires discretizing grasp and placement classes, entailing the same problem of heavy pre-processing.

By constructing and using a high-level Grasp-Placement Table, we decouple a pick-and-place manipulation planning problem into two layers of planning. The *high-level* task planning layer consists in finding a sequence of $\mathcal{G} \cap \mathcal{P}$ configurations that answers the query, while the *low-level* motion planning layer consists in finding *actual* motions between the $\mathcal{G} \cap \mathcal{P}$ configurations. Since the emphasis of this paper is on the high-level planning, *i.e.*, how we construct and use a high-level Grasp-Placement Table to help solve a manipulation query, addressing uncertainty, which takes place at the second layer, is beyond the scope of this paper.

Note also that we focus here on industrial assembly, where parallel-jaw grippers are pervasive owing to their cost-effectiveness and ease of integration. We first specifically consider the case when movable objects are either boxes or composed of boxes. These properties enables an efficient parameterizations of grasps and placements. We discuss extension to broader types of objects in Section VI-A.

The rest of this paper is organized as follows. In Section II, we briefly review related literature. In Section III, we present definitions and conventions that will be used in the sequel. Section IV introduces the high-level Grasp-Placement Table, its construction, and its use in planning. Comparisons between the proposed planner and other manipulation planners are presented in Section V. Finally, Section VI offers a brief discussion and sketches future research directions.

II. Related Literature

A. Manipulation Planning

Manipulation planners can be seen as a generalization of motion planners where the robot is allowed to displace specific objects, called movable objects, in its environment via specific interactions, e.g., pushing or grasping. Early work considered pick-and-place manipulation planning as a *regrasping problem* [1], [5], [6]. The authors discretized $\mathcal{G} \cap \mathcal{P}$ and checked at each discretized point whether the placement was stable and the grasp was feasible. They then executed a deterministic search of a regrasping sequence on the set of feasible placements and grasps. Evaluation of object placements based on minimum tipping energy was proposed in [5].

A recent work on regrasping algorithms [4] also utilized a discretization of $\mathcal{G} \cap \mathcal{P}$ to construct a regrasp graph, which represents connectivity of $\mathcal{G} \cap \mathcal{P}$ connected components, to analyze workcell utility. The authors improved efficiency of their algorithm by delaying IK computation in the graph construction until it is necessary. However, the discretization cost still remains considerable compared to our method of construction.

Most later work is based on the notion of Manipulation Graph [3] and composite configuration space [2]. Initially the planners usually assumed discrete sets of grasps and placements [3], [7], [8]. Later work extended the approach to handle continuous representation [2]. The approach has also been extended to facilitate bimanual manipulation planning [9]. Apart from permitting only two *modes* of motions, transit and transfer, there is also work on generalization that permitting finding solution paths with multiple modes of motions [10], [11].

Although a majority of manipulation planners are sampling-based, there also exist deterministic manipulation planners [12], [13]. These planners construct a lattice graph representing a discretized configuration space and then use heuristic searches to find solution paths. However, these planners are currently capable of planning only single-mode motions such as reach-to-grasp motions or motions once the robot has already grasped an object.

B. Object Rearrangement

Another problem related to manipulation planning is object rearrangement problem [14], [15]. The problem consists in finding a rearrangement path such that all movable objects are moved by the robot to their goal poses. Generally the problem can be decomposed into two subproblems. The first problem is finding a sequence of object rearrangements and the second one is finding manipulation paths connecting adjacent rearrangements. Here pick-and-place planners can serve as local planners to connect nodes of object rearrangements.

C. Task and Motion Planning

Planning pick-and-place motions can also be considered in a framework of task and motion planning [16], [17], [18], [19]. A planning problem is solved through two layers of planning : high-level task planning and low-level motion planning. A task planner executes symbolic searches for high-level actions such as *pick*, *move*, and *place*, not considering geometries nor kinematics. A motion planner then computes actual commands to follow the strategy—a sequence of actions—given by the task planner.

From the perspective of task and motion planning, instead of planning tasks by a symbolic task planner, we generate task plans by using information provided by our high-level Grasp-Placement Table. A grasp search algorithm is used to extract task plans, which will then inform the manipulation planner about how to make transitions between $\mathcal{G} \cap \mathcal{P}$ connected components. This makes the overall planner less likely to generate solutions with redundant grasp and ungrasp operations.

III. Definitions and Convention

In this section, we present the definitions and conventions involved in manipulation planning. The terminology introduced here will facilitate the discussions in the sequel.

A. Mathematical Definitions

We represent a composite configuration with a couple $(\boldsymbol{q}, \boldsymbol{T})$, where $\boldsymbol{q} \in \mathcal{C}_{\text{robot}} \subseteq \mathbf{R}^n$ is a robot configuration, n is the degrees-of-freedom (DOF) of the robot, and $\boldsymbol{T} \in SE(3)$[2] is a homogeneous transformation matrix of the object. For a composite configuration in $\mathcal{G}$ we define a grasp as a vector of parameters describing the relative transformation of the gripper and the object. Therefore, a grasp can generally be described by a vector of 7 parameters of quaternion and translation, which can be uniquely identified from $\boldsymbol{q}$ and $\boldsymbol{T}$. The number of parameters, however, can be reduced via grasp parameterization. An example of grasp parameterization can be found in [20].

