Long-Horizon Manipulation of Unknown Objects via Task and Motion Planning with Estimated Affordances

Aidan Curtis*  Xiaolin Fang*  Leslie Pack Kaelbling  Tomás Lozano-Pérez  Caelan Reed Garrett

Abstract—We present a strategy for designing and building very general robot manipulation systems involving the integration of a general-purpose task-and-motion planner with engineered and learned perception modules that estimate properties and affordances of unknown objects. Such systems are closed-loop policies that map from RGB images, depth images, and robot joint encoder measurements to robot joint position commands. We show that following this strategy a task-and-motion planner can be used to plan intelligent behaviors even in the absence of a priori knowledge regarding the set of manipulable objects, their geometries, and their affordances. We explore several different ways of implementing such perceptual modules for segmentation, property detection, shape estimation, and grasp generation. We show how these modules are integrated within the PDDLStream task and motion planning framework. Finally, we demonstrate that this strategy can enable a single system to perform a wide variety of real-world multi-step manipulation tasks, generalizing over a broad class of objects, object arrangements, and goals, without any prior knowledge of the environment and without re-training.

I. INTRODUCTION

Our objective is to design and build robot policies that can interact robustly and safely with large collections of objects that are only partially observable, where the objects have never been seen before and where achieving the goal may require many coordinated actions, as in putting away all the groceries or collecting all the ingredients for a meal. Our goal is a policy that will generalize without specialized re-engineering or re-training to a broad range of novel objects, physical environments, and goals, but also be able to acquire whole new competencies, cumulatively, through incremental engineering and learning.

There is a broad appreciation of the importance of generality in design methods: trajectory optimization and reinforcement learning, for example, are both general tools that can address a large array of problems. However, the policies that are typically built with them are quite narrow in their domain of application. We seek, instead, systems generality, in which the focus is on the generality of a single policy. In this paper, we describe an approach for building such policies as deliberative systems and then instantiate it with an implementation that is able to manipulate novel objects in novel arrangements to achieve novel goals, both in simulation and on a real robot. It makes use of engineered as well as machine-learned modules for object segmentation, shape estimation of 3D object meshes, and grasp prediction, along with a state-of-the-art task-and-motion planner.

The operation of our system, called M0M (Manipulation with Zero Models), is illustrated in Figure 1. The goal is for all perceivable objects to be on a blue target region. Importantly, the system has no prior geometric models of objects and no specification of what objects are present in the world. It takes as input RGB-D images, which it segments and processes to find surfaces, colored target regions and object candidates (see Section VII-A). The goal for this task is communicated to the system by the following logical formula:

\[ \forall obj. \exists region. \text{on}(obj, region) \land \text{is}(region, \text{blue}). \]

This formula involves a relation \text{on} that the system knows how to achieve, by picking and placing, and perceptual properties (\text{is}) such as color that the system can compute from the input images (see Section IV). The goal does not reference any individual objects by name because, in our problem setting, the object instances have no names. Instead, goals existentially and universally quantify over the perceivable objects, which may vary substantially in number and properties across, and even within, problem instances.

Initially, two objects are purposefully hidden behind the tall cracker box so that the robot cannot perceive them. Finding only a single object on the table, the robot first picks and places the cracker box on the blue target region. It selects a placement for the cracker box on the blue region that is roughly in the middle of the region. Because the initial cracker box placement was planned without knowledge of

\*The first two authors contributed equally and are listed in alphabetical order. The authors are at CSAIL, MIT, USA: \{curtisa, xiaolinf, lpk, tlp, caelan\}@mit.edu.
the other two objects, upon observing the new objects, the robot intentionally moves the cracker box to a temporary new placement to make room for the tape measure and green cup. Finally, the robot plans a new placement for the cracker box that avoids collisions with the other two objects while also satisfying the goal. A video of this trial is available at: https://youtu.be/f-GCKQWuPyM. Additional experiments are described in Section VIII and in Appendix C.

MOM can perform purposeful manipulation for a general class of object shapes, object arrangements and goals, while operating directly from perceptual data, even in partially observable settings. Importantly, the system is designed in a modular fashion so that different modules can be used for perceptual tasks such as segmenting the scene or choosing grasps on detected objects. Furthermore, new manipulation operations, such as pushing or pouring, can be added and immediately combined with existing operations to achieve new goals. Many more examples of MOM in operation are illustrated in the remainder of the paper and at https://tinyurl.com/open-world-tamp.

