Hierarchical planning for multi-contact non-prehensile manipulation

Gilwoo Lee, Tomás Lozano-Pérez, and Leslie Pack Kaelbling

***Abstract*—Manipulation planning involves planning the combined motion of objects in the environment as well as the robot motions to achieve them. In this paper, we explore a hierarchical approach to planning sequences of non-prehensile and prehensile actions. We subdivide the planning problem into three stages (object contacts, object poses and robot contacts) and thereby reduce the size of search space that is explored. We show that this approach is more efficient than earlier strategies that search in the combined robot-object configuration space directly.**

I. Introduction

Humans exploit both prehensile and non-prehensile manipulation when working with objects. For a small, light object, such as a cup, a simple grasp operation often suffices. For a large, heavy object, such as a piece of furniture, non-prehensile manipulation, such as pushing, may achieve the desired result. In other cases, it may be necessary to combine prehensile and multiple non-prehensile operations, including pushing, tilting, tumbling, etc., to achieve a desired result.

Manipulation planning involves planning the combined motion of objects in the environment as well as the robot motions to achieve them. We would like to be able to plan prehensile and non-prehensile manipulation in an integrated fashion. Ideally, a planner could pick from among a large repertoire of diverse action types to achieve a desired state in the most reliable and efficient manner. However, although great progress has been made in planning motions in the robot's configuration space in the presence of obstacles, progress in general manipulation planning has been more limited.

While the state of the world and robot can be represented in a single combined configuration space, the manipulation planning problem has a number of complicating factors. The dimensionality of the combined space may be very large, and the domain is severely under-actuated: the robot can only affect the state of objects when it is contact with them and the interactions are mediated by possibly complicated dynamics constraints. As a result, there are few planners that address the problem of combining prehensile and non-prehensile manipulation. Some exceptions are the work of Hauser and Latombe [1], Dogar and Srinivasa [2], Barry et al. [3] and King et al. [4], but they have mostly focused on pushing.

This work was supported in part by the NSF (grant 1420927). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We also gratefully acknowledge support from the ONR (grant N00014-14-1-0486), from the AFOSR (grant FA23861014135), and from the ARO (grant W911NF1410433).

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA allielee@mit.edu, tlp@mit.edu, lpk@mit.edu

Manipulation planning is a form of multi-modal planning [1, 3]; that is, it is planning in a hybrid space, in which the modes represent sub-manifolds where different motion constraints, such as contact with a particular object face, hold. The presence of these constraints mean that the classic sample-based algorithms for robot motion planning do not apply directly, since samples need to be generated at the lower-dimensional mode transitions. Extensions to this style of algorithm to multi-modal planning exist; however, when the modes form continuous families (which is the case in manipulation), planning in such multi-modal spaces generally requires some form of hierarchical planning for search guidance. For example, planning for the manipulated object ignoring the robot (and possibly obstacles) and then using the abstract plan to guide search for a full plan for the robot [3], or decomposing *push* operation into pre- and post-contact, using an A* search for pre-contact trajectory and then using pre-computed post-contact policy [5].

In this paper, we explore a hierarchical approach to planning sequences of non-prehensile and prehensile actions. This hierarchical approach is more efficient than earlier strategies that search in the combined robot-object configuration space directly. We illustrate the approach through an initial implementation that manipulates a single convex planar object among planar obstacles using point contacts with friction; we leave robot shape, robot kinematics and grasp stability considerations for future work.

II. Approach

Our planner operates hierarchically, first finding a sequence of qualitative "object contact states" that characterize which parts of the moving object are in contact with which parts of other objects, then finding a feasible sequence of poses for the object, and finally finding a sequence of contact points for the manipulators on the object. This hierarchical structure provides significant search guidance, and divides the problem into three search problems that are much smaller than a search in the full combined configuration space of the object and manipulators.

The formulation of the first two search problems builds on existing work on contact-state graphs [6, 7]. We start by constructing a graph that represents the contacts between the moving object and other fixed objects in the world, not yet worrying about the exact pose of the object or the robot. A contact-state path provides a useful constraint for the actual path of the object. Figure 1 shows a contact-state graph of an object moving on a table, containing sample object poses in each contact state. To move the object from $\mathbf{pose_i}$ to $\mathbf{pose_j}$ without lifting it, we can first find a path from $\mathbf{contact\ state_k}$ to $\mathbf{contact\ state_m}$ and then find a pose sequence

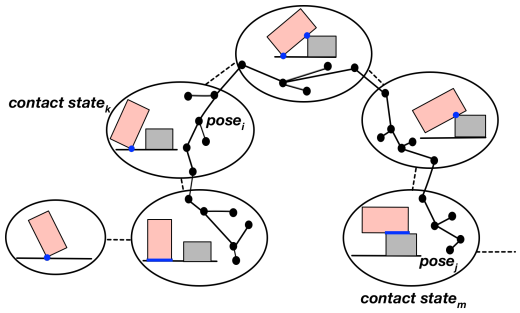

Fig. 1: A contact state graph with poses connected through linear interpolation. Poses connecting two contact states are very close to each other.

within the contact-state path. Finally, we search for a robot-contact plan, which is a sequence of robot contacts and forces on the object, which will cause the object's motion to comply to the pose path found in the previous phase.