A **single-mode path** is defined as a continuous function P from the unit interval $[0, 1]$ to a level set of $\mathcal{G}$ or $\mathcal{P}$. There are two types of single-mode paths : transit and transfer. A **transit path** maps the unit interval into a level set of $\mathcal{P}$, *i.e.*, for any two configurations $(\boldsymbol{q}_1, \boldsymbol{T}_1)$ and $(\boldsymbol{q}_2, \boldsymbol{T}_2)$ on the path, $\boldsymbol{T}_1 = \boldsymbol{T}_2$. A **transfer path** maps the unit interval into a level set of $\mathcal{G}$, *i.e.*, a grasp remains constant along the path.

Next, we define a binary operation called **composition** operation. First of all, the composition of two single-mode paths P_1 and P_2 is defined as

$$(P_1 * P_2)(s) = \begin{cases} P_1(2s/t) & 0 \le s \le t/2, \\ P_2(2s/t - 1) & t/2 \le s \le t, \end{cases}$$

where $t = 1$ if both paths are of the same type and $t = 2$ otherwise. Note that the composition operation is only defined when $P_1(1) = P_2(0)$. From the above definition, when P_1 and P_2 have the same type, $M = P_1 * P_2$ is also a single-mode path. We use $|M|$ to denote the **parameterization domain length** or **domain length** of M. Note that all single-mode paths have unit domain length.

Let $\prod_{i=a}^{b} P_i = P_a * P_{a+1} * \ldots * P_b$, where $b \ge a$. We can define the composition of k single-mode paths as

$$\left(\prod_{i=1}^{k} P_i\right)(s) = \begin{cases} (P_1 * P_2)(s) & 0 \le s \le t_2 \\ M_2(s - t_2) & t_2 \le s \le |M_2| + t_2 \end{cases}$$

$$= \begin{cases} M_1(s) & 0 \le s \le |M_1| \\ (P_{k-1} * P_k)(s - |M_1|) & |M_1| \le s \le |M_1| + t_1, \end{cases}$$

where $M_1 = \prod_{i=1}^{k-2} P_i$, $M_2 = \prod_{i=3}^{k} P_i$, and t_1 and t_2 are one if both single-mode paths are of the same type and two otherwise. Notice that the composition operation is associative but not commutative.

According to the above definition of the composition operation, we can now define a **manipulation path** as a composition of single-mode paths. A manipulation path is **irreducible** if no two consecutive single-mode paths are of the same type, *i.e.*, it is an alternating composition of transit and transfer paths. Any composition $\prod_{i=1}^{k} P_i$ can always be written it as an irreducible manipulation path $\prod_{i=1}^{l} P'_i$ with $l \le k$. For an irreducible manipulation path M, we have that the number of **transitions** (between transit and transfer paths or vice versa) is $|M| - 1$.

[2]$SE(3)$ is the special Euclidean group of rigid body transformations.

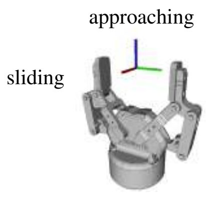


Fig. 1: The parallel gripper used in this paper with its local frame. The lateral direction is orthogonal to both finger surfaces. The sliding direction is parallel to both finger surfaces and is defined such that the approaching direction is pointing out of the gripper.

B. Placement Classes and Grasp Classes

The sets $\mathcal{P}$ and $\mathcal{G}$ can be partitioned into finite disjoint classes called placement classes and grasp classes, respectively. A class groups together composite configurations with similar properties. Each placement class indicates how an object is placed on a table, and thus normally refers to a surface of the object's convex hull being in contact with the table. Similarly, each grasp class indicates how the object is grasped by the gripper. In our case, with an assignment of the gripper's local frame as shown in Fig. 1, each grasp class refers to the relative direction of the gripper's approaching direction with respect to the object.

Consider a gripper grasping a box. We say that the approaching direction is $+x$ if the positive direction of the gripper's approaching axis is aligned with $+x$-direction of the box's local frame. Therefore, there are 6 possible directions along which the gripper can approach the box[3]. For convenience, we will use integers 1 to 6 to denote approaching directions $+x, +y, +z, -x, -y, -z$, respectively. Now consider the case when the object is composed of m boxes. Suppose we index those boxes with integers from 1 to m. There will be in total $6m$ possible grasp classes. When the gripper is approaching the object in the direction i of the box j, the grasp class index is $i + 6(j - 1)$. For example, in the grasp class 7, the gripper is grasping box 2 and the approaching axis is aligned with $+x$-axis of the box.