II. APPROACH

Any robot system that has an extended interaction with its environment, selecting actions based on the world state and the outcomes of its previous actions, can be seen at the most basic level as a control policy that maps a sequence of inputs (generally intensity and depth images and joint angles) into motor torques. It has been traditional to hand-design and implement such control policies. A classical strategy of multi-level model-predictive control with a general-purpose planner at the top level results in very robust behavior that generalizes over a wide range of situations and goals [1], [2], [3], [4], [5]. However, these approaches have traditionally required a substantial amount of prior knowledge of the objects in the world and their dynamics.

A relatively newer strategy for constructing such control policies is to learn them via supervised, imitation, or reinforcement-learning methods in simulation or real-world settings [6], [7]. These approaches are attractive because they require less human engineering but they make heavy demands for real-world training, which again poses a substantial development burden. In addition, these learned policies are often narrowly focused on a single “task”.

In this paper, we present a strategy for obtaining the best of both worlds: we encode fundamental, very generic, aspects of physical manipulation of objects in three-dimensional space in an algorithmic framework that implements a feedback control policy mapping sensory observations and a goal specification into motor controls. To instantiate this framework for a new domain one must provide:

- A description of the robot’s kinematics and a basic position trajectory controller;
- A characterization of the manipulation operations that robot can use; and
- A set of perceptual modules that estimate properties of objects that the system will interact with, which can generally be acquired via off-line training and shared over a variety of applications and robots.

Systems that instantiate this framework will immediately generalize without any re-engineering or re-training to a broad range of novel objects, physical environments, and goals. Due to the modularity of the architecture, they can also serve as a basis for acquiring whole new competencies, cumulatively, by adding new learned or engineered modules.

Our approach leverages the planning capabilities of general-purpose task and motion planning (TAMP) systems [8]. The key insight behind our approach is that such planners do not necessarily need a perfect and complete model of the world, as is often assumed; they only need answers to some set of “queries”, which can be answered by direct recourse to perceptual data. Existing TAMP systems that have been demonstrated in real-world settings, including our prior work, require known object instance 3D mesh models that can be accurately aligned to the observed data using human-calibrated fiducials or pose estimators, which restricts their applicability to known environments, often with only a few unique object instances [1], [2], [3], [4], [5]. Even several extensions to TAMP-based systems that actively deal with some uncertainty from perception (such as substantial occlusions) [9], [10], [11] require observations in the form of poses of known objects. This pose registration process is critical for these approaches for identifying human-annotated affordances, such as grasps and placement surfaces, and representing collision volumes during planning. However, we show that one can also fulfill these operations using only the observed point cloud, without the need for prior models. In this paper, we develop a strategy in which all such queries are resolved in sensory data, see Section V.

The system instance described in this paper constructs a “most likely” estimate of the current scene by segmenting it into objects that can then be used to estimate shapes, grasp affordances, and other salient properties. It then solves for a multi-step motion plan to achieve the goal given that interpretation, executes the first few steps of the plan, re-observes the scene, determines whether the goal is satisfied, and if not, re-plans.

We demonstrate, in simulation and on a real robot, that our system can handle objects of unknown types and a variety of goals. Even if it makes perceptual errors, which are often reflected in taking imperfect actions, it recovers from these problems by continually re-perceiving and re-planning. We experiment with different implementations of perceptual modules, illustrating the importance of modularity for the overall flexibility and extensibility of the system.

III. RELATED WORK

We have already discussed relevant work in TAMP and in policy learning in Section II. We discuss relevant work in standalone perception in Section VII.

The most closely related work to ours in manipulation without shape models is by Gualtieri et al. [12]. Many of the components of our system, e.g., grasping and shape estimation, are analogous to theirs. They, however, assume a
task-specific rearrangement planner is provided and do not consider tasks that may require more general manipulation of the environment, e.g., moving an object out of the way, or the more complex goals enabled by a TAMP system.

A number of other approaches [13], [14] demonstrate systems that exploit the ability to gain information by interacting with objects. There is also a long line of work aimed at “interactive segmentation”, that is, using robot motions to disambiguate among object hypotheses when manipulating in clutter [15]. Object search under partial observability has been studied within a partially observable Markov decision process (POMDP) framework [16], [17], including work that learns policies that uncover hidden objects in piles [18].

IV. MANIPULATION WITH ZERO MODELS

We begin by describing the scope of the Manipulation with Zero Models (M0M) framework for prehensile manipulation, in which the robot moves objects using pick and place operations. We have previously implemented a variety of other manipulation operations, including pushing, pouring, scooping, and unscrewing bottle caps [19], [20]. In this paper, we focus on a single, prehensile, manipulation “mode”, which is to pick up objects in a rigid grasp, move them while not contacting any other objects, and then place them stably back onto a surface. The M0M system has already built into it the necessary descriptions of these operations for planning; an overview can be found in Section V.

