A Hierarchical Framework for Long Horizon Planning of Object-Contact Trajectories

A Hierarchical Framework for Long Horizon Planning of Object-Contact Trajectories

Bernardo Aceituno and Alberto Rodriguez

Department of Mechanical Engineering — Massachusetts Institute of Technology
<aceituno,albertor>@mit.edu

***Abstract*— Given an object, an environment, and a goal pose, how should a robot make contact to move it? Solving this problem requires reasoning about rigid-body dynamics, object and environment geometries, and hybrid contact mechanics. This paper proposes a hierarchical framework that solves this problem in 2D worlds, with polygonal objects and point fingers. To achieve this, we decouple the problem in three stages: 1) a high-level *graph search* over regions of free-space, 2) a medium-level randomized *motion planner* for the object motion, and 3) a low-level *contact-trajectory optimization* for the robot and environment contacts. In contrast to the state of the art, this approach does not rely on handcrafted primitives and can still be solved efficiently. This algorithm does not require seeding and can be applied to complex object shapes and environments. We validate this framework with extensive simulated experiments showcasing long-horizon and contact-rich interactions. We demonstrate how our algorithm can reliably solve complex planar manipulation problems in the order of seconds.**

I. Introduction

Dexterous manipulation is a versatile skill to solve robotic tasks in varied environments. Enabling it, however, requires our algorithms to reason about dynamics, geometry and, perhaps most challenging, contact mechanics. Contact-rich interaction is a key enabler to deploy robots for manipulation tasks. Unfortunately, the addition of contacts leads to discrete events within decision making (e.g. push from the side of the object or pull from the top). This makes the manipulation problem very challenging without any kind of guidance. These challenges have divided the state-of-the-art into two main approaches: 1) prescribing a set of *primitives* (e.g. grasping, pushing, or pivoting) and concatenating them to manipulate the object [1], [2], [3], [4], which restricts the set of solvable tasks, or 2) formulating a non-differentiable and non convex optimization problem that can solve the task implictly [5], [6], which is computationally intractable without careful seeding.

In this work, we explore the problem of efficiently planning object trajectories with robot contact interaction to solve a manipulation task. We refer to this problem as **Object-Contact Trajectory planning**. In contrast to using pre-designed primitives, we design a hierarchical algorithm that reasons about the object motion and the robot contact interaction as part of the plan. Our goal is to incorporate multi-contact interaction between the object, the robot, and the environment, without sacrificing efficiency in long-horizon trajectories. Solving this problem has three key challenges: 1) dynamics, which determine the motion of the object, 2) geometry, which constrains the set of actions and configurations, 3) and non-smoothness, which is required to describe rich contact interactions.

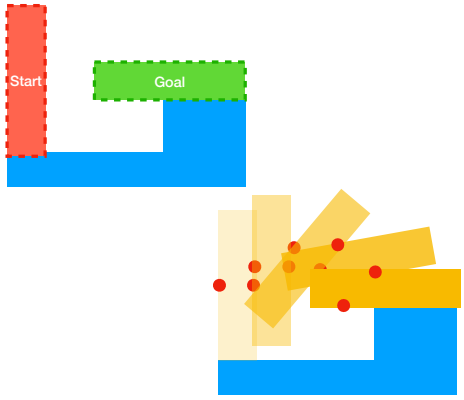


Fig. 1: Given an object, an environment, a start and a goal, our framework finds a manipulation plan to complete the task.

Our approach is to decompose the problem into three stages, which can be evaluated hierarchically and solved efficiently: 1) A dynamic-programming search over regions of the object configuration space, 2) A sampling-based motion planner for the object trajectory between connected regions, and 3) A contact-trajectory optimization to find a manipulator contact-trajectory (finger trajectory + forces) to execute the sampled object trajectory. These three stages can be solved iteratively in a hierarchy with backtracking, where the result of each stage informs the previous one. We name this approach *VI-MIQP*, acronym for Value Iteration (VI) with Mixed Integer Quadratic Programming (MIQP). We restrict our implementation in this paper to 2D scenes, over which it is simpler to reason about geometry. While this is a limitation, 2D manipulation problems encompass a wide range of skills that can be composed to solve practical tasks. The main contributions of this paper are:

- **Framework** to solve manipulation tasks with alternating sticking contact interactions using dynamic programming, sampling-based planning, and mixed-integer optimization.
- **Validation** of this approach applied to manipulation tasks in challenging environments without a prescribed object motion, contact schedule or primitives.

The remainder of this paper is organized as follows: Sec. II reviews concepts and literature relevant to this work. Sec. III provides an overview of our framework, its assumptions and properties. Sec. IV describes the proposed algorithm in detail, while Sec. V discusses its implementation. Sec. VI demonstrates the algorithm with simulated experiments, and we conclude in Sec. VII summarizing the contributions and limitations of the work.