IV. Manipulation Planning Using High-Level Grasp-Placement Table

A. High-Level Grasp-Placement Table

1) Overview: A high-level Grasp-Placement Table (or graph) is an undirected, unweighted graph whose nodes represent different subsets of $\mathcal{G} \cap \mathcal{P}$. According to the partitioning of $\mathcal{G}$ and $\mathcal{P}$, the set $\mathcal{G} \cap \mathcal{P}$ is then partitioned into finite disjoint subsets. Each subset of $\mathcal{G} \cap \mathcal{P}$ is the intersection between a particular pair of placement and grasp classes. Therefore, a node in the graph can be represented by a pair of integers, a placement class index and a grasp class index. We visualize a high-level Grasp-Placement Table in the similar way as the authors of [1] did for their Grasp-Placement Table. Our Table is plotted on a two-dimensional grid. Each vertical line corresponds to a placement class

[3]This means the approaching axis of the gripper must be perpendicular to a surface of the box. A configuration that the approaching direction is not perpendicular to any surface can then be achieve by rotating the gripper about the sliding axis.

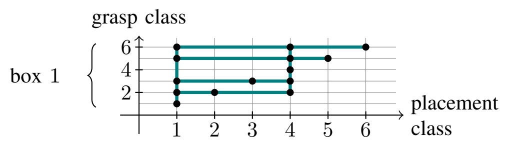


Fig. 2: An example of a high-level Grasp-Placement Table for a box of dimension 28.0 cm. $\times$ 4.9 cm. $\times$ 2.5 cm. For clarity, self-loops are not depicted. Note that every node on the same line is connected to all other nodes although we do not draw separate lines for them. For example, node $(1, 6)$ is reachable from $(6, 1)$ in one step.

whereas each horizontal line corresponds to a grasp class. Intersections of vertical with horizontal lines then represent subsets of $\mathcal{G} \cap \mathcal{P}$.

If there were no collision avoidance constraints and kinematic reachability constraints, any combination of an object contact surface and a gripper's approaching direction would have been possible. Any pair of nodes on the same vertical or horizontal line will be connected. Imposing those constraints makes some nodes and edges infeasible. However, verifying all constraints at the graph construction phase entails costly computations. Also, by checking kinematic reachability, the graph will become robot- and environment-dependent : it will need to be re-computed when the environment changes. Therefore, we propose to construct the graph by verifying only collision avoidance constraints (between the gripper and the table). Robot kinematic reachability constraint verification (IK computation) is postponed until the planning phase.

2) Graph Construction: In the first step, we check for feasibility of graph nodes. For each placement class, we place the object at a nominal location on the table. Then for each box composing the object, we check whether the gripper can approach the box from any of the 6 directions, *i.e.*, $+x$-, $+y$-, $+z$-, $-x$-, $-y$-, and $-z$-directions, of the box's local frame, without colliding with the table. If there is no collision, we add a node that represents the respective placement and grasp classes into the graph. We continue this procedure for all placement classes. Since we do not consider the robot kinematics here, the actual location of the object on the table, *i.e.*, how far the object is from the robot, does not affect the resulting graph.

After we obtain all feasible nodes in the first step, we display them on a grid. Then we connect every pair of nodes on the same vertical or horizontal line. These edges represent *potential* connections between nodes. An edge connecting a pair of nodes on the same vertical line represents *transit* paths while an edge connecting a pair of nodes on the same horizontal line represents *transfer* paths. Every node also has two self-loops. One loop corresponds to a transit path; the other loop corresponds to a transfer path. Fig. 2 shows an example of a high-level Grasp-Placement Table for a box of dimension 28.0 cm. $\times$ 4.9 cm. $\times$ 2.5 cm. For simplicity, we do not show self-loops in the plot.

The construction can be generalized to handle start and goal configurations in $\mathcal{G}$ and/or $\mathcal{P}$ by adding special nodes into the graph. To handle configurations in $\mathcal{P}$, we can add a grasp class index 0 to the graph. For example, in the above example of the box, we will obtain 6 new nodes, $(1, 0), (2, 0), \ldots, (6, 0)$. These nodes are connected to all other nodes with the same placement class. Note, however, that there are no horizontal edges between them since it is not possible to travel directly from one placement class to another in $\mathcal{P}$. To handle configurations in $\mathcal{G}$, we can add a placement class index 0, similarly to the previous case.

3) Guiding a Manipulation Planner via Task Plans: One of quality measures of manipulation paths is the number of transitions (as defined in Section III-A). Fewer transitions means fewer grasp/ungrasp operations and fewer single-mode paths. This may therefore lead to a shorter overall execution time.

From the constructed graph we can extract *task plans* of a specific length[4] which serve as a guide for the planner about how to travel between different subsets of $\mathcal{G} \cap \mathcal{P}$. The planner, instead of exploring randomly how to go from one subset of $\mathcal{G} \cap \mathcal{P}$ to another, will follow those plans to search for a manipulation path of a specific number of transitions. Here a **task plan** is a sequence of graph nodes connected by graph edges. For example, from Fig. 2 a task plan of length 3 to travel from $(6, 6)$ to $(2, 2)$ is given by $(6, 6) \rightarrow (4, 6) \rightarrow (4, 2) \rightarrow (2, 2)$.

The task plans extracted from the graph only provide high-level information on how to pick and place the object. Exact parameter values for each grasp and placement classes will be assigned by the planner, *i.e.*, via random sampling, in the planning phase (see Section IV-B.1).