This single domain description is used for all the objects, arrangements and goals. The description provided here is the most basic version of the framework; Section IX discusses the simplifications and assumptions inherent in this version and outlines strategies for relaxing them.

a) Robot-specific information: To apply M0M to a manipulation robot, it is necessary to provide a URDF description of the robot’s kinematics $R$ and a position configuration controller for the robot. The robot may have multiple manipulators that move sequentially.

b) Perceptual modules: An instantiation of M0M requires perceptual modules of several different types. The first modules take an RGB-D image as input:

- **Rigid objects**: Output is a set $O = \{o_1, \ldots, o_n\}$ of object hypotheses, each of which is characterized by an RGB partial point cloud.

- **Fixed surfaces**: Output is a set $S = \{s_1, \ldots, s_n\}$ of approximately horizontal surfaces (such as tables, shelves, parts of the floor) that could serve as support surfaces for placing objects.

Associated with each entity is an arbitrary reference coordinate frame, the simplest being the robot’s base frame. When we speak of $\text{Pose}(o, p)$, we mean a transform $p$ relative to the reference coordinate frame of $o$. This notion of a pose is useful for representing relative transformations but has no semantics outside the system. The remaining modules take an object point cloud $o$ as input:

- **Grasps**: Output is a possibly infinite sequence of transforms between the robot’s hand and the reference coordinate frame of $o$ such that, if the robot were to reach that relative pose with the gripper open and then close it, it would likely acquire a stable grasp of $o$.

- **Collision volume**: Output is a predicted volume regarding $o$, primarily used for reasoning about collision-avoidance and containment.

- **Object properties**: Output is a list of properties of $o$, which will be used in goal specifications. They can include object class, aspects of shape, color, etc.

These modules can (and do) use different representations for their computations. Some may use conservative over-estimates of the input point cloud to find volumes for avoiding collisions, others may use tight approximations of local areas to find candidate grasps, while others may use learned networks operating on the whole input to compute such affordances.

V. M0M USING PDDLSTREAM

Our implementation of M0M uses PDDLstream [5], an existing open-source domain-independent planning framework for hybrid domains. We have previously used PDDLstream to solve a rich class of observable manipulation problems; however, in our previous applications, object shapes were assumed to be known exactly. Other TAMP frameworks that provide a similar interface between perceptual operations and the planner through for example, suggestors [1] or a refinement layer [2], could also be used as the basis of an implementation of our approach.

A. Overview

PDDLstream takes as input models of the manipulation operations, in the form of Planning Domain Definition Language (PDDL) operator descriptions (see Figure 3), and a set of samplers (referred to as streams), which produce candidate values of continuous quantities, including joint configurations, grasps, object placements, and robot motion trajectories that satisfy the constraints specified in the vocabulary of the problem (see Figure 4). Critically, aside from a small declaration of the properties that their inputs and outputs satisfy, the implementation of each stream is treated as a blackbox. As a result, PDDLstream is agnostic toward both the representation of stream inputs and outputs as well as whether operations are implemented using engineering or learning techniques. This allows state-of-the-art machine learning methods to be flexibly incorporated, without modification, during planning where they will be automatically combined with other independent operations by the PDDLstream planning engine.

Problems described using the PDDLstream planning language can be solved by a variety of PDDLstream planning algorithms [5]. Because the PDDLstream planning engine is responsible for querying the perceptual operations (in the form of streams), it will automatically decide online which operations are relevant to the problem and how many generated values are needed. Furthermore, several
PDDLstream algorithms (e.g. the FOCUSED algorithm) will lazily query the perceptual operations in order to avoid unnecessary computation. As a result, the planner will not perform computationally expensive perceptual operations on images and point clouds to, for example, predict properties and grasps unless the segmented object or property have been identified as relevant to the problem.

Below we describe our PDDLstream formulation in detail so as to make the contract between perceptual operations and action descriptions precise.

B. PDDLstream formulation

In PDDL, an action is specified by a list of free parameters (:parameters), a precondition logical formula (:precondition) that must hold to correctly apply the action, and an effect logical conjunction (:effect) that describes changes to the state when the action is applied. Figure 3 gives the PDDL description of the move and place actions for M0M. The move action models collision-free motion of the robot while it is not holding anything. The place action models the instantaneous change from when its hand is exerting a force to hold an object to when a force is no longer applied and the object is released. The move-holding and pick actions are described in Figure 15 in Appendix A.

A state is a goal state if the goal formula holds in it. Goal specifications, even those with quantifiers, can be directly and automatically encoded in a PDDL formulation using axioms, logical inference rules [21], [22], [5]. Intuitively, an axiom has the same precondition and effect structural form as an action but is automatically derived at each state. Due to their similarities with actions, axioms can straightforwardly be incorporated in PDDL, enabling a planner to efficiently reason about complex goal conditions, such as the ones present in M0M.