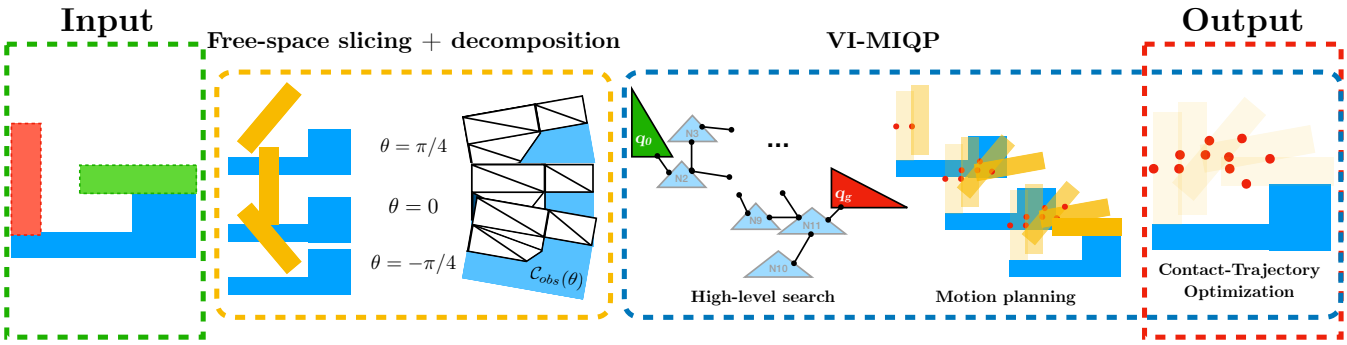


Fig. 2: Our proposed framework receives an object with a start and goal pose. We discretize the free-space in slices and decompose them into polygonal regions. Our algorithm performs a high-level search over the graph of regions, building a roadmap. At each new edge of the roadmap, we sample an object trajectory that connects the two regions. For each object trajectory, we optimize a manipulator contact-trajectory that executes the motion.

II. Background

In this section we review previous research relevant to this work and introduce concepts that we reference through the paper.

A. Motion Planning

Motion planning is a well studied problem within robotics. There are two main approaches relevant to this work. First, Sampling-based motion planning algorithms, such as RRT [7], which have been a successful approach to solving problems with plenty of local minima. On the other hand, optimization-based techniques, such as CHOMP or GPMP [8], [9], have also seen success at getting high-quality solutions in high-dimensional setups, although local in nature. In contrast to these two approaches, Dynamic Programming (DP) [10] can solve optimal control problems to optimality with the caveat of high-computational complexity, which restricts this approach to short horizon problems.

B. Manipulation Planning

The most common approach to manipulation is planning motions with *primitives*, such as grasping [11], pushing or pivoting. Many models can accurately optimize motions with these primitives [12]. Different primitives can be concatenated to manipulate a simple object [4]. A key limitation, however, is that primitives are specific to an object and an environment, which makes them hard to generalize to different tasks. Another line of research has focused on optimizing the contact interaction as part of the task [5], [6], [13], without assuming a primitive. However, optimizing contact interaction requires scheduling contact modes along the motion, which leads to combinatorial complexity, or modeling contacts implicitly through complementarity conditions, which are not differentiable. This makes it very challenging to optimize a manipulation plan without making assumptions on the contact interaction or providing significant seeding [14].

C. Manipulation on Long-Horizons

Recent work has tried to alleviate these issues through sampling-based motion planning [15], which removes assumptions on the contact schedule and object motion. However, this approach is unable to guarantee the quality of its solution. Morever, as with RRT, this approach will struggle to reason globally on long-horizons, leading to slow computation. Other recent work has shown that the robot *Contact-Trajectory* can be globally optimized efficiently [16], [17], with the caveat that the object trajectory must be prescribed. Our work draws inspiration from these two philosophies and will aim to find object motions using a sampling-based approach, guided by a high-level search, and optimize the robot contact-trajectory using mixed-integer optimization.

III. Problem Setup

In this section we provide an overview of our framework inputs and discuss its assumptions. Given a polygonal object, our algorithm will find a sequence of contact interactions that move it toward a goal pose.

A. Inputs and Notation

Our algorithm receives as inputs the initial and final pose of the object and the geometries of the object and environment. We introduce the following notation and variables:

1) **Object:** A polygonal rigid body $\mathcal{O}$ with N_v vertices and N_f facets in a workspace $\mathcal{W}$. Each facet $\mathbb{F}_f$ has 2 vertices, with nominal positions $\mathbf{v}_v^f$, with a corresponding friction cone $\mathbb{FC}_f$, represented with 2 rays.
2) **Trajectory:** A set of object poses over discrete time steps, the length of the plan T is not known a-priori. We describe each pose, at a time step t, as $\mathbf{q}(t) \in \mathcal{C}$, where $\mathcal{C}$ is the configuration space of the object. $\mathcal{C}$ corresponds to the x,y,θ coordinates of $SE(2)$. The starting configuration is $\mathbf{q}_0$ and the goal configuration is $\mathbf{q}_g$.
3) **Manipulator:** A set of N_c contacts points. We describe the c_{th} contact-point, at time-step t, as $\mathbf{p}_c(t) \in \mathcal{W}$, where $\mathcal{W}$ is the workspace.The workspace corresponds to each position (x,y) that can be reached by the robot. We describe the force applied by the manipulator to the object, at time step t, as $\lambda_c(t) \in \mathbb{R}^2$.
4) **Environment:** A polygonal environment $\mathcal{E}$ with N_e facets, described as planes with friction cones $\mathbb{FC}_e$. Additionally, we segment the free-space between the object and environment into N_R convex polytopic regions $\mathcal{R}_r = \{\mathbf{x} \in \mathcal{W} \mid A_r\mathbf{x} < b_r\}$. We label the force between the environment and vertex v of the object as λ_v^e and the force applied by the vertex v^e of the environment into the object facet f as $\lambda_{v^e}^f$.

Then, we operate our planning on the following spaces:

1) **Configuration Space:** A set $\mathcal{C}$ of x, y, θ coordinates in $SE(2)$ where the object can move. The set of configurations where the object penetrates the environment is called

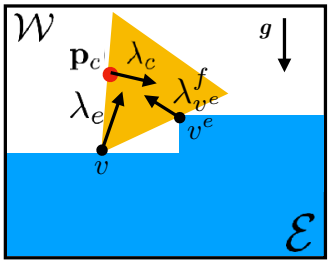


Fig. 3: Elements of our problem: a polygonal object $\mathcal{O}$, a point-finger manipulator $\mathbf{p}_c$ and an environment $\mathcal{E}$. The contact between object and environment leads to a reaction force.

C-obstacles or $\mathcal{C}^{obs}$ and is defined by the Minkowski sum between the object geometry (across all orientation of the configuration space) and the environment edges.