B. Manipulation Planning Algorithm

1) Algorithm Details: Our pick-and-place manipulation planner proceeds in two phases : pre-processing phase and planning phase. In the pre-processing phase, it constructs a high-level Grasp-Placement Table based on the models of the object and of the gripper (the latter is needed for collision checking). Note that neither the kinematic model of the robot nor the environment is needed at this phase. Then this high-level Grasp-Placement Table is used in the planning phase to guide a bidirectional tree-based planner, similar to [21], to search for a manipulation path. The main algorithm is listed in Algorithm 1.

The main algorithm takes as its input a robot model $\mathcal{R}$; an object model $\mathcal{M}$; start and goal robot configurations $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$; start and goal object transformations $\boldsymbol{T}_{\text{start}}$ and $\boldsymbol{T}_{\text{goal}}$; and a maximum running time for the planner t_{max}. Details of functions in Algorithm 1 are listed below :

- `Preprocess` gathers geometric information of the gripper and the object to construct a high-level Grasp-Placement Table G, as described in Section IV-A.2.
- `Vertex` initializes a new tree vertex to store information of a composite configuration. Note that this function also examines G to find the graph node to which the input composite configuration belongs.
- `FindShortestPathLength` finds the length l of the shortest task plan(s) to go from the graph node c_{start} to the graph node c_{goal}. Here $l - 1$ serves as the lower bound of

[4]A task plan of length k has $k - 1$ transitions

Algorithm 1: Main algorithm

```
Main($\mathcal{R}, \mathcal{M}, \boldsymbol{q}_{\text{start}}, \boldsymbol{T}_{\text{start}}, \boldsymbol{q}_{\text{goal}}, \boldsymbol{T}_{\text{goal}}, t_{\max}$):
 1  $G \leftarrow$ Preprocess($\mathcal{R}, \mathcal{M}$)
 2  $v_{\text{start}} \leftarrow$ Vertex($\boldsymbol{q}_{\text{start}}, \boldsymbol{T}_{\text{start}}$)
 3  $v_{\text{goal}} \leftarrow$ Vertex($\boldsymbol{q}_{\text{goal}}, \boldsymbol{T}_{\text{goal}}$)
 4  $l \leftarrow$ FindShortestPathLength($G, v_{\text{start}}, v_{\text{goal}}$)
 5  $t \leftarrow 0, k \leftarrow l$, found $\leftarrow$ False
 6  $\Delta \leftarrow$ ChoosePathLengthIncrement($v_{\text{start}}, v_{\text{goal}}$)
 7  while ($t < t_{\max}$) and (not found) do
 8      $t_{\text{s}} \leftarrow$ GetTime()
 9      $\Pi \leftarrow$ FindPlansOfGivenLength($k, v_{\text{start}}, v_{\text{goal}}$)
10      $Q \leftarrow$ CreateDirectedGraph($\Pi$)
11      {found, $\pi$} $\leftarrow$
        PlanPath($Q, \mathcal{R}, \mathcal{M}, v_{\text{start}}, v_{\text{goal}}, t_{\max} - t$)
12      $t_{\text{e}} \leftarrow$ GetTime()
13      $t \leftarrow t + (t_{\text{e}} - t_{\text{s}})$
14      if found then
15          return $\pi$
16      $k \leftarrow k + \Delta$
17  return None
```

Algorithm 2: Planning phase

```
PlanPath($Q, \mathcal{R}, \mathcal{M}, v_{\text{start}}, v_{\text{goal}}, t_{\max}$):
 1  $\mathcal{T}_{\text{FW}} \leftarrow$ Tree($v_{\text{start}}$), $\mathcal{T}_{\text{BW}} \leftarrow$ Tree($v_{\text{goal}}$)
 2  $\mathcal{T}_{\text{a}} \leftarrow \mathcal{T}_{\text{FW}}$, $\mathcal{T}_{\text{b}} \leftarrow \mathcal{T}_{\text{BW}}$
 3  $t \leftarrow 0$
 4  while $t < t_{\max}$ or $Q$.haspath do
 5      $t_{\text{s}} \leftarrow$ GetTime()
 6      $v \leftarrow$ SampleTree($\mathcal{T}_{\text{a}}$)
 7      $Q \leftarrow$ RemoveInfeasibleEdges($Q$)
 8      result $\leftarrow$ ExtendFrom($v, Q$)
 9      if result == REACHED then
10          $\pi \leftarrow$ ExtractPath($\mathcal{T}_{\text{FW}}, \mathcal{T}_{\text{BW}}$)
11          return {True, $\pi$}
12      $t_{\text{e}} \leftarrow$ GetTime()
13      $t \leftarrow t + (t_{\text{e}} - t_{\text{s}})$
14      Swap($\mathcal{T}_{\text{a}}, \mathcal{T}_{\text{b}}$)
15  return {False, None}
```

the number of transitions of manipulation paths connecting $(\boldsymbol{q}_{\text{start}}, \boldsymbol{T}_{\text{start}})$ and $(\boldsymbol{q}_{\text{goal}}, \boldsymbol{T}_{\text{goal}})$.