PDDLstream builds on PDDL by introducing stream descriptions, which are similar in syntax to PDDL operator descriptions. An stream is declared by a list of input parameters (:inputs), a logical formula that all legal input parameter values must satisfy (:domain), a list of output parameters (:outputs), and a logical conjunction that all legal input parameter values and generated output parameter values are guaranteed to satisfy (:certified). Each stream is accompanied by a procedure that maps input parameter values to a possibly infinite sequence of output parameter values. Figure 4 displays six streams used in M0M, which we will describe in detail in Section V-D.

C. Predicate semantics

First, we describe the semantics of the predicates used in Figure 3 and Figure 4. The following predicates to encode parameter values type: (Conf ?q) indicates ?q is a continuous robot joint configuration; (Traj ?t) indicates ?t is a continuous robot joint trajectory; (ObjectCloud ?oc) indicates ?oc is an object, which crucially is represented by a segmented point cloud observation; (Pose ?oc ?p) indicates ?p is a pose transform for an object point cloud ?oc relative to its observed frame; (Grasp ?oc ?g) indicates ?g is a grasp transform for an object point cloud ?oc relative to its observed frame. The choice to use the observed frame as the reference frame for an object is arbitrary and has no bearing on the system as poses are only used internally during planning. As a result of this decision, the initial pose ?o of each object cloud is the identity pose. (Property ?pr) denotes that ?pr is a perceivable property, such as a particular color or category.

The following fluent predicates model the current state of the system: (AtConf ?q) represents the current robot configuration ?q; (HandEmpty) is true if the robot’s hand is empty; (AtGrasp ?oc ?g) indicates that object cloud ?oc is held by the robot at grasp ?g; (AtPose ?oc ?p) indicates that object cloud ?oc is resting at location ?p; (On ?oc ?oc2) signifies that object cloud ?oc is resting on object cloud ?oc2. Normally, in a fully observable TAMP setting, ?oc would be the name of an object instance;

Fig. 2. Structure of the M0M policy, which maps RGB-D and robot configuration observations to robot position controls.
The learned streams can use a combination of machine learning and classical estimation techniques. In our system, we consider several implementations of each stream that each can be directly implemented by any off-the-shelf motion planner, such as RRT-Connect [24]. The plan-motion stream plans a continuous trajectory \( t \) between configurations \( q_1 \) and \( q_2 \) that respects joint limits and self collisions, certifying (Motion \( q_1 \), \( t \), \( q_2 \)). It can be directly implemented by any off-the-shelf motion planner, such as RRT-Connect [24].

The learned streams can use a combination of machine learning and classical estimation techniques. In our system, we consider several implementations of each stream that each are a wrapper around a state-of-the-art estimation technique for their subproblem. The predict-grasps stream generates grasps \( g \) for object cloud \( oc \) that are predicted to remain stably in the robot’s hand, certifying (Grasp \( oc \), \( g \)). In Section VII-C, we describe several machine learning implementations of predict-grasps, some of which make predictions directly from \( oc \) without any intermediate representation.

The predict-placements stream generates poses \( p_1 \) for object cloud \( oc \) that are predicted to rest stably on object cloud \( ocw \) when at pose \( p_2 \), certifying (Stable \( oc \), \( p_1 \), \( ocw \), \( p_2 \)). Our implementation of predict-placements decomposes the operation into two estimation subprocedures. First, we perform point cloud completion (Section VII-B) on object cloud \( ocw \) and then estimate approximately horizontal planar surfaces in \( ocw \) when at pose \( p_2 \) using Random Sample Consensus (RANSAC) [25] plane estimation. Next, we perform shape estimation (Section VII-B) on object cloud \( oc \) and then estimate stable orientations relative to a planar surface using the resulting mesh [26]. By combining these two subprocedures, we obtain placements \( p_1 \) for object cloud \( oc \).

The predict-cfree stream predicts whether all robot configurations along trajectory \( t \) do not collide (i.e. are collision-free) with object cloud \( oc \) at pose \( p_2 \), certifying (CFreeTrajPose \( t \), \( oc \), \( p_2 \)). By finely sampling configurations along trajectory \( t \), this test can be reduced to sequence of robot configuration and object cloud collision predictions. Although these predictions could be made directly, we instead use shape estimation (Section VII-B) to estimate the collision volume of both the observable and unobservable object volume as a set of convex bodies. This enables us to use fast convex body collision checkers to answer these queries [27]. A similar predict-traj-grasp stream that predicts collisions with a grasped object is described in Appendix A. Finally, the detect-property stream tests whether object cloud \( oc \) has property \( pr \) and, if so, certifies (Is \( oc \), \( pr \)). Section VII-D describes two property estimators, which detect the category and color of an object from the RGB image observation.