2) **Free-Space:** a set of configurations $\mathcal{C}^{free}$ where $\mathcal{O}$ does not penetrate the environment, defined as $\mathcal{C}^{free} = \mathcal{C} - \mathcal{C}^{obs}$, which is a subset of $SE(2)$. For practical purposes, we will discretize this space over the orientation component with a sample S $\mathcal{C}-$slices.

A diagram describing these elements, at a fixed time-step, can be seen in Fig. 3.

B. Modeling Assumptions

In order to build the manipulation planning problem, we make the following assumptions:

1) Objects are rigid, with uniform contact surfaces (such that line contacts can be approximated by contact with two vertices), and approximated as combinations of "simple" polygons.
2) Object motions occur at low speeds, such that high-order inertial effects are negligible.
3) Robot fingers have small masses, such that there are no impacts between the robot and the object.
4) Interactions between the object and robot are a sequence of alternating sticking contacts, at which friction is captured by a cone represented by two rays.

The upcoming sections will describe how these modeling decisions translate into our planning framework.

IV. Hierarchical Framework

In this section, we present our hierarchical approach to long-horizon manipulation planning. Each subsection describes a stage of the hierarchy and provides technical and implementation details. Our framework has three stages, which help alleviate some of the key issues of contact-rich manipulation:

1) **High-Level:** The first stage constructs a roadmap over a decomposition of the free-space into convex regions. We represent this roadmap as a graph G with edges E. We search over this roadmap to find a path from the start to goal configuration.
2) **Motion Level:** The second stage finds object trajectories that connect edges of the roadmap. These trajectories are concatenated while searching on the graph. If there is not a feasible trajectory corresponding to an edge, then we remove the edge from the roadmap.
3) **Contact Level:** This stage verifies that the motion-level trajectory can be executed with the manipulator. This problem is known as Contact-Trajectory Optimization. If the optimization has no solution then we reject the object trajectory as infeasible.

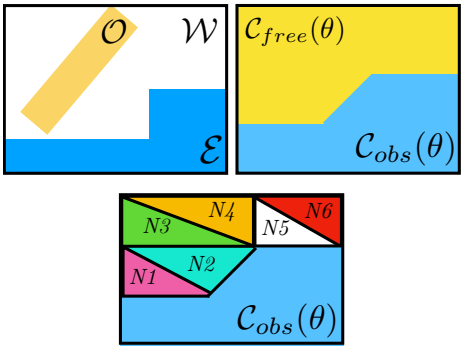


Fig. 4: Free-space decomposition for high-level planning. Top: Workspace $\mathcal{W}$ and free-space slice at $C^{free}(\theta)$. Bottom: convex decomposition of the slice $C^{free}(\theta)$.

These three stages are evaluated in a hierarchy where stage informs the previous one. Our approach leverages dynamic programming, sampling-based motion planning, and mixed-integer optimization to construct the full pipeline, illustrated in Fig. 2.

A. High-Level Search

An effective approach to guide a long-horizon planning problem is to build a high-level roadmap that outlines a coarse plan for a low-level finer planning stage. This outline ensures a certain level of optimality on the resulting trajectory. Without a roadmap, sampling-based planning tools, such as RRT, take a long time to solve long-horizons problems, and the resulting solution often has poor quality.

1) Roadmap construction: To construct our roadmap we operate under the following decomposition of the free-space:

- Slicing: we discretize the free-space of the object, which lies in $SE(2)$, into slices of constant orientation. We refer to $\mathcal{C}^{free}$-slice of orientation θ as $C^{free}(\theta)$. Each slice is a 2D region of coordinates in $\mathbb{R}^2$. We showcase some examples in Fig. 4 (left).
- Segmentation: we further decompose each $\mathcal{C}^{free}$-slice into convex regions, as demonstrated in Fig. 4 (right).

We use each of the convex regions as a node in our graph G, where edges E are the overlapping side or area with regions in the same slice or in adjacents slices[1]. We label the node of the i_{th} region as N_i, where $E_{i,j}$ is the edge between N_i and N_j. This graph is the roadmap used to plan our high-level path. In practice, we compute $C^{free}(\theta)$ as the Minkowski sum $C^{free}(\theta) = \mathcal{E} \bigoplus R(\theta)\mathcal{O}$, where $R(\cdot)$ is a rotation matrix, and perform the convex decomposition using Delaunay triangulation.

[1]The separation between slices can cause two regions not to overlap, which can lead to path non-existence. Hence, this makes the resulting plan dependent on the resolution of the slicing

2) Dynamic programming: Once we construct the roadmap G,E we need to search over the nodes to find the best path to the goal and determine the distance from each node to the goal. Since we need to find an object motion as we search through the graph, as part of the motion-level planning, it is more reasonable to pre-compute a value function for each edge $V(E_{i,j})$ using a heuristic. With this value function, we iteratively explore different paths in subsequent stages.

We start by finding start and goal nodes such that:

$$\mathbf{q}_0 \in N_0,\, \mathbf{q}_g \in N_g,$$

and set

$$V(E_{i,j}) = \infty,\, V(E_{i,g}) = d(i,g),$$

where $d(i,j)$ is the normalized $SE(2)$ distance between the center of N_i and N_j[2]. Finally, we determine the value of each node by running value iteration on the graph, using the Bellman equation [10]:

$$V(E_{i,j}) = d(i,j) + \min_k V(E_{j,k}) \;,\; k \neq j$$

This relation is evaluated at each node iteratively until convergence. If $V(E_{0,k}) = \infty$ then there is no path from the start to the goal under the current slicing.