- `ChoosePathLengthIncrement` chooses the suitable value of Δ for the query. If either $(\boldsymbol{q}_{\text{start}}, \boldsymbol{T}_{\text{start}})$ or $(\boldsymbol{q}_{\text{goal}}, \boldsymbol{T}_{\text{goal}})$ is in $\mathcal{G} \cap \mathcal{P}$, Δ can be 1. Otherwise, Δ has to be 2 in order to keep the resulting manipulation path irreducible.
- `CreateDirectedGraph` creates a directed graph Q from task plans in Π. A node of Q is encoded as a couple (d_i, c_i), where c_i is the corresponding node of G and d_i indicates the number of steps c_i is away from c_{start}[5].
- `PlanPath`, which acts as a motion planner, searches for a manipulation path according to information provided by Q. `PlanPath` is listed in Algorithm 2.
- `ExtendFrom` first randomly samples a new composite configuration in the grasp and placement classes suggested by Q. Then it will attempt to connect the composite configuration contained in v with the newly sampled one. The function returns `True` if the attempt is successful.
- `RemoveInfeasibleEdges` removes edges in Q which lead to more than N failed attempts by `ExtendFrom`, where N is a threshold value set by the user.

2) Example: Consider the task of moving the box, whose high-level Grasp-Placement Table is shown in Fig. 2, from the placement and grasp $c_{\text{start}} = (6,6)$ to $c_{\text{goal}} = (2,2)$. In the first planning loop in Algorithm 1, the algorithm will try to find a manipulation path of length $k = 3$. The set Π will store two task plans: $(6,6) \rightarrow (4,6) \rightarrow (4,2) \rightarrow (2,2)$ and $(6,6) \rightarrow (1,6) \rightarrow (1,2) \rightarrow (2,2)$. The directed graph Q, constructed from Π, will serve as a guidance for the planner as follows.

In each while loop in Algorithm 2, the planner will randomly pick an existing vertex v_{sample} on a tree. Suppose v_{sample} contains a composite configuration in placement and grasp classes $(4,6)$. From Q, the planner then knows that the next extension should be attempted towards a composite configuration in the placement and grasp classes $(4,2)$. `ExtendFrom` will then sample a composite configuration in that subset of $\mathcal{G} \cap \mathcal{P}$ and call a local planner, e.g., a bidirectional RRT planner [21], to attempt the connection.

C. Remarks

Remark 1: The high-level Grasp-Placement Table G, and hence Q, is constructed without considering kinematic constraints. Failed attempts by `ExtendFrom` are likely due to kinematic infeasibility. Therefore, we need to set a threshold N to prevent the planner from repetitively attempting infeasible connections.

This threshold also affects the running time of our planner. If we set N too low, feasible edges of Q can also be removed due to false negative reports of kinematic infeasibility. This will lead to obtaining solutions with redundant transitions. On the other hand, if the threshold is set too high, it will take longer time for the planner to declare an infeasible edge and hence longer overall running time.

Remark 2: We can tailor the sampling of vertices from a tree, done in `SampleTree`, by putting different sampling weight on different vertices. For example, we can use weights proportional to the level of c_{sample} in Q, *i.e.*, d_{sample}. In this way, vertices farther away from the root will be more likely to be selected.

Remark 3: Our planner tends to work better in a less constrained environment. In a very constrained environment[6] in which the robot needs to grasp and ungrasp the object a number of times before it can move the object to the final transformation, our planner will spend a considerable amount of time verifying infeasible edges. However, in

[5]We need to encode the level d_i of c_i into each node of Q since the node c_i may appear at different steps in different task plans. This helps the planner not to get lost when many task plans contain the same node c_i.

[6]This may be seen as a narrow passage in $\mathcal{C}$. An example of such cases is the problem considered in [2] where the robot needs to pull a bar out of a tight cage mounted on the floor.

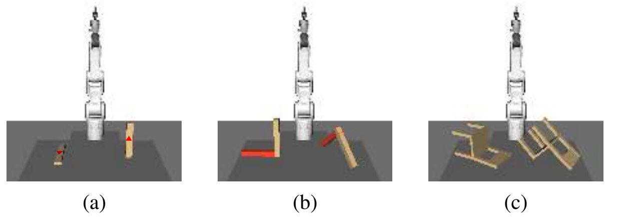


Fig. 3: Scenes used in our experiments. There are two identical (movable) objects shown in each figure. The objects on the left are at the initial transformations. The objects on the right are at the final transformations.

actual industrial settings, although the environment might be cluttered, it is usually not tightly constrained.

V. Results and Comparisons with Other Manipulation Planners

In this section, we present details of comparisons between our planner and two other planners : Primitive Manipulation Planner (Section V-B.1) and Discretization-Based Manipulation Planner (Section V-B.2). We implemented all planners in Python and used OpenRAVE [22] as a simulation environment. The local planner employed in all manipulation planners was the OpenRAVE built-in bidirectional RRT. The robot was a 6-DOF industrial manipulator Denso VS-060 equipped with a 2-finger Robotiq gripper 85. All simulations were run on a 3.2 GHz Intel® Core™desktop with 3.8 GB RAM.