VI. MANIPULATION POLICY

The pseudocode for the manipulation policy, which at its core leverages planning using the model described in Section V, is displayed in Algorithm 1. The M0M solution strategy A flowchart of the policy is illustrated in Figure 1. The policy assumes the set of manipulation actions \( A \) (Section V-B) and the engineered streams \( S_E \) (Section V-D). It requires a implementation of the learned streams \( S_L \). Several options per stream are discussed in Section VII. The policy is conditioned on a particular robot \( \mathcal{R} \) and a specified goal \( \mathcal{G} \). To apply the policy to a new robot \( \mathcal{R} \), it is necessary to provide a URDF description of the robot’s kinematics \( \mathcal{R} \) and a position configuration controller for the robot.

On each decision-making iteration, the robot receives the current RGB-D image \( I \) from its camera and its current joint configuration \( q \) from its joint encoders. From each input RGB-D image, it segments out table point clouds \( T \) and object point clouds \( O \). The segmented object and table
point clouds as well as the robot configuration instantiate the current PDDLstream state \( s \) of the world and robot. This current state along with the goal \( G \), actions \( A \), and streams \( S_E \cup S_L \) form a PDDLstream planning problem, which is solved by \texttt{SOLVE-PDDLSTREAM}, a procedure that denotes a generic PDDLstream planning algorithm. In some cases, such as when a necessary attribute is not detected, \texttt{SOLVE-PDDLSTREAM} will return \texttt{None}, indicating that the goal \( G \) is unreachable from the current state \( s \). Otherwise, \texttt{SOLVE-PDDLSTREAM} will return a plan \( \pi \), which consists of a finite sequence of instances of the actions in \( A \). If the plan is empty (i.e. \( \pi = [] \)), the current state \( s \) was proved to satisfy the goal and the policy terminates successfully. Otherwise, the robot executes the first action \( a_1 = \pi[0] \) using its position controllers and repeats this process by reobserving the scene. Note that this control structure enforces that the robot observes the scene to infer whether it has achieved the goal; otherwise, the robot could erroneously declare success after executing a plan open loop.

![Fig. 4. A PDDL.stream description of the streams, which represent engineered and learned operations. The underlined predicates denote properties estimated by perceptual modules. See Figure 13 in Appendix A for a description of the predict-traj-grasp stream.](image)

Algorithm 1 The M0M policy

Assume: \( A = \{ \text{move, move-holding, pick, place} \} \)
Assume: \( S_E = \{ \text{inverse-kinematics, plan-motion} \} \)
Require: \( S_L = \{ \text{predict-grasps, predict-placements, predict-cfree, ..., detect-attribute} \} \)

1. \texttt{procedure EXECUTE-POLICY}(\( \mathcal{R}, \mathcal{G} \)) Robot \( URDF \) \( \mathcal{R} \), goal \( \mathcal{G} \)
2. \texttt{while True do}
3. \texttt{I, q = OBSERVE()} \( \triangleright \) RGB-D image \( I \), robot conf \( q \)
4. \texttt{T, O = SEGMENT(I)} \( \triangleright \) Tables \( T \), objects \( O \)
5. \texttt{s \leftarrow OBJ-STATE(I, T, O) \cup ROBOT-STATE(R, q)}
6. \texttt{\pi \leftarrow SOLVE-PDDLSTREAM(s, \mathcal{G}, A, S_E \cup S_L)}
7. \texttt{if \pi = None then return False} \( \triangleright \) Failure: unreachable
8. \texttt{if \pi = [] then return True} \( \triangleright \) Success: \( s \in \mathcal{G} \)
9. \texttt{EXECUTE-ACTION(\( \mathcal{R}, \pi[0]\))}

VII. IMPLEMENTATION

We have implemented an instance of M0M, using PDDLstream and experimented with different strategies for implementing the perceptual modules. All make use of RGB-D images gathered from the PR2's Kinect 1 sensor. In this section, we briefly describe implementation of individual modules and include experimental results comparing alternative implementations of several of the modules.

We used standard position trajectory controllers for simulation in PyBullet [28] and on a physical PR2 robot, and simply opened and closed the parallel-jaw grippers to implement grasping and releasing objects. We used the actual opening of the gripper after commanding the gripper to close to detect grasp failure.

A. Segmentation of objects and surfaces

Category-agnostic segmentation is used to identify rigid collections of points that collectively move as an object when manipulated. We compare three different segmentation approaches: UOIS-net-3D, geometric clustering, and a combined method. UOIS-net-3D [29] is a neural-network model that takes RGB-D images as input and returns a segmentation of the scene. It assumes that objects are generally resting on a table; it attempts to segment out image regions corresponding to the table, as well as a set of objects.