To find a robot-contact plan, we begin by discretizing the object's surface into a set of possible contact points and define a state to contain an object pose and a set of contacts of the robot's manipulators on the object. We then identify states that are *feasible:* both *geometrically accessible*, meaning that the robot can reach all of the specified contacts, and *stabilizable*, meaning that there exists a set of contact forces between the object and the robot's manipulators, as well as the fixed objects, that can stabilize the object against gravity. Throughout this paper we will focus on the case of two point-manipulators with hard contacts. For example, if the object is at pose $p = (0, 0, \pi/6)$ and one manipulator is on c_1, as shown in figure 2, the manipulator can passively support the object. The range of feasible locations of c_1 depends on the angle and the coefficient of friction. For a particular object pose p, we can evaluate all combinations of c_1, and c_2 to find feasible robot-contact states. Any geometrically valid pose p can be associated with a space of discretely sampled robot contact pairs. The left part of figure 2 shows such a space.

The decomposition of robot motions into *transit* and *transfer* motions [8] can be viewed as transitions within and across robot-contact spaces. A *transit*, changing the manipulator contacts with the object in a fixed pose, is a transition between two feasible states within the same robot-contact space. For example, in figure 2, (p, c_1, c_2), $(p, c_1, none)$, and (p, c_1, c_3) are all feasible, and the manipulator can *transit* from (p, c_1, c_2) through $(p, c_1, none)$ to (p, c_1, c_3) by moving $\mathbf{hand_2}$ from c_2 to c_3. In the robot contact space, this is a vertical transition among three states that share c_1. A *transfer*, changing the object pose while maintaining the manipulator contacts, is a transition between (p_1, c_1, c_2) and (p_2, c_1, c_2), where p_1 and p_2 are neighboring poses in an object-pose path. This is a transition from a state in p_1's robot contact space to the corresponding state in p_2's robot contact space, as shown in figure 3. For a *transfer* to be valid, it must also be possible for the manipulator to induce the wrench necessary to move the object in the desired direction.

Figure 4 illustrates the connected search spaces: within the discrete contact states in the contact-state graph, there are individual object poses, and a path through object-contact space can be realized by a path through object pose space.

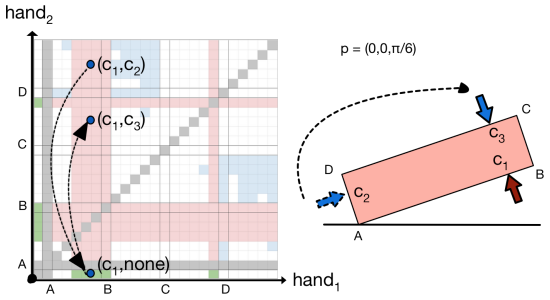

Fig. 2: Robot contact space for $p = (0, 0, \pi/6)$. Each axis represents possible contact points along the object's surface accessible by $hand_1$ and $hand_2$. The leftmost column and the bottom row represent no-contact for $hand_1$ and $hand_2$, respectively. Green cells represent *feasible* states in which the object can be balanced by one hand. If one hand is sufficient to stabilize the object, the other hand can place itself on any *geometrically accessible* surface; these states are colored in red. For example, if a row's leftmost cell is green, all geometrically accessible cells in the same row are red. States that require both hands are colored in blue. Grey cells represent states with geometrically inaccessible contacts; white cells represent those that are not stabilizable. A *transit* is a transition from a red state to another red state in the same row or column. The figure shows transits from (c_1, c_2) to $(c_1, none)$ and then to (c_1, c_3).

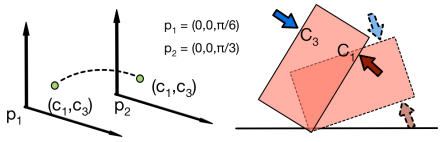

Fig. 3: *Transfer* from (p_1, c_1, c_3) to (p_2, c_1, c_3).

Then, for each object pose, there is a set of robot contacts, and a path through object pose space can be realized by a path of transit and transfer motions through the combined space of robot contacts and object poses.

Figure 5 illustrates the hierarchical search process. At the bottom is a path in contact space; above that is a realization of it in object pose space, and at the top is a realization of that path in the combined robot-contact and object-pose space. Because we are reducing the search space at each step, we also reduce the set of feasible paths at each step. Hence, if we fail to find a path in the final search, we backtrack to find different solutions in the previous searches. Despite the possibility of having to

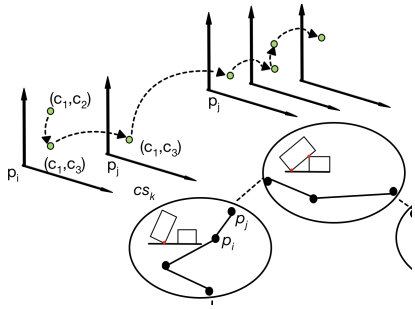

Fig. 4: The relationship between the spaces of object contacts, object poses, and robot contacts.

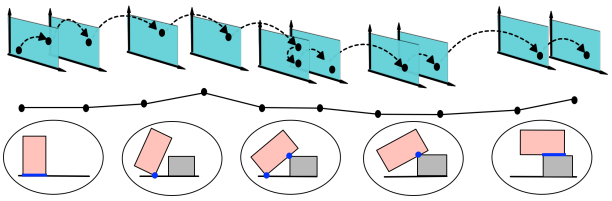

Fig. 5: Object-contact state path, object-pose path, and robot-contact path aligned together.