3) Forward pass: Finally, after computing the value of each node, we proceed to explore the tree in a forward pass. We create a list of nodes $N = [N_0]$, including the start node, and a list of configurations $q = [\mathbf{q}_0]$, including the start configuration. We then proceed to explore the graph. We set $N_t = N_0$ and choose:

$$N_{t+1} = \operatorname*{argmin}_k V(E_{t,k})$$

Then, we verify with the motion-level plan if there is a trajectory from node N_t to N_{t+1}. We then follow the logic:

1) If the motion-level planner finds a feasible motion $\mathbf{q}_{new}$ and contact-trajectory for the entire motion $\mathbf{p}_c, \lambda_c$: we push the solution $q = [q, \mathbf{q}_{new}]$, $N = [N, N_{t+1}]$, and set $N_t = N_{t+1}$.
2) If the motion-level planner cannot find a trajectory, then we remove the edge from the graph by setting $V(E_{t,t+1}) = \infty$.

If all the edges of node N_t have $V(E_{t,t+1}) = \infty$, then we remove N_t from the list of nodes and pop the latest $\mathbf{q}_{new}$ from the list of configurations. We continue this process until $N_t = N_g$, at which point we run the motion-level planner to find a motion to the goal.

B. Motion-Level Planning

While running the forward pass through the optimal graph, at each edge with nodes N_t and N_{t+1}, we need to verify if there is a motion for the robot to move the object. This requires finding a feasible contact-trajectory for the given path, which results in a non-smooth and nonlinear optimization problem, such as in [5]. This problem is hard to solve and usually requires a warm-start and depends heavily on numerical conditioning, which often makes it impractical.

In our case, we decouple the problem by randomly sampling object motions between N_t and N_{t+1}, since each node is associated with a convex polygon, using the function $Sample(\cdot)$, which randomly samples an $SE(2)$ configuration within a convex polygon. For each sample we solve an optimization problem $CTO()$ to find a contact-trajectory, discarding the sample if $CTO()$ is infeasible. We sample up to N_{MAX} trajectories before deciding an edge is not feasible. We summarize this procedure in algorithm 1.

Algorithm 1 RRT_{CTO}

Require: $\mathcal{O}$, $\mathcal{E}$, N_t, N_{t+1}, N_{MAX}
  trial = 0
  while $trial < N_{MAX}$ **do**
    $\mathbf{q}_{new} = [\mathbf{q}_0, Sample(N_t), Sample(N_t \cap N_{t+1})$
    $\mathbf{p}_{new}, \lambda_{new} = CTO([\mathbf{q}, \mathbf{q}_{new}])$
    if Success **then**
      return $\mathbf{q}_{new}, \mathbf{p}_{new}, \lambda_{new}$
    else
      trial = trial + 1
    end if
  end while
  return Failure

Note that $CTO(\cdot)$ is called over the entire object trajectory, including previous nodes. We remark that the $Sample(\cdot)$ function needs to account for vertices and edges to have completeness, since each node is tied to a convex polygon. For this reason, we sample: 1) uniformly inside the polygon with probability p_1, 2) uniformly inside of a random polygon facet with probability p_2, and 3) in a random polygon vertex with probability $1 - p_1 - p_2$. This guarantees that the sampler will consider trajectories where the object traverses across facets or vertices of the node. We note that this is particularly important since environmental interaction will occur only at facets of vertices of each node. In practice, we always sample along a straight line during the first trial, to bias towards a smooth motion, and sample freely in subsequent trials.

C. Contact-Trajectory Optimization

Once we sample an object trajectory, we need to verify that the manipulator can execute it. To do this, we need to solve an optimization problem that verifies that contact dynamics allow for such execution, under the components depicted in Fig. 5. One effective method to solve this problem is to apply Mixed-Integer Programming (MIP) once the object trajectory is prescribed, as in our case.

The mixed-integer programming formulation receives the object-trajectory as an input and returns the contact-trajectory of the manipulator. Note that once we sample a trajectory we also obtain the contact schedule of the environmental contacts, encoded by $\mathbb{FC}_e$, and the manipulator free-space regions $\mathcal{R}_r$. To achieve this, we apply the following modeling assumptions, based on [16], which models constraints using the following convex relations:

a) Quasi-Dynamics: We model the linear object dynamics, at each time-step t, under the quasi-dynamic relation[3]:

[2]We normalize this distance to avoid over-emphasis on the rotation component.

[3]we distinguish between $\lambda_v^e(t)$ and $\lambda_v^e(t)$ because environmental forces have pre-fixed friction cones while robot friction-cones need to be found by the model.

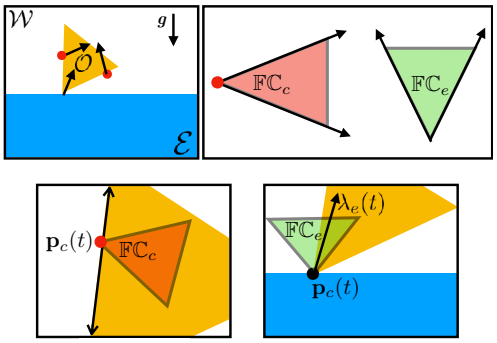


Fig. 5: Once the object trajectory is sampled, we optimize a set of finger motions and contact forces that execute the trajectory. This problem involves the finger trajectories, applied forces, reaction forces, and the object facets and friction cones.