For geometric clustering, we first remove the points assigned to the table by UOIS-net-3D, then use density-based spatial clustering of applications with noise (DBSCAN) [30], which finds connected components in a graph constructed by connecting points in the point cloud that are nearest-neighbors in 3D Euclidean distance. In a combined approach, we apply DBSCAN to the segmented point cloud produced by UOIS-net-3D in order to reduce under-segmentation. We additionally use post-processing to filter degenerate clusters.

Figure 5 displays the segmentation mask predicted by UOIS-net-3D while our system was executing the task 5 trial displayed in Figure 14. As can be seen, UOIS-net-3D generally correctly segments the four instances; however, it does oversegment the cracker box into two contacting instances in the last two images.

We compared all three segmentation methods on the ARID-20 dataset of the object clutter indoor dataset (OCID) [31] and GraspNet-1 Billion [32] datasets. Detailed results are reported in the appendix. We found that the different segmentation algorithms have advantages in different settings. In domains where objects have simple geometries and are scattered on the table, an Euclidean-based approach pro-
duces reliable predictions. But in a more cluttered domain, the learned approach often outperforms the Euclidean-based approach. When it comes to challenging situation where objects have more complicated geometry, the performance of the learned approach drops down but it still outperforms the pure Euclidean-based approach. In all experiments, the combined approach performs better than the pure learned approach, indicating the effectiveness of applying DBSCAN and filtering to neural-network-predicted results. In our system experiments, we use the combined method.

B. Shape estimation

A subroutine of our implementation of both the predict-placements and predict-cfree stream operations (Section V-D) is shape estimation, which takes in a partial point cloud as input and predicts a completed volumetric mesh. We again explore a combination of neural-network-based and geometric methods.

The morphing and sampling network (MSN) [33] is a neural-network model that takes as input a partial point cloud and predicts a completed point cloud. Our geometric method works by augmenting the partial point cloud by computing the projection of the visible points onto the table plane. This simple heuristic is motivated by the intuition that the base of an object must be large enough to stably support the visible portion of an object and is particularly useful given a viewpoint that tends to observe objects from above. As a post-processing step for both methods, we filter the result by back-projecting predicted points onto the depth image and pruning any visible points that are closer to the camera than the observed depth value.

1) Mesh interpolation: While it is possible directly use the estimated point cloud in downstream operations, for example by treating the points as spheres or downsampling them as into voxel grid, it is more accurate and efficient to interpolate among the points to produce a volumetric mesh. The simplest way to do this is to take the convex hull of the points; however, this can substantially overestimate the volume when the object is non-convex and fail to find feasible plans when attempting to grasp non-convex objects such as bowls. Instead, we produce the final volume by computing a "concave hull" in the form of an alpha shape [34], a non-convex generalization of a convex hull, from the union of the visible, network-predicted, and projected points. To enable efficient collision checking in the predict-cfree stream, we build an additional representation that approximates the mesh as the union of several convex meshes, implemented by volumetric hierarchical approximate convex decomposition (V-HACD) [35].

Figure 6 visualizes the estimated meshes produced by four of the shape estimation strategies in an uncluttered scene with a diverse set of objects. The first two images compare creating a mesh by taking the convex hull (Figure 6 left) versus a concave hull (Figure 6 middle-left) of the set of visible points (V). The convex hull can significantly overestimate non-convex objects in certain areas, as evidenced by the spray bottle in the top left of the image and the real-world power drill in the right side of the image. The last three images compare three strategies for populating the set of points to be used in a concave hull. Adding the shape-completed points from MSN (VLF) fills in some but not all of the occluded volume of each object, as shown by the cracker box in the middle of the image (Figure 6 middle-right). Also including the projection of the points to the table (VLPF) better fills in the occluded volume at the cost of overestimating the volume when the ground truth base projection is smaller than the visible base projection (Figure 6 right). We evaluated the performance of these methods in four different domains, each on 2000 images taken from a randomly-sampled camera pose; details of the experiments and results can be found in Appendix B. The fully combined method (VLPF) in general performed the best across the domains and is the one we use in the system experiments.

C. Grasp affordances

Grasp affordances are transformations between the robot’s hand and an object’s reference frame such that, if the robot’s hand was at that pose and closed its fingers, it would acquire the object in a stable grasp. They are purely local and do not take reachability, obstacles, or other constraints into account. The modularity of our planning framework enables
us to consider three interchangeable grasping methods for implementing the predict-grasps stream, each take a partial point cloud as input. Grasp Pose Detection (GPD) [36] first generates grasp candidates by aligning one of the robot’s fingers to be parallel to an estimated surface in the partial point cloud and then scores these candidates using a convolutional neural network, which is trained on successful grasps for real objects. GraspNet [37] uses a variational autoencoder (VAE) to learn a latent space of grasps that, when conditioned on a partial point cloud, yields grasps.