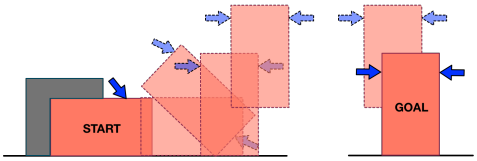

Fig. 6: A combination of prehensile and non-prehensile manipulation. In this example, the coefficient of friction between the box and the environment is small relative to the coefficient of friction between the box and the manipulator, which allows the manipulator to apply enough force in the tangential direction to move the box horizontally. The manipulator drags the box out of a shelf, reorients it to a graspable pose, and then moves the box to another table by lifting it.

repeat these steps multiple times, we find empirically that this hierarchical approach is much faster than performing a single search in the combined space directly.

Our planner can be extended to enable combinations of prehensile and non-prehensile manipulation by adding extra connections within the "free" contact state, in which the object is not in contact with any of the fixed objects. For any stable set of robot contacts, all collision-free transfers (changes to the object pose), possibly subject to additional constraints such as force closure, are allowed. This combination of manipulation strategies is very rich: for example, we can move a box from a tight shelf to a detached table by first pulling it out, reorienting it, and lifting and placing it on the table, as shown in figure 6.

The main advantage of our approach is that the individual search spaces are relatively small. In section V, we compare our approach with a single search in the combined space and show a major computational advantage.

III. Related Work

There has been substantial research on planning individual non-prehensile manipulation operations, such as pushing [9, 10] and toppling [11]. There has been less work on planning sequences of non-prehensile operations. Early work by Erdmann and Mason [12] considered planning a sequence of "sensorless" tilting operations to place an object at a desired final pose. Abell and Erdmann [13] developed a planner for rotating a 2D polygon by maintaining two contacts that can induce quasi-static rotations of the object. Erdmann [14] later presented a non-prehensile two-palm manipulation planner that reorients an object by calculating angles in which the object is held in equilibrium and angles in which the object would slide or rotate around a vertex. A more recent work by Yamawaki and Yashima [15] presented a two-phase, whole-arm manipulation planner for reorienting planar polygonal objects.

The line of work most closely related to our own is by Maeda and Arai; they have successfully demonstrated non-prehensile manipulation planning with multiple robot contacts across multiple contact states [16, 17]. Initially, they assumed a planar environment and that the contact state graph was given; they discretized the combined configuration space of the object and the robot hands in each contact state, connecting poses for transits and transfers. They extended the approach to three dimensions and incrementally generated poses for a single contact states. They performed A* search in the combined object, robot configuration space, in this case, limited to a single contact state. In both cases, this approach resulted in a very large state space with many possible transition edges. Since checking for geometric constraints and manipulation feasibility is expensive, it resulted in large running times. More recently [18] they applied an RRT-variant to perform the search which resulted in some improvements in average running time but with high variance. Their work has also pursued a careful assessment of the stability of contacts, finding plans that are most stable under this measure, an important issue that we have not considered here.

In this paper, we integrate contact-state compliant task-planning and force stability analysis to achieve non-prehensile manipulation planning. Our planner considers motions compliant to a sequence of contact states, as in Xiao and Ji [6][7], to connect the initial and goal poses of the object. Then, we identify manipulator contacts that can move the object along a chosen path. To do this we apply the force feasibility analysis introduced by Han et al. [19].

IV. Algorithms

The top-level structure of our planner is as follows:

- Construct a sequence π_{oc} of connected abstract states representing contacts of the robot with the other objects in the world.
- Construct a sequence π_{op} of connected object poses that moves through, while conforming to, the chosen elements of π_{oc}.
- Construct a sequence π_{rcop} of the robot's contacts on the object, that will allow the robot to apply wrenches that will move the object along the trajectory specified by π_{op}.

We then use a feedback controller to apply the robot contact forces needed to execute π_{rcop} to move the target object to its goal pose.

Formally, we are given the following as input: a description of the convex polyhedral shape for a target object to be moved; initial and goal object poses at which the object is statically stable without any robot contacts; a description of all other objects $O_1, \ldots, O_m$ in the world, in the form of a union of convex polygons; a number n of point contacts the robot can exert; coefficients of friction for all contact surfaces; and a maximum amount of force that can be applied by the robot. The following sections describe

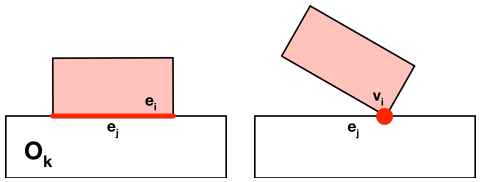

Fig. 7: The left contact state is (O_k, e_i, e_j), and the right contact state is (O_B, v_i, e_j). Since e_i contains v_i, the principal contact (O_k, v_i, e_j) is contained by (O_k, e_i, e_j), and therefore the right contact state is a less-constrained neighbor of the left contact state.

algorithms for solving each of the planning problems, given the input.

A. Object-contact planner

An *object-contact graph* (OCG) is a connected set of abstract states that are characterized by the contacts between one target object and other objects in its environment. Contact graphs have been suggested for planning compliant motions [20, 21] and used by Maeda and Arai [16] for planning non-prehensile manipulation. Xiao and Ji [6] provide algorithms for constructing an OCG given descriptions of the objects' shapes. We construct an OCG and use it to find an abstract plan for moving the target object.