A. Task Details

The planners had to plan pick-and-place motions for three objects : a box, an L-shaped object, and a small chair. The robot was to move from `Home`, *i.e.*, $\boldsymbol{q} = \boldsymbol{0}$, pick up the object at a given object transformation, place it at another given transformation, and finally return to `Home`. Snapshots of all three settings are shown in Fig. 3. The Grasp-Placement Tables of the L-shaped object and the chair are shown in Fig. 4(a) and Fig. 4(b), respectively. (The Grasp-Placement Table of the box is similar to the one in Fig. 2.) These tasks are useful especially for assembly operations, such as furniture assembly where the robot needs to grasp a furniture part at some placement and move it to some desired transformation to attach the part to other parts. We set chose start and goal object transformations such that the robot needed to perform regrasp operations at least once in order to complete the tasks.

B. Descriptions of Alternative Manipulation Planners

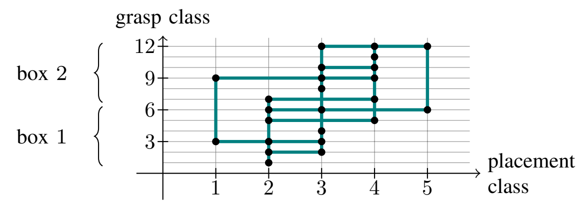
We implemented two alternative manipulation planners, each of which lies on opposite ends of the spectrum of Manipulation Graph construction. The first planner does not explicitly construct the graph while the second constructs the graph by means of discretization. Our planner lies midway between the two.

1) Primitive Manipulation Planner (PMP): PMP has minimal knowledge of the structure of $\mathcal{C}$ as it has no preprocessing stage. It explores $\mathcal{C}$ according to the transition diagram, similar to [23], shown in Fig. 5.

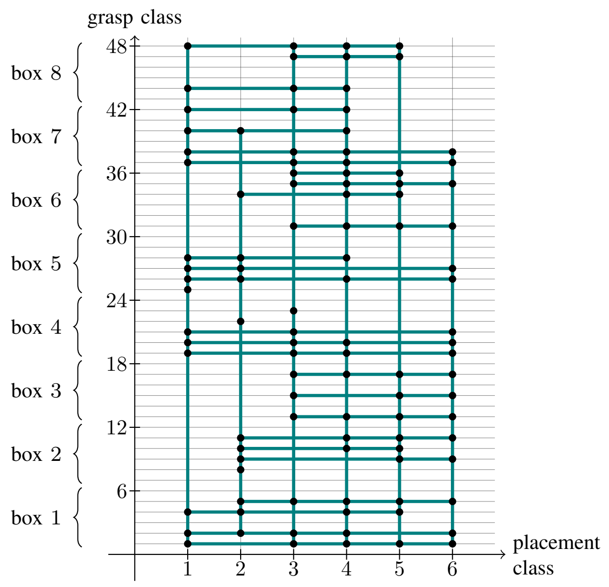
We implemented PMP as a bidirectional tree-based planner. It grows one tree rooted at $(\boldsymbol{q}_{\text{start}}, \boldsymbol{T}_{\text{start}})$, the other rooted at $(\boldsymbol{q}_{\text{goal}}, \boldsymbol{T}_{\text{goal}})$. In each iteration, it samples a new composite configuration and tries to connect it with existing vertices on a tree. For example, if the newly sampled configuration is in $\mathcal{G} \cap \mathcal{P}$ and is to be connected with a configuration in $\mathcal{P}$ on a tree, then the local planner will try connecting them with a transit path. For the distance metric used in our implementation, we defined a distance d between two configurations $(\boldsymbol{q}_1, \boldsymbol{T}_1)$ and $(\boldsymbol{q}_2, \boldsymbol{T}_2)$ as a weighted sum of the Euclidean distance between the robot configurations and a distance between the object transformations, *i.e.*, $d = \alpha\|\boldsymbol{q}_2 - \boldsymbol{q}_1\|^2 + (1-\alpha)w(\boldsymbol{T}_1, \boldsymbol{T}_2)$, where $0 \le \alpha \le 1$ and $w(\boldsymbol{T}_1, \boldsymbol{T}_2)$ is a weighted sum of the minimal geodesic distance between two rotations (see [24] for more detail) and the Euclidean distance between two displacements.

2) Discretization-Based Manipulation Planner (DBMP): In contrast with PMP, DBMP constructs its variant of Manipulation Graph, *two-layer regrasp graph* [4], by discretizing placements and grasps

We implemented DBMP following [4]. The planner starts by constructing the set of possible grasps by means of sampling. Then it builds a two-layer regrasp graph : the first



(a) The high-level Grasp-Placement Table for an L-shaped object



(b) The high-level Grasp-Placement Table for a small chair

Fig. 4: High-level Grasp-Placement Tables of objects used in our simulations.

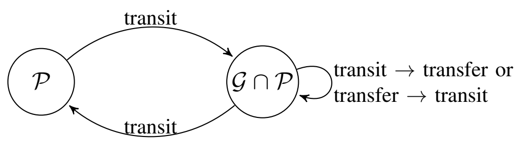


Fig. 5: A transition diagram used in Primitive Manipulation Planner.

TABLE I: Pre-Processing time, planning time, and numbers of transitions from three problems averaged over 100 runs.