We also developed a method, estimated mesh antipodal (EMA), that performs shape estimation using the methods described in Section VII-B and then identifies antipodal contact points on the estimated mesh. Specifically, to generate a new grasp, EMA samples two points on the surface of the estimated mesh that are candidate contact points for the center of the robot’s fingers. The pair of points is rejected if the distance between them exceeds the gripper’s maximum width or if the surface normal at either of the corresponding faces is not approximately parallel to the line between the two points. Then, EMA samples a rotation for the gripper about this line and yields the resulting grasp if the gripper, when open and at this transformation, does not collide with the estimated mesh. A key distinction between EMA’s and GPD’s candidate grasp generation process is that, by using shape estimation, EMA is able to directly reason about the occluded regions of an object instead of just the visible partial point cloud. Additionally, it can take into account unsafe contacts between the robot’s gripper and the object.

Figure 7 illustrates some of the grasps produced by these three approaches in three scenes with varying amounts of clutter, where clutter introduces additional opportunities for occlusion. We performed a real-world experiment to compare the success rates of GPD, GraspNet, and EMA. The details of the experiment and the results can be found in Appendix B. GPD and EMA outperformed GraspNet in our experiments, both in speed and accuracy, with EMA having an edge over GPD. We used EMA in our system experiments.

D. Object properties

In our implementation of the detect-property stream, we considered detectors for two object properties: category and color. We use Faster R-CNN [38] trained on the bowl-cup subset of IIT-AFF [39] to detect bowls and cups so that the robot can identify which objects can contain other objects. Additionally, we use Mask R-CNN [40], trained on both real images in Yale-CMU-Berkeley (YCB) Video Dataset [41] and a synthetic dataset we generated using PyBullet [28], to classify any YCB objects [42] that are mentioned in the goal formula. We also have simple modules that aggregate color statistics directly from segmented RGB images.

VIII. WHOLE-SYSTEM EXPERIMENTS

Finally, we evaluated the whole M0M system by testing its ability to solve challenging real-world manipulation tasks. As an example, Figure 8 illustrates a task where the goal is for a mustard bottle to be on a blue target region:

\[
\exists \text{obj. } \exists \text{region. On(obj, region) } \land \text{Is(region, blue) } \land \text{Is(obj, mustard).}
\]

In its effort to solve this task, the robot moved two obstructing objects out of the way to safely pick the mustard bottle and then place it on the goal region. Additionally, although not pictured, the robot’s first attempt to pick the mustard bottle fails, causing the system to abort execution, re-observe, re-plan, and execute a new grasp that this time was successful. The full video of the trial can be seen at https://youtu.be/tNHjpXP8RFO.

A. Repeated trials

We performed experiments consisting of five repeated real-world trials for five tasks, obtaining the results shown in
Figure 8. M0M acting to satisfy the goal for a mustard bottle to be on a blue target region. Left: for its first action, the robot picks up a water bottle that prevents the robot from safely reaching the mustard bottle. Middle: the robot also picks and relocates the obstructing cracker box. Right: finally, the robot places the mustard bottle on the blue target region.

Figure 9. Here, tasks are loosely defined as a set of problems with the same goal formula and qualitatively similar initial states. Recall that, outside of the PR2’s description, the goal formula is the only input to each trial. We summarize the results here and describe the tasks in subsequent subsections. In addition to these tasks, we applied the system to over 25 individual problem instances. Appendix C highlights several of these tasks along with how the system behaved to solve them. See [https://tinyurl.com/open-world-tamp](https://tinyurl.com/open-world-tamp) for full videos of our system solving these problems.

The column Iterations refers to the average number of combined estimation and planning iterations that were performed per trial. Unless the initial state satisfies the goal conditions, the system always takes two or more iterations because it must at least achieve the goal and then validate that the goal is in fact satisfied. Sometimes the system will perform more than two iterations in the event that the perception module identifies a new object due to undersegmentation, an action is aborted due to a failed grasp, or an action has unanticipated effects.

The columns Estimation, Planning, and Execution report the average time spent perceiving, planning, and executing per iteration. Each module was implemented in Python to flexibly support multiple implementations of each module. Many of the perceptual operations that manipulate raw point clouds could be sped up by using C++ instead of Python and deploying the system using state-of-the-art graphics hardware. During planning, a majority of the time is spent checking for collisions, particularly when the robot is planning free-space motions. The overall runtime could be reduced by simultaneously planning motions for later actions while executing earlier actions [11]. The column Successes reports the number of times out of five trials that the system terminated having identified that it achieved the goal. Our system was able to achieve the goal on every trial except for a single trial that was a part of Task 3. These results show that this single system can perform a diverse set of long-horizon manipulation tasks robustly and reliably.