Each node in an OCG is a *geometrically valid object-contact state*. We define an *object-contact state* for target object T to be a set of *principal contacts* between T and fixed objects $O_1, \ldots O_m$. Each principal contact (PC) has the form (O_k, ϕ_i, ϕ_j), where ϕ_i is a feature (edge, or vertex) of T and ϕ_j is a feature of O_k. So, for example, if edge e_j of T makes contact with vertex v_j of O_k, the principal contact would be (O_k, e_i, v_j). An object-contact state is *geometrically valid* if and only if there is at least one collision-free pose that satisfies the geometric constraints imposed by the state.

To define edges between object-contact states, we make use of the following properties, defined by Xiao and Ji [6] and illustrated in figure 7.

- A principal contact (O_k, ϕ_i, ϕ_j) *contains* principal contact (O_k, ϕ_a, ϕ_b) if and only if (a) $\phi_a \in \phi_i$ and $\phi_b \in \phi_j$ *or* (b) $\phi_a = \phi_i$ and $\phi_b \in \phi_j$ *or* (c) $\phi_a \in \phi_i$ and $\phi_b = \phi_j$.
- A contact state S_1 *contains* contact state S_2 if and only if (a) $|S_1| \geq |S_2|$ and (b) for every PC $z_2 \in S_2$, either it is in S_1 or there is a unique PC $z_1 \in S_1$ such that z_1 contains z_2 and (c) no two PCs in S_2 are contained by the same PC in S_1.
- If S_1 contains S_2, we call S_2 a *less-constrained neighbor* of S_2.

There is an undirected edge between two object-contact states S_1, S_2 in the OCG if S_1 is a less-constrained neighbor of S_2. Edges correspond to adding and removing principal contacts, and to changing the nature of a principal contact (for example, changing an edge contact to a vertex contact on that edge).

The object-contact planner constructs an OCG and uses it to find a connected path π_{oc}. It takes as input contact states S_0 and S_g corresponding to the initial and goal poses of the object. It also takes *seeds*, a set of user-provided contact states representing the most constrained, geometrically valid contact states. The result depends on the shapes and poses of the permanent objects and the shape of the moving object.

The algorithm begins by constructing a sub-graph for each seed, by recursively generating the set of object-contact states that are less constrained than the seed, using the less-constrained neighbor relationships. In the absence of user-provided seeds, an additional pre-processing step to identify them from the models is required.

Some object-contact states generated in this process may be geometrically invalid, in which case we discard them. Testing for geometric validity can be difficult: it is necessary to construct a collision-free pose for the object that satisfies the contact constraints. Therefore, this is an instance of a geometrically constrained motion-planning problem that can be addressed (approximately) with a sampling-based motion planner [7]. In our implementation we use a hybrid approach. For states with single principal contact, we randomly sample poses that satisfy the principal contact, and check whether they are collision-free. For states with multiple principal contacts, we formulate and solve an optimization problem. We begin by constructing a plausible solution pose, p_0. For each principal contact z_i in object-contact state S, we find an arbitrary satisfying pose p_i. Then we set p_0 to be the average of the p_i. This pose is not likely to satisfy either the contact or collision constraints. So, we formulate the problem of finding p to minimize $(p_0 - p)^T(p_0 - p)$ subject to the contact constraints in S and to the constraint of not colliding with any of the fixed objects. The optimization problem is non-convex, but in practice we have found that a satisfying solution can be found, possibly with a few restarts using different values of p_0. If we find such a pose, then we consider S to be geometrically valid, otherwise, we discard S.

Now, we have a set of graphs grown from the seeds. Generally, the graphs will have some contact states in common, which will allow us to merge them into a single graph. If not, we identify more seed states and repeat the process until S_0 is connected to S_g. Once S_0 and S_g become connected in the graph, we search using breadth-first search for the path with the fewest transitions and return the resulting path, π_{oc}, through the OCG.

B. Object-pose planner

Given a qualitative plan for the object in terms of its contacts with other objects, π_{oc}, we search for a detailed plan for the object motion, in the form of a sequence of poses, π_{op}, such that the continuous object trajectory resulting from linear interpolation of these poses moves the object through the contact states π_{oc}.

Again, we treat the problem as one of constructing a graph and then searching it for a path. The nodes in the *object pose graph* (OPG) are *object pose states*, which have the form (p, S), where p is an object configuration ($p = (x, y, \theta)$ in the planar case), and S is an object-contact

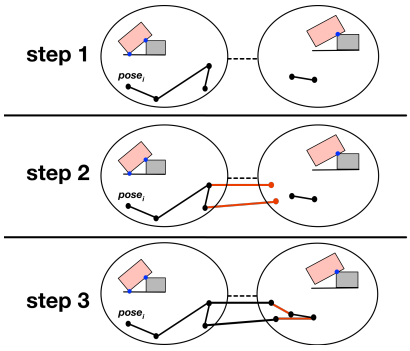

Fig. 8: Once we connect poses within each contact state (step 1), we connect poses across neighbor contact states. We apply a small perturbation to each pose in the more-constrained contact state. If this perturbation results in a valid pose in the less-constrained contact state, we add the pose to the graph and connect it with its original pose (step 2) as well as with other poses in the less-constrained contact state (step 3).

state. It is relatively simple to compute S from p but it is more efficient to represent it explicitly; we will only consider nodes for which the object placed at p satisfies the constraints in S and does not collide with the permanent objects.