$$m[\ddot{\mathbf{q}}_x(t),\ddot{\mathbf{q}}_y(t)]^T=\sum_c\lambda_c(t)+\sum_v\lambda_v^e(t)+\sum_{v^e}\lambda_{v^e}^f(t)-m\mathbf{g}, \tag{CT1}$$

where $\mathbf{g}$ is the gravity vector, note that this equation disregards Coriolis effects. Here, we compute time derivatives using second-order finite differences. For rotational dynamics, since torques require a non-convex bilinear cross product, we use the following approximate model:

$$I\ddot{\mathbf{q}}_\theta(t)=\sum_c\tau_c(t)+\sum_v R(\mathbf{q}_\theta(t))\mathbf{v}_v\times\lambda_v^e(t)+\sum_{v^e}\mathbf{v^e}_v\times\lambda_{v^e}^f(t), \tag{CT2}$$

where $\tau_c\approx(\mathbf{p}_c-[\mathbf{q}_x,\mathbf{q}_y]^T)\times\lambda^c$ and $R(\cdot)$ is a rotation matrix. We approximate the bilinear cross product $(\times)$ in τ_c using the McCormick Envelopes technique [18]. McCormick Envelopes are a piecewise outer approximation of the bilinear product $w=x\cdot y$. Finally, we determine the value of environmental forces via the friction cone:

$$\lambda_v^e(t)\in\mathbb{FC}_e^v(t),\lambda_{v^e}^f(t)\in\mathbb{FC}_f(t), \tag{CT3}$$

where the value of $\mathbb{FC}_e(t)$ is determined once the trajectory is sampled.

b) Geometry: We need to constrain the manipulator fingers to only lie in the free-space between the object and the environment. To achieve this, we introduce a binary decision matrix $\mathcal{H}\in\{0,1\}^{N_c,N_R,T}$ that maps each contact $\mathbf{p}_c(t)$ to a convex region $\mathcal{R}_r(t)$, at each time-step t. This is encoded by the mixed-integer linear constraint

$$\mathcal{H}(c,r,t)=1\Rightarrow\mathbf{p}_c(t)\in\mathcal{R}_r(t), \tag{CT4}$$

where the $\Rightarrow$ operator is encoded using the big-M formulation [19] .

c) Contact mechanics: We encode the hybrid mechanics of applied contact by introducing a binary decision matrix $\mathcal{T}\in\{0,1\}^{N_c,N_f,T}$. This matrix map each manipulator contact to a facet of the object and a friction cone as:

$$T(c,f,t)=1\Rightarrow\begin{cases}\mathbf{p}_c(t)\in\mathbb{F}_f(t),\\ \lambda_c(t)\in\mathbb{FC}_f(t),\end{cases} \tag{CT5}$$

and

$$\sum_f T(c,f,t)=0\Rightarrow\lambda_c(t)=0, \tag{CT6}$$

which are all linear constraints with big-M formulation.

We aggregate these sets of constraints into a single optimization problem. This results in the following mixed-integer optimization problem:

$$\textbf{CTO}:\underset{\mathcal{T},\mathcal{H},\mathbf{p},\lambda}{\text{minimize}}\ J=\begin{bmatrix}\mathcal{T} & \mathcal{H} & \mathbf{p} & \lambda\end{bmatrix}Q\begin{bmatrix}\mathcal{T}\\ \mathcal{H}\\ \mathbf{p}\\ \lambda\end{bmatrix}+q^T\begin{bmatrix}\mathcal{T}\\ \mathcal{H}\\ \mathbf{p}\\ \lambda\end{bmatrix}$$

subject to:

1) For time-step $t=1$ to $t=T$:
   a) Quasi-Dynamics (CT1)-(CT2).
   b) Environmental Contact (CT3).
   c) For fingers $c=1$ to $c=N_c$
      - Non-Penetration (CT4).
      - Contact-Trajectory Assignment (CT5)-(CT6).

This optimization problem has the following properties: 1) Given a convex cost function, we will always find the optimal solution, and 2) The model only reports infeasibility if the original non-convex problem is also infeasible, thanks to the McCormick envelope approximation used in (CT2). This is comes with the drawback of combinatorial complexity, which grows exponentially with the number of binary variables. However, we find that this optimization problem can be solved very quickly in practice, on the order of tens of milliseconds.

Moreover, this formulation also allows for the inclusion of additional constraints, such as kinematics or robustness margins [20], provided that these can be formulated in a mixed-integer linear fashion. In the final $CTO(\cdot)$ call of the algorithm, when reaching the goal pose, we add a quadratic cost function that minimizes the applied force, smooths the finger trajectories, and maximizes contact robustness:

$$J=\sum_{t=1}^{T}\sum_{c=1}^{N_c}||\boldsymbol{\lambda}_c(t)||^2+||\ddot{\mathbf{p}}_c(t)||^2-\beta_c(t)$$

The term β is a lower-bound convex-approximation of the distance between each contact-force and the border of its friction cone, computed as in [14], [4], this term has to be linear in order for the cost-function to be convex. We depict this β term in Fig. 6.

V. VI-MIQP

Putting all the previous stages together, we formulate our planning framework under the stages outlined in the previous section. This algorithm receives an object shape, initial pose, goal pose, and free-space $\mathcal{C}^{free}$ sliced over orientations. We summarize our proposed framework in algorithm 2.

This formulation has a few benefits and theoretical properties:

- First, the dynamic programming over a roadmap yields an optimal solution to the high-level plan, conditioned on the

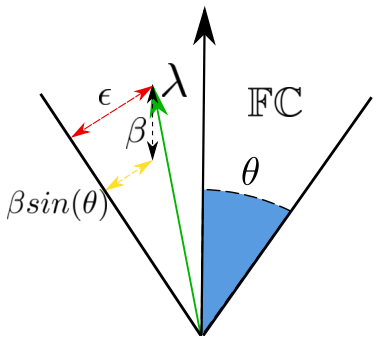


Fig. 6: Contact robustness margin: we compute a convex inner approximation of the stability margin of our motion.