	Problem 1 (Box)				Problem 2 (L-shaped object)				Problem 3 (Small chair)			
	prep. time (s.)	plan. time (s.)	# transitions	success rate	prep. time (s.)	plan. time (s.)	# transitions	success rate	prep. time (s.)	plan. time (s.)	# transitions	success rate
Our planner	0.24	12.18	4.0	100%	0.50	15.32	4.0	100%	1.95	29.19	4.0	100%
PMP	–	32.02	5.52	100%	–	46.83	5.07	73%	–	68.82	5.88	50%
DBMP	35.53	17.99	4.0	72%	55.18	39.76	4.0	53%	102.38	37.08	4.0	46%

layer composes of placements and the second layer composes of grasps. Two placements in the first layer are connected together if they share at least one common valid grasp. To solve a query, DBMP first searches for placement sequences that connect the start and goal placements. Then it examines each placement sequence and searches for feasible grasps associated with it.

However, since all the searches are deterministic and proceed in a depth-first fashion, the time-complexity of DBMP is significantly large. Instead of having three varying parameters for each placement class : two parameters for a location on the table and one parameter for the rotation about an axis normal to the table's surface, we constrained each placement class to have only a single varying parameter, the rotation. The same was also done implicitly in [4]. Furthermore, the deterministic search is also much sensitive to indexing of grasps and placements. To reduce this effect we shuffled grasp and placement indices before each run.

Note that our implementation of grasp set computation was different from the method used in [4], which was basically a discretization. Therefore, the pre-processing time put in Table I is intended only for reference. The numbers of total grasps computed in the cases of a box, an L-shaped object, and a chair are 124, 242, and 331, respectively.

C. Simulation Results and Comparisons

For each object, we ran each of the three planners 100 times. The data collected are pre-processing time, planning time, numbers of transitions, and success rate. (If no solution was found in 100 s., then the run was considered failed.) Data were averaged over successful runs and reported in Table I.

1) Comparison with Primitive Manipulation Planner: From the data reported in Table I, we can see that by exploiting more information about the connectivity of subsets of $\mathcal{G} \cap \mathcal{P}$, *i.e.*, by constructing high-level Grasp-Placement Tables, our planner was able to search for manipulation paths more systematically and efficiently, as reflected through the running time and the path quality achieved by our planner. As PMP had no information about the connectivity, much planning time was spent attempting infeasible connections between $\mathcal{G} \cap \mathcal{P}$ configurations. Furthermore, when the object was composed of more boxes, connection attempts were even more prone to failure since from any subset of $\mathcal{G} \cap \mathcal{P}$, the ratio of the number of reachable $\mathcal{G} \cap \mathcal{P}$ subsets to the number of all the subsets decreased.

2) Comparison with Discretization-Based Manipulation Planner: In fact, the two-layer regrasp graph [4] is similar to ours. One major difference is that they did not exploit any grasp parameterization in building the graph. Therefore, in their case, each grasp class contains exactly one grasp, hence numerous grasp classes. Also, the deterministic search employed in DBMP makes its capability relatively limited. Firstly, all placements have to be constrained to only one location on the table, or at most a few, due to time complexity of the search. Second, a significant amount of time is spent on exploring infeasible placement and grasp sequences due to the depth-first fashion of the search. This makes the success rate of DBMP in the given time much lower than the other planners.

D. Hardware Experiment

Besides simulations, we also conducted a hardware experiment. In this experiment, the robot must pick up the leg of a stool, which we approximated by three boxes, and hold it with a given grasp to facilitate a subsequent screwing operation (to be performed by another robot; not included in this experiment). Snapshots of the start and goal configurations are shown in Fig. 6. One can see that the initial pose of the object does not allow grasping with the desired grasp. Thus, the robot needs to regrasp the object several times. A video of the robot performing the task can be found at `https://youtu.be/tLouwjOwITQ`.

VI. Discussion and Conclusion

A. Extension to Broader Classes of Object Models

Our method for constructing the high-level Grasp-Placement Table is mainly based on parameterizations of grasp and placement classes. Parameterization enables categorizing infinitely many grasps and placements into a finite number of grasp and placement *classes*. This enables us to effectively capture the connectivity of $\mathcal{G} \cap \mathcal{P}$ subsets and encode it into a high-level Grasp-Placement Table without using any discretization.

Our method can generalize to models for which it is possible to find efficient placement and grasp parameterizations. In general, it is not difficult to find *placement classes* of a general object as one can compute the convex hull of the object and then test which surfaces of the hull result in stable placements. Therefore, the main requirement is that the object is *grasp-parameterizable*.

A wide variety of objects can indeed be grasp-parameterized, including many daily-life objects. For example, for a bottle, one can categorize grasps into three classes, see Fig. 7. For highly irregular objects with no efficient grasp parameterization, one may have to resort to grasp discretization. In such a case, our high-level Grasp-Placement Table will be similar to the regrasp graph of [4].