The object-pose planner constructs an OPG and uses it to find a connected path π_{op}. It takes as input the object-contact state path π_{oc}, the initial object pose p_0 and the goal object pose p_g. It begins by generating sample states, (p_i^j, S_i), for each of the S_i in π_{oc}, that satisfy contact constraints of S_i and are collision-free. (p_i^j, S_i), (p_i^k, S_i), are connected if a linear interpolation between p_i^j and p_i^k that continuously satisfies the non-collision constraint exists. The interpolation algorithm is described by Ji and Xiao [7]. Next, for each transition between object contact states, S_i, S_{i+1} in π_{oc}, and for each sampled pose p_i^j for S_i we generate new poses by applying small translations and rotations, as described by Xiao and Ji [6]: If the resulting object pose, p' satisfies the conditions of object contact state S_{i+1}, then we add node (p', S_{i+1}) to the graph, connect it to the (p_i^j, S_i) from which it was constructed, and attempt to linearly connect (p', S_{i+1}) to other object-pose states in object-contact state S_{i+1}. Finally, we add nodes for the initial and final poses, (p_0, S_0) and (p_g, S_g), and connect them to directly reachable nodes in the graph with matching S values. Now, we search the graph to find a path π_{op} through the object-pose graph from the initial to the goal pose.

Note that in the construction of this pose graph, we evaluate and connect poses only within a contact state and across neighboring contact states. The number of evaluations that must be performed for connecting poses is $O(n^2)$, where n is the number of sampled poses per contact state. In practice, the size of n can be very small relative to the total number of sampled poses as the number of contact states gets large.

C. Robot-contact planner

The last level of planning is to construct a sequence π_{rcop} of the robot's contacts on the object, that will allow the robot to apply wrenches that will move the object along the pose trajectory specified by π_{op}. Once again, we construct and then search a graph.

Nodes The *robot contact graph* (RCG) has *robot contact states* as nodes. A *robot contact state* (RCS) has the form (p, S, C) where p is an object pose, S is an object contact state and C is a vector of contacts, corresponding to contacts the robot makes on the object. Each robot contact c is either *none* or a point of contact between the robot and the object. For a two-fingered robot and planar objects, when both fingers are in contact and there is one object contact, e.g. with the table, the entire state is described by: $((x, y, \theta), \{(O^1, \phi_i^1, \phi_j^1), \ldots, (O^m, \phi_i^m, \phi_j^m)\}, (c^1, c^2))$. The contact points c^i, can be represented as displacements along the movable object's surface, in our planar examples, it is sufficient to represent a one-dimensional displacement from a fixed vertex on the polygon. This can be mapped to x, y coordinates as necessary.

An RCS is a legal node in an RCG if it is both geometrically accessible and stabilizable. An RCS is *geometrically accessible* if the robot contact points c^1 and c^2 are on a surface of the movable object that is not in contact with any other object. This is a very simplified condition that should be extended to require the existence of a collision-free path for a particular physical robot, as well. An RCS is *stabilizable* if there exists a solution to its corresponding *force feasibility problem* [19].

Consider an object (assumed not to be in motion) that has contacts $c^1, \ldots, c^n$, including robot contacts as well as contacts with other objects (implied by the contact state S); contact between two edges is approximated as two point contacts at the two endpoints of the edge contact. We wish to determine whether there is a combination of forces that can be exerted by these contacts that will balance the force of gravity.

Let $f^i = (f_x^i, f_y^i)$ be the force exerted at c^i in the coordinate frame defined by c^i. For each contact c^i, let G^i be a transformation matrix that maps points in the contact's frame of reference to points in a frame of reference fixed at the object's center of mass. Let $\boldsymbol{f}$ be the vector $(f^1, \ldots, f^n)$ of forces applied at each of the contacts. Then the total wrench exerted on the object is

$$\boldsymbol{G f} = (G^1, \ldots, G^n)\boldsymbol{f}$$

in the frame of the object. To maintain equilibrium, this wrench must balance the wrench caused by gravity:

$$\boldsymbol{G f} + \boldsymbol{G'} \begin{bmatrix} 0 \\ -mg \\ 0 \end{bmatrix} = 0 \tag{1}$$

where $\boldsymbol{G'}$ is a transformation matrix that maps vectors in a world coordinate frame to the object's coordinate frame. In addition, we also have the following frictional constraints for each contact:

$$f_y^i \geq 0$$

$$|f_x^i| \leq \mu_i f_y^i \tag{2}$$

where μ_i is the coefficient of friction for c_i. If there is an upper bound on the amount of force applied by the robot, we

add this constraint as well. Given the set of robot contacts and contacts with other objects, it is possible to determine whether there exists a set of forces at these contacts that can balance the force of gravity. The exact distribution of forces is statically indeterminate, but the minimum norm solution (that minimizes $\boldsymbol{f}^T\boldsymbol{f}$) provides a reasonable prediction of the actual force distribution [22]. Given the same set of contacts, there may be other solutions that balances gravity, but we always take the minimal net force solution. Hence, each state is unique in the RCG.