1) Task 1: This task evaluates our system’s ability to grasp and stably place novel objects that are not well approximated by a simple box. Almost all existing TAMP approaches assume that the manipulable objects can be faithfully modeled using a simple shape primitive for the purpose of manually specifying grasps. The goal in this task is for all objects to be on a blue target region, which corresponds to the following logical formula:

\[
\forall \text{obj.} \exists \text{region. On}(\text{obj, region}) \land \text{Is(region, blue).}
\]

In each trial, a single object is placed arbitrarily on the table. The five objects we used across the five trials were a bowl, a real power drill, a plastic banana, a cup, and a tennis ball. Figure 10 demonstrates a successful trial where the object was a bowl. A video of this trial is available at: [https://youtu.be/PREUU8nVetI](https://youtu.be/PREUU8nVetI).

2) Task 2: This task evaluates our system’s ability to safely place multiple objects in tight regions. The goal in this task is also for all objects to be on a blue target region. The robot must plan a pair of placements and motions for the objects avoid collision. Figure 11 visualizes a successful trial involving a mustard bottle and a toy drill. A video of this trial is available at: [https://youtu.be/BPa_MpxF31M](https://youtu.be/BPa_MpxF31M).

| Task | Iterations | Estimate | Plan | Execute | Success |
|------|------------|----------|------|---------|---------|
| 1    | 2.0        | 29.6s    | 18.5s| 16.1s   | 5/5     |
| 2    | 3.0        | 34.0s    | 37.4s| 23.6s   | 5/5     |
| 3    | 3.0        | 36.8s    | 28.3s| 40.5s   | 4/5     |
| 4    | 2.0        | 39.6s    | 41.6s| 44.6s   | 5/5     |
| 5    | 2.4        | 47.7s    | 18.9s| 28.3s   | 5/5     |
Fig. 11. Task 2: the goal is for all objects to be on a blue target region. First, the robot first picks up the toy drill and places it near the bottom-right corner of the blue target region. Then, the robot picks up the mustard bottle and places it in the remaining collision-free area, which is near the top-left corner of the blue target region.

3) Task 3: This task evaluates our system’s ability to react to unexpected observations. The goal in this task is also for all objects to be on a blue target region. The task was presented in the introduction to the paper, see Figure 1.

Fig. 12. Task 3 failed trial: the goal is for all objects to be on a blue target region. The cracker box twists as the robot picks it up, which knocks the tennis ball outside the reachable workspace of the robot.

4) Task 4: This task evaluates our system’s ability to reason about collisions when placing a target object. The goal in this task is for the object that is closest in hue to red to be on a blue target region, which corresponds to the following logical formula:

$$\exists \text{obj}. \exists \text{region}. \forall \text{obj} \neq \text{obj}.$$  
$$\land \text{On(obj, region)} \land \text{Is(region, blue)}.$$  
$$\land \text{CloserInColor(obj, obj, red)}.$$  

Initially, two objects that are far away in hue from red are located on the blue target region, occupying most of the region, and a third object that is close in hue to red is on the table. Figure 13 shows a successful trial. The robot picks and relocates a sugar bottle and mustard bottle that initially cover the blue target region in order to make room for the toy drill to be safely placed on the region. A video of this trial is available at: https://youtu.be/uqZT5gUBo0.

Fig. 13. Task-4: the goal is for the object that is closest in color to red to be on a blue target region. The robots picks and places both the sugar box and mustard box elsewhere in order to make room to place the toy drill, the most red object, on the goal target region.

5) Task 5: This task evaluates our system’s ability to reason about collisions when attempting to pick objects. The goal is for the object that is closest in color to yellow to be on a blue target region; the goal has a similar form to that in Task 4. Initially, a mustard bottle is placed near the exterior of the table, surrounded by three potentially obstructing objects placed near the table’s interior.

Fig. 14 displays a successful trial. First, the robot picks and relocates the obstructing water bottle. Second, the the robot picks and relocates the obstructing detergent bottle. It falls over during placement; however, the robot is able to infer this during its next observation and its estimates. Finally, the robot picks the mustard bottle and places it on the blue target region. A video of this trial is available at: https://youtu.be/qBD2FyR2ktc.

Fig. 14. Task 5: the goal is for the object that is closest in color to yellow to be on a blue target region. The robot moves both the water bottle and detergent bottle out of the way in order to reach the mustard bottle and then place it on the blue target region.

IX. EXTENSIONS TO MOM

We have presented a simple instance of the