Algorithm 2 VI-MIQP

Require: $\mathcal{O}, \mathcal{E}, \mathbf{q}_0, \mathbf{q}_g$
  G,E = Delaunay($\mathcal{C}^{free}$) ▷ Roadmap construction
  $V(E_{i,j}) = \infty, \forall\, i,j$
  $V(E_{i,g}) = 0, \forall\, i$
  while V not converged **do**
    $V(E_{i,j}) = d(i,j) + \min_k V(E_{j,k})$ ▷ Value iteration
  end while
  $N_t = N_0$ ▷ Starting node
  $N = [N_t]$ ▷ Starting list
  while $N_t \neq N_g$ **do** ▷ Forward pass
    for $N_{t+1} = \text{argmin}\ V(E_{t,t+1})$ **do**
      $\mathbf{q}_{new}, \mathbf{p}_{new}, \lambda_{new} = RRT_{CTO}(q, N_t, N_{t+1})$
      if Success **then**
        $\mathbf{q} = \text{push}(\mathbf{q}, \mathbf{q}_{new})$, $\mathbf{p} = \mathbf{p}_{new}$, $\lambda = \lambda_{new}$
        $N = [N, N_{t+1}]$ ▷ Adds edge to plan
        $N_t = N_{t+1}$
        break **for**
      else
        $V(E_{t,t+1}) = \infty$ ▷ Remove edge
      end if
      if $V(E_{t,t+1}) = \infty\ \forall N_{t+1} \in Neigh(N_t)$ **then**
        $N_t = N_{t-1}$ ▷ Removes node
        pop(q), pop(N)
      end if
    end for
  end while

resolution of the slicing of the free-space and in the limit of sampling of the motion planner.

- Thanks to the mixed-integer optimization formulation, we can guarantee that the contact-trajectory found by $CTO(\cdot)$ is **globally optimal**, conditioned to the object trajectory found in the motion-level.

These properties are very useful to understand the feasibility of a task and the quality of the solution.

VI. Validation and Results

To demonstrate the capabilities of our framework, we implement algorithm 2 and asses its application to a set of long-horizon manipulation problems. First, we aim to validate the model's ability to find solutions to complex manipulation tasks and detect infeasible ones. We then show how the algorithm performance scales over problems with varying complexity.

We generate all the trajectories in MATLAB R2021b, running on an Intel Core i9 laptop with Mac OS X Big Sur. We use Gurobi 9.1.0 [21], an off-the-shelf optimization software, as our MIP solver. All of our tests are done with a $\mathcal{C}^{free}$ sampled in 7 slices between $-90°$ and $90°$[4]. We use piecewise McCormick envelopes of 12 segments. We segment the free space of each task into convex regions $\mathcal{R}$ using Delaunay triangulation. We compute second derivatives within the mixed-integer model with the backwards-Euler scheme for simplicity and numerical stability.

A. Simulated Validation

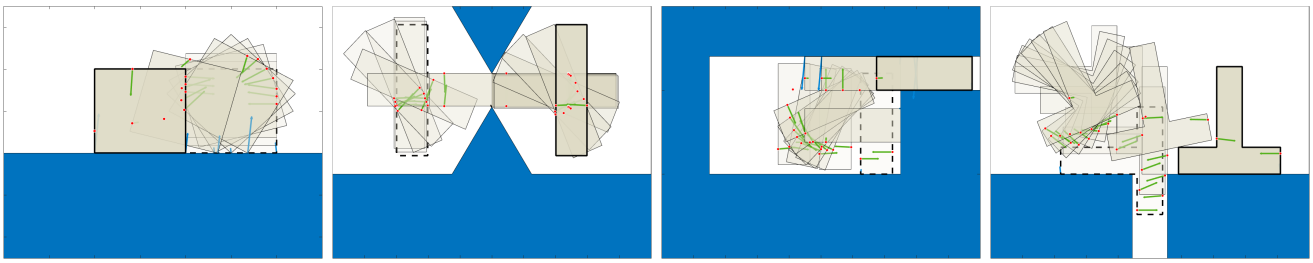
We start by validating the functionality of the algorithm in several manipulation problems. In particular, we show its ability to solve problems that demand long horizon and multi-contact reasoning. These tasks require alternating contact switching with the robot and the environment. All the tasks are solved with a 2-finger manipulator with kinematic constraints based on an ABB YuMi robot, encoded in $CTO(\cdot)$. For this, we use three objects: 1) a block, 2) a rectangle, a 3) a non-convex T object. The contacts between all surfaces have a friction coefficient of $\mu = 0.1$. We solve the following manipulation problems:

Block Pivoting: we ask the planner to rotate a block $-90°$ in the sagittal plane and slide it $200mm$ in the $-X$ direction (Fig. 7a). Due to the low friction with the surface, our algorithm finds a trajectory that: 1) grasps the block with two fingers, slowly rotating in the process, 2) pivots the object with one finger, while pushing it to the goal, and 3) uses the two fingers to immobilize in the final goal pose. This trajectory demonstrates the contact-rich nature of our approach, since the solution to a problem can choose to grasp an object and then proceed to use a single finger to complete the task. This trajectory is found in 0.2s.

Rectangle across Narrow Corridor: we ask the planner to grasp a rectangle and move to the other side of a region with a very narrow corridor in the moddle (Fig. 7b). The algorithm performs a strategy of: 1) grasping the object, 2) rotating the object $-90°$, 3) sliding the object through the corridor with one finger, 4) grasp the object the two fingers, and 5) pivot it back for $0°$. This trajectory demonstrates the long-horizon reasoning that our planner can achieve, considering contact switches and global path. This type of trajectory would likely require a very large amount of exploration from randomized sampling-based planner. This trajectory is found in 7.2s.