Edges There are two qualitatively different types of transitions between RCSs: transit and transfer. In a *transit*, the object's pose stays constant but the robot contacts change; in a *transfer*, the robot's contacts on the object stay constant with respect to the object, and the object's pose changes.

A transit between two legal RCSs $(p, S, (c^1, \ldots, c^n))$ and $(p, S, (d^1, \ldots, d^n))$ is an edge in the RCG if for every contact i, at least one of the following conditions holds: $c^i = d^i$, $c^i = none$ or $d^i = none$. This means that the object will be stable during the execution of an interpolated path between those states.

For a *transfer* to be a legal edge, it is necessary but not sufficient for the two end states to be stabilizable. For example, a box on a table can be in equilibrium at two distinct poses with no robot contact, but without any robot contact, there is no single *transfer* between the two states. In order for a pose change to be legal, there must be a combination of contact forces that induces the necessary motion. If the transfer involves a combined translation and rotation, we break it into two steps, one pure translation and one pure rotation. We also assume that a rotation always pivots around a contact with the environment.

Algorithm The robot-contact planner constructs an RCG and uses it to find a connected path π_{oprc}. It takes as input the object-pose state path π_{op}, the number n_c by which we discretize the object surface, the initial RCS (p_0, S_0, C_0) and the goal RCS (p_g, S_g, C_g). We assume that the object should be statically stable at initial and goal poses, so both C_0 and C_g are $(none, none)$.

We begin by constructing nodes in the graph. We discretize the object's surface by n_c to generate a set of possible robot contact points $\mathcal{C}$; also included in $\mathcal{C}$ is the element *none* representing no contact. For each (p_j, S_j) in π_{op}, we consider RCS $(p_j, S_j, (c^1, \ldots, c^n)$ for all $c^1, \ldots, c^n \in \mathcal{C}^n$. Any of these RCSs that are geometrically inaccessible are immediately discarded.

For each of the remaining candidate RCSs, we attempt to find a set of forces $\boldsymbol{f}$ that make the state stabilizable. We use the gravitational and friction constraints to form a quadratic program that seeks to minimize total force ($\boldsymbol{f}^T\boldsymbol{f}$), as in [22]. If a solution exists, with forces that can be exerted by the robot, it is added to the graph.

Transit edges are added between any pair of RCSs that differ only by adding or deleting contacts. For any transition between OPSs $(p_i, S_i), (p_{i+1}, S_{i+1})$ in π_{op}, we consider transfer edges between (p_i, S_i, C) and (p_{i+1}, S_{i+1}, C), for all $C \in \mathcal{C}^n$. We must check to see whether the contacts C can generate a wrench that would cause the necessary rotation or translation.

To check whether the robot contacts can generate the necessary wrench, we once again formulate a force feasibility problem. First, we identify the contact mode change for contacts with other objects during the motion. During a transfer between (p_i, S_i, C) and (p_{i+1}, S_{i+1}, C), all robot contacts remain in contact, but some of the contacts with other objects may change their modes. Some of them may be removed, and some of the contacts may become sliding contacts, depending on the nature of the motion. If a contact c^k is removed during the transfer, we remove the relevant terms from $\boldsymbol{G}$ and $\boldsymbol{f}$. If c^k becomes a sliding contact, we modify its corresponding inequality constraints in equation 2 to an equality constraint that opposes the direction of the motion.

Then, given the contacts and their contact modes, we must check whether these contacts can generate a wrench that cause the necessary rotation or translation. Under the quasi-static assumption for planar objects, the desired wrench direction can be approximated with the desired motion direction of the object's center of mass. Let $[\delta x \quad \delta y \quad \delta\omega]^T$ be the desired motion direction. In case of translation, this can be found be taking the offset between the two poses, with $\delta\omega = 0$. In case of rotation, this can be found by identifying the center of rotation and finding the motion direction of the center of mass, with $\delta\omega$ indicating the sign of rotation. Once we identify $[\delta x \quad \delta y \quad \delta\omega]^T$, we can modify equation 1 of the force feasibility problem to be the following:

$$\boldsymbol{G}\boldsymbol{f} = \boldsymbol{G}'\left(\begin{bmatrix} 0 \\ mg \\ 0 \end{bmatrix} + k \begin{bmatrix} \delta x \\ \delta y \\ \delta\omega \end{bmatrix} \right) \tag{3}$$

with k being constrained to be positive. If there is a solution to this problem, it implies that the contacts can generate wrench in the direction of the object's desired motion.

Once the graph has been constructed, we search it for a path from the initial RCS to the goal. If there is no such path, the overall search backtracks and tries to find paths at the previous levels.

D. Plan execution system

Given the final plan π_{rcop}, we use a simple closed-loop controller to execute it on the robot.

During execution, we position-control the robot's contacts on the object. Some contacts require active application of force while others passively support the object to resist gravity. In order to move the object in the desired direction, we need active "pushing" only when the robot contact's motion direction, determined by its force vector, is in the same direction (within some predetermined range) as the object's motion direction.