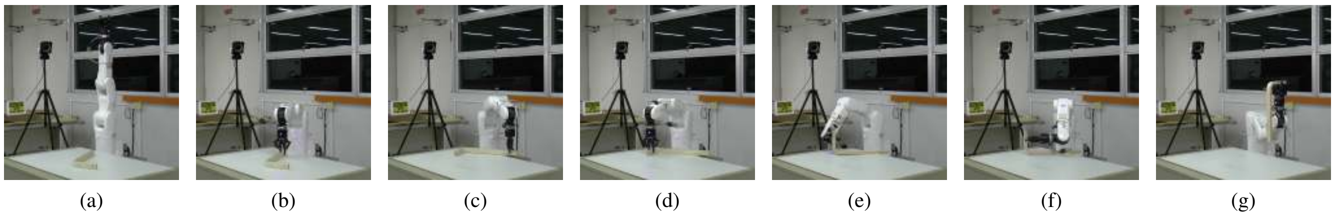


Fig. 6: Snapshots from the experiment on the real robot. (a) The start configuration. (b)–(f) The robot performed pick-and-place motions to move the object to a stable placement which allowed grasping with the desired final grasp. (g) The goal configuration. The video can be found at `https://youtu.be/tLouwj0wITQ`.

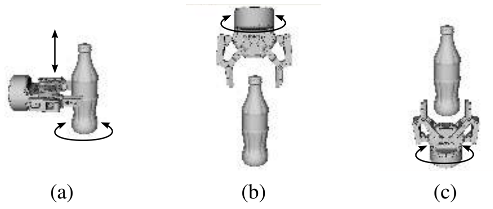


Fig. 7: Three possible grasp classes for a parallel gripper grasping a bottle. Arrows show gripper motions that result in grasps contained in the same class.

B. Conclusion

We have presented a manipulation planner to tackle pick-and-place planning problems. We first proposed a method to construct a high-level Grasp-Placement Table based on the models of the robot, the object, and the environment, without resorting to discretization of $\mathcal{G} \cap \mathcal{P}$, the fundamental set of configurations where transition between transit and transfer paths may occur. Our construction is therefore associated with a full parameterization of $\mathcal{G} \cap \mathcal{P}$, in contrast to previously proposed methods. The Table then serves as a guide for the planner to explore the composite configuration space.

Our method to construct the Grasp-Placement Table readily applies to movable objects that are boxes or composed of boxes. In such cases, assuming that the gripper can exert large enough forces with its fingers, all grasp classes considered here are also force-closure. We also discussed extension of the method to handle broader classes of objects.

The experimental results presented in Section V confirmed that the high-level Grasp-Placement Table helps improve both running time and manipulation path quality as compared to existing manipulation planners.

References

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

[2] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7–8, pp. 729–746, 2004.

[3] R. Alami, J.-P. Laumond, and T. Siméon, "Two manipulation planning algorithms," in *WAFR Proceedings of the workshop on Algorithmic foundations of robotics*, 1994, pp. 109–125.

[4] W. Wan, M. M. Mason, R. Fukui, and Y. Kuniyoshi, "Improving regrasp algorithms to analyze the utilityof work surfaces in a workcell," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015, pp. 4326–4333.

[5] F. R ohrdanz and F. M. Wahl, "Generating and evaluating regrasp operations," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 3, 1997, pp. 2013–2018.

[6] S. A. Stoeter, S. Voss, N. P. Papanikolopoulos, and H. Mosemann, "Planning of regrasp operations," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1, 1999.

[7] Y. Koga and J.-C. Latombe, "On multi-arm manipulation planning," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, vol. 2, 1994, pp. 945–952.

[8] C. L. Nielsen and L. E. Kavraki, "A two level fuzzy PRM for manipulation planning," in *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 3, 2000, pp. 1716–1721.

[9] K. Harada, T. Tsuji, and J.-P. Laumond, "A manipulation motion planner for dual-arm industrial manipulators," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014, pp. 928–934.

[10] 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.

[11] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Manipulation with multiple action types," in *Experimental Robotics*, ser. Springer Tracts in Advanced Robotics. Springer International Publishing, 2013, vol. 88, pp. 531–545.

[12] B. J. Cohen, S. Chitta, and M. Likhachev, "Search-based planning for manipulation with motion primitives," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 2902–2908.

[13] B. Cohen, S. Chitta, and M. Likhachev, "Single- and dual-arm motion planning with heuristic search," *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 305–320, 2013.

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

[15] A. Krontiris and K. E. Bekris, "Dealing with difficult instances of object rearrangement," in *Proceedings of Robotics: Science and Systems*, 2015.

[16] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[17] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2014, pp. 3684–3691.

[18] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russel, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014, pp. 639–646.

[19] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "FFRob: an efficient heuristic for task and motion planning," in *Algorithmic Foundations of Robotics XI*. Springer International Publishing, 2015, vol. 107, pp. 179–195.

[20] N. Yamanobe and K. Nagata, "Grasp planning for everyday objects based on primitive shape representation for parallel jaw grippers," in *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, 2010, pp. 1565–1570.

[21] S. M. Lavalle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch, and D. Rus, Eds. A. K. Peters, 2001, pp. 293–308.

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

[23] F. Lamiraux and J. Mirabel, "HPP: a new software framework for manipulation planning," 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01138118

[24] F. C. Park and B. Ravani, "Smooth invariant interpolation of rotations," *ACM Transactions on Graphics*, vol. 16, no. 3, pp. 277–295, 1997.