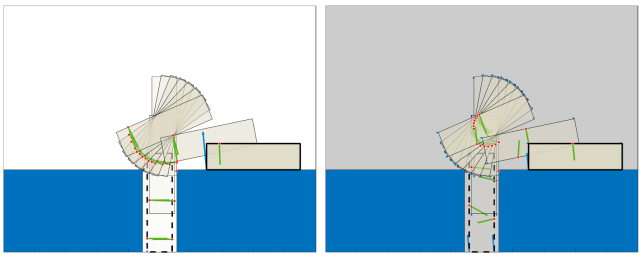
Rectangle Peg-in-wall: we ask the planner to pick a rectangle from the ground and insert it into a tight hole in a wall, rotated by $-90°$ (Fig. 7c). The algorithm performs a strategy of: 1) grasping the object, rotating it while grasped, 2) pushing it to the wall with one finger, 3) sliding it up to the "ceilng", and 4) pushing it with the two fingers, against the ceiling contact, to insert the object in the wall. This trajectory demonstrates multi-contact interaction with the robot and the environment. Here, the wall and ceiling contacts are used to enable more stability and to make the trajectory geometrically feasible. This trajectory is found in 2.5s.

[4]This choice is made to leave a gap of $30°$ between slices, which is enough to find paths in the shapes used. Environments with narrower paths or more non-convex shapes may require further slicing.



(a) Pivoting a block and sliding (b) Moving in a narrow corridor (c) Peg-in-wall for a rectangle (d) Unpegging a T from a hole

Fig. 7: Long-horizon manipulation tasks solved with our algorithm. Green dots correspond to the point-fingers of our manipulator. Doted object contour corresponds to the initial pose $\mathbf{q}_0$ and bold object contour corresponds to goal pose $\mathbf{q}_g$.



(a) Sagittal unpeg-from-hole (b) Traversal unpeg-from-hole

Fig. 8: Comparison between the same task solved in a sagittal and traversal environment

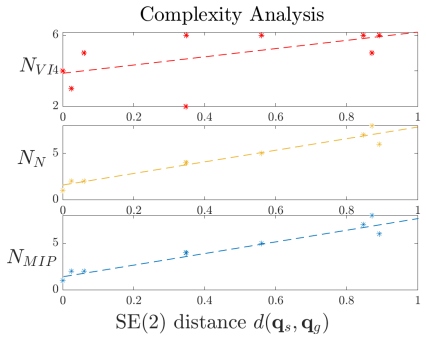


Fig. 9: Complexity analysis for our model

T Unpeg: we ask the planner to pick a T object from a hole and rotate it by $90°$ into the floor (Fig. 7d). The algorithm finds a strategy of: 1) grasping the object from the hole, 2) pivoting it while grasped, 3) switching the grasp configuration, and 4) placing the T in the goal location. This tasks demonstrates the ability of our algorithm to reason over non-convex geometry in the objects and environment. This trajectory is found in 6.3s.

As a reference, all trajectories are optimized in the range of $0.2\,\mathrm{s}$ to $7.2\,\mathrm{s}$. We stress the ability of $CTO(\cdot)$ to report when a task is infeasible, either because it requires an additional contact force or because it is geometrically infeasible, which can help the planner quickly discard edges and find a path. We note that all these tasks are performed in the sagittal plane.

B. Application to Sagittal and Traversal tasks

A key question is how this algorithm performs across the sagittal $-XZ-$ plane and one problem in the traversal $-XY-$ plane. The traversal plane adds the presence of Coulomb friction forces in the object surface, following the maximum discipation principle. We can easily model this patch contact in our $CTO(\cdot)$ problem by transforming the friction cone at each object vertex to the vector:

$$\mathbb{FC}_e^v(t) = \frac{-\mu_e}{\sqrt{\dot{v}_x^2(t)+\dot{v}_x^2(t)}}[\dot{v}_x(t),\dot{v}_y(t)],$$

where $v(t) = [v_x(t), v_y(t)]$ is the position of the object vertex v at time-step t. We then contrast how our model performs in both traversal and sagittal scenarios with the following tasks:

Saggital Unpeg: we ask the planner to pick a rectangle from a hole and rotate it by $90°$ into the floor (Fig. 8a). The algorithm finds a strategy of: 1) pushing the object to one side of the hole to create some clearance, grasping it up from the hole, 2) pivoting it with respect to one of the vertices of the hole, 3) grasping it outside of the hole, and 4) dropping it into the goal location in the floor. Similar to before, this trajectory demonstrates multi-contact interaction and dynamics with the robot and the environment. Here, the planner effectively finds a space clearance to pivot the object against the environment and then use gravity to drop it to the goal. This trajectory is found in 3.2s.

Traversal Unpeg: we ask the planne to pick a rectangle from a hole and rotate it by $90°$ into the floor (Fig. 8b). The algorithm finds a similar strategy of: 1) pushing the object to one side of the hole to create some clearance, grasping it up from the hole, 2) pivoting it with respect to one of the vertices of the hole, 3) grasping it outside of the hole, and 4) pushing it into the goal location in the floor. This trajectory shows how the presence of traversal contact, dictated by the maximum dissipation principle, forces the planner to find a different contact schedule for the trajectory. This trajectory is found in 9.6s.

These trajectories are optimized in the range of $3.2\,\mathrm{s}$ to $9.6\,\mathrm{s}$. This demonstrates the versatility of our model when accounting for new constraints and dynamic effects.

C. Complexity Analysis

One natural question is understanding the computational complexity of this algorithm and how efficient it can be at solving problems of different scale. Assessing the speed of a mixed-integer optimization problem can be very challenging, since off-the-shelf tools will use different techniques to solve the problem. Nevertheless, we can analyze other metrics that relate to the complexity of the problem, such as number of nodes of explored N_N, $CTO(\cdot)$ calls N_{MIP}, and value iteration steps N_{VI}.

We come back to the task of "rectangle across a corridor" and sample several intermediary goals. We measure the $SE(2)$

distance between $\mathbf{q}_s$ and each of these goals and then run our algorithm, recording: number of value iteration steps, number of nodes explored, and number of MIP calls. We report the results in Fig. 9. We find the complexity our framework to grow approximately linearly with the distance to the goal. This comes with the caveat that MIP calls can often take up to a second, becoming the main source of delays. However, this also suggests that our approach can scale well to very long-horizon tasks.