At time t, the desired pose, contact point, and contact forces, $p^t, c_1^t, f_1^t, c_2^t, f_2^t$, are provided. If the desired motion direction at a contact point is within $\pi/2$ of the force direction, we move the robot contact, r_i, in the direction of the contact force, f_i. If it is in the opposite direction, the hand is slightly detached from the object to prevent applying extra force. The motion direction for the robot manipulator is defined by

$$dr_i^t = \text{sign}(dc_i^t \cdot f_i^t)\delta f_i^t$$

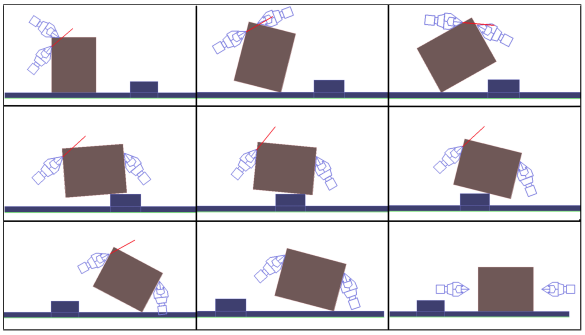

Fig. 9: Key frames from a sample solution trajectory for tumbling one box over another. Red lines indicate the force direction of the robot contact pushing the object. The "hands" simply highlight the location of the chosen robot contacts.

where dc_i^t is the motion direction for the contact point and δ is a small constant.

Transit motions with the robot not in contact with the object are executed with pure position control.

If our goal were to track the designed trajectory very precisely, we would need a more sophisticated controller. However, since we only desire to move the object through the sequence of object contact states to a final pose, this simple approach suffices.

V. Implementation and Results

We have implemented a version of this planner for planar objects and two robot contacts, without any further kinematic or collision constraints introduced to model the robot performing the manipulation. The implementation is in Python, with collision-checking done by Box2D [23]. The geometric validity of poses is checked using MATLAB libraries; this call to MATLAB is currently the bottleneck in our implementation. The results were tested in simulations that were constructed in Box2D. We compared the performance of our approach to that of a "combined" planner (similar in structure to those of Maeda et al. [16, 17]) that follows an object contact-state path but plans for both and robot contacts at the same time.

We tested these approaches on two problems. The first, shown in figure 9, focuses on sequencing non-prehensile manipulation steps. There is an obstacle in the middle of the table, and the goal in this problem is to move the box to the other side of the table. Allowing only nonprehensile manipulation, the planner is able to find a solution.

Tables Ia and Ib summarize the running times for the hierarchical method on this problem. In this example, the OCG path always contains the same sequence of 9 OCSs. Given this OCG path, we varied the number of sampled poses per OCS in table Ia. Although the number of nodes and the search time increases, the size of the resulting OPG remains around 15 nodes. Hence, the impact on the next stage, robot-contact planning, is minimal. Using the 15 OPS nodes, we varied the discretization of the object surface (the potential robot contacts); the results are in table Ib. The increased discretization leads to a rapid rise in the size of the graph and the search time. However, relatively small values are often sufficient.

TABLE I: Results for the example in figure 9

(a) OPG construction and search with different number of pose samples; values are averaged over 3 trials. Used 9 OCS.

Poses / OCS	Nodes in OPG	Nodes in path	Time(s)
5	60	14	109.4
10	84	15	145.4
15	110	15	203.3

(b) RCG construction and search with different discretization of object surface. Used 15 OPS.

Discretization	Nodes in RCG	Nodes in path	Time(s)
12	452	24	19.7
20	1,157	22	58.5
40	4,095	21	278.5

The hierarchical approach is much more efficient than the combined approach that considers both pose and robot contacts at the same time. Table II shows a comparison between the two approaches, given the same OCS path. The leftmost column indicates the discretization of the object surface. In the combined approach, increases in either the number of pose samples or the discretization compound, with significantly effect on both time and space. Even with 5 poses per object-contact state, the running time increased so dramatically that we could not complete the graph construction when the object surface discretization was 40. Likewise, we could not complete the graph construction for 10 poses per OCS case with surface discretization of 20 or 40. In the hierarchical approach, increasing the number of sample poses generally affect only the object-pose planner, and its impact on the robot-contact planner is small.

The second example, shown in figure 10, sequences non-prehensile manipulation with prehensile manipulation. Here, a portion of the object is inaccessible because of a shelf and so the robot cannot grasp it. If the shelf is slippery, that is, the friction coefficient between with the object is lower than that with the robot, the robot can apply tangential force to slide the box from the shelf and then grasp it to move it to another table. This is a simpler problem and the running times are substantially smaller, as shown in tables IIIb and IIIa.

TABLE II: Combined approach compared with the hierarchical approach. Leftmost column indicates discretization of object surface. For hierarchical approach, the sums of OPG and RCG for corresponding columns are taken from table Ia and Ib. * indicates that the computation was incomplete after 3 hours.

	5 pose per OCS				10 pose per OCS			
	Combined		Hierarchical		Combined		Hierarchical	
	Nodes	Time(s)	Nodes	Time	Nodes	Time	Nodes	Time
12	1,492	362	512	129	2,250	832	536	165
20	3,315	1,666	1,216	168	7,208	*	1,241	204
40	*	*	4,154	388	*	*	4,179	424

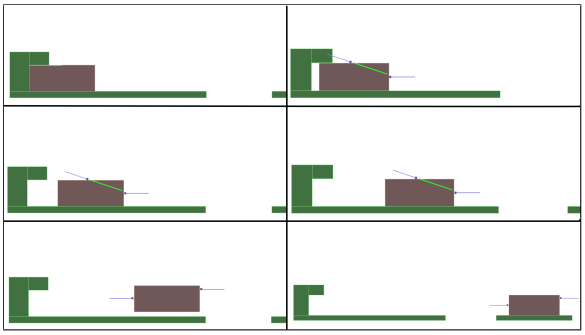

Fig. 10: Key frames from a sample solution trajectory for sliding a box out from underneath a shelf. Green lines indicate the force direction of the robot contact pushing the object.

TABLE III: Results for the example in figure 10

(a) OPG construction and search with different number of pose samples.

Poses / OCS	Nodes in OPG	Nodes in path	Time(s)
3	7	3	0.7
5	9	3	1.0
10	17	3	3.7

(b) RCG construction and search with different surface discretization.

Discretization	Nodes in RCG	Nodes in path	Time(s)
12	86	7	1.3
20	206	7	4.6
40	612	7	28.9

VI. Conclusion

We have presented a hierarchical approach to planning sequences of non-prehensile and prehensile actions. We have compared it with a less hierarchical approach and showed that our approach is significantly more efficient for both time and space. This significant reduction in search space may sometimes require us to backtrack to previous stages when the search fails, but we found emperically that this approach is still tractable even with some backtracks and that such backtracks happen infrequently.

One challenging aspect of our work is extension to three dimensions and to three or more robot contacts. In three dimensions and with more robot contacts, the number of possible states in OPG and RCG would increase dramatically, and thus search in both stages would be very expensive. But we believe that we can limit the dramatic expansion of the search space by focusing on a subset of the contacts that are more robust and steering the planning in the small subset of the search space.

References

[1] K. Hauser and J.-C. Latombe, "Multi-modal motion planning in non-expansive spaces," *IJRR*, vol. 29, 2010.

[2] M. R. Dogar and S. S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Autonomous Robots*, vol. 33, no. 3, 2012.

[3] J. Barry, L. P. Kaelbling, and T. Lozano-Perez, "A hierarchical approach to manipulation with diverse actions," in *ICRA*, 2013.

[4] J. King, M. Klingensmith, C. M. Dellin, M. R. Dogar, P. Velagapudi, N. S. Pollard, and S. S. Srinivasa, "Pregrasp manipulation as trajectory optimization." in *Robotics: Science and Systems*, 2013.

[5] M. Koval, N. Pollard, and S. Srinivasa, "Pre-and post-contact policy decomposition for planar contact manipulation under uncertainty," in *Proc. Robotics: Science and Systems*, 2014.

[6] J. Xiao and X. Ji, "Automatic generation of high-level contact state space," *IJRR*, vol. 20, no. 7, 2001.

[7] X. Ji and J. Xiao, "Planning motions compliant to complex contact states," *IJRR*, vol. 20, no. 6, 2001.

[8] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *IJRR*, vol. 23, 2004.

[9] M. T. Mason, *Mechanics of Robotic Manipulation*. MIT Press, 2001.

[10] K. M. Lynch and M. T. Mason, "Stable pushing: Mechanics, controllability, and planning," *IJRR*, vol. 15, no. 6, 1996.

[11] K. M. Lynch, "Toppling manipulation," in *ICRA*, 1999.

[12] M. A. Erdmann and M. T. Mason, "An exploration of sensorless manipulation," *IEEE Journal of Robotics and Automation*, vol. 4, no. 4, 1988.

[13] T. Abell and M. Erdmann, "Stably supported rotations of a planar polygon with two frictionless contacts," in *IROS*, vol. 3, 1995.

[14] M. Erdmann, "An exploration of nonprehensile two-palm manipulation," *IJRR*, vol. 17, no. 5, 1998.

[15] T. Yamawaki and M. Yashima, "Randomized planning and control strategy for whole-arm manipulation of a slippery polygonal object," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 2485–2492.

[16] Y. Maeda, H. Kijimoto, Y. Aiyama, and T. Arai, "Planning of graspless manipulation by multiple robot fingers," in *ICRA*, 2001.

[17] Y. Maeda and T. Arai, "Planning of graspless manipulation by a multifingered robot hand," *Advanced Robotics*, vol. 19, no. 5, pp. 501–521, 2005.

[18] K. Miyazawa, Y. Maeda, and T. Arai, "Planning of graspless manipulation based on rapidly-exploring random trees," in *IEEE International Symposium on Assembly and Task Planning*, 2005.

[19] L. Han, J. C. Trinkle, and Z. X. Li, "Grasp analysis as linear matrix inequality problems," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 6, 2000.

[20] H. Hirukawa, Y. Papegay, and T. Matsui, "A motion planning algorithm for convex polyhedra in contact under translation and rotation," in *ICRA*, 1994.

[21] W. Meeussen, E. Staffetti, H. Bruyninckx, J. Xiao, and J. De Schutter, "Integration of planning and execution in force controlled compliant motion," *Robotics and Autonomous Systems*, vol. 56, no. 5, 2008.

[22] M. Y. Wang and D. M. Pelinescu, "Contact force prediction and force closure analysis of a fixtured rigid workpiece with friction," *Journal of manufacturing science and engineering*, vol. 125, no. 2, 2003.

[23] E. Catto, "Box2d: A 2d physics engine for games," 2013.