VII. Discussion

In this paper, we have presented a hierarchical framework to solve long-horizon object-contact trajectories for manipulation tasks in 2D polygonal environments. We decouple the problem into three stages by searching over a high-level roadmap of a discretized configuration space, planning motion over local free-space regions, and optimizing contact interactions over sampled trajectories. This provides us with a resolution complete algorithm with optimality bounds on high-level search and contact-trajectory. Our framework accounts for object shape, environment contacts, and allows for the inclusion of additional constraints within the contact-trajectory optimization.

We implement this framework using off-the-shelf optimization software and MATLAB. We validate its application on a variety of manipulation tasks with varying time horizons, physics constraints, and geometries. Our planner returns a solution between $0.2\,\mathrm{s}$ to $9.6\,\mathrm{s}$, varying with the horizon of the task and complexity of the problem. Our long-term vision is that the execution of these plans will be robust to uncertainty using geometry and certification, as proposed in [22], and supported by a low-level tactile controller, as demonstrated in [4].

a) Algorithm limitations: This algorithm has a few limitations that constrain its scope. First, the use of mixed-integer programming leads to combinatorial complexity in the contact-trajectory optimization step. Second, we constrain our manipulator to perform alternated sticking, which eliminates the possibility of in-hand sliding as part of our plans. Finally, the sampling nature of this approach will usually lead to non-smooth object trajectories, with jerky motion. Moreover, extending this approach to 3D tasks is not straightforward. Since we rely on free-space slicing over the orientation component to construct a roadmap. A generalization to 3D would require a less naive alternative decomposition of the free-space into convex regions.

b) Future work: The first set of extensions to this work come from its limitations. First, we can extend the contact-trajectory optimization model to account for sliding motion, with the caveat that it would require more binary variables. Secondly, the trajectories from this model could be post-processed through a nonlinear optimizer, fixing the contact schedule, in order to get smoother object motions.

While this approach is not directly applicable to 3D tasks, we can sequentially compose planar tasks between sagittal and traversal environments in order to generate a "2.5D" behavior. In such case, we could also slice over the depth dimension of the space and generate a graph of convex regions over two rotation dimensions, leading to manipulation skills like those of [4]. Another natural extension would be to combine the solutions of this algorithm with a state-of-the-art feedback controller [23], [13], [24]. The use of contact forces could rely on feedback from localized tactile sensing [25], [4].

c) Source Code: The entire source code used as part of this work is publicly available on GitHub: https://github.com/baceituno

Acknowledgments

We would like to thank members of the MCube Lab for insightful discussions and advice during the development of this project.

References

[1] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *ICRA*. IEEE, 2016.

[2] N. Chavan-Dafle, R. Holladay, and A. Rodriguez, "In-hand manipulation via motion cones," in *RSS*, 2018.

[3] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *IROS*. IEEE, 2018, pp. 4238–4245.

[4] F. Hogan, J. Ballester, S. Dong, and A. Rodriguez, "Tactile dexterity: Manipulation primitives with tactile feedback," in *ICRA*, 2020.

[5] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *IJRR*, 2014.

[6] Z. Manchester and S. Kuindersma, "Variational contact-implicit trajectory optimization," in *ISRR*. Springer, 2017.

[7] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[8] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 489–494.

[9] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using gaussian processes and factor graphs." in *Robotics: Science and Systems*, vol. 12, no. 4, 2016.

[10] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 1.

[11] D. Prattichizzo and J. C. Trinkle, "Grasping," in *Springer handbook of robotics*. Springer, 2016, pp. 955–988.

[12] M. T. Mason, *Mechanics of robotic manipulation*, 2001.

[13] F. H. N. Doshi and A. Rodriguez, "Hybrid differential dynamic programming for planar manipulation primitives," in *ICRA*, 2020.

[14] B. Aceituno-Cabezas, C. Mastalli, H. Dai, M. Focchi, A. Radulescu, D. G. Caldwell, J. Cappelletto, J. C. Grieco, G. Fernández-López, and C. Semini, "Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization," *RA-L and ICRA*, 2018.

[15] X. Cheng, E. Huang, Y. Hou, and M. T. Mason, "Contact mode guided sampling-based planning for quasistatic dexterous manipulation in 2d," in *ICRA*, 2021.

[16] B. Aceituno-Cabezas and A. Rodriguez, "A global quasi-dynamic model for contact-trajectory optimization," in *RSS*, 2020.

[17] C. Chen, P. Culbertson, M. Lepert, M. Schwager, and J. Bohg, "Trajectotree: Trajectory optimization meets tree search for planning multi-contact dexterous manipulation," in *IROS*, 2021.

[18] G. P. McCormick, "Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems," *Mathematical programming*, 1976.

[19] T. Marcucci and R. Tedrake, "Mixed-integer formulations for optimal control of piecewise-affine systems," in *HSCC*, 2019.

[20] H. Dai, G. Izatt, and R. Tedrake, "Global inverse kinematics via mixed-integer convex optimization," in *ISRR*, 2017.

[21] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com

[22] B. Aceituno-Cabezas, J. Ballester, and A. Rodriguez, "Certified grasping," in *ISRR*, 2019.

[23] W. Han and R. Tedrake, "Local trajectory stabilization for dexterous manipulation via piecewise affine approximations," in *ICRA*, 2020.

[24] F. R. Hogan and A. Rodriguez, "Reactive planar non-prehensile manipulation with hybrid model predictive control," *IJRR*, 2020.

[25] L. D. J. R. E. Chuah, L. Epstein and S. Kim, "Bi-modal hemispherical sensor: A unifying solution for three axis force and contact angle measurement," in *IROS*, 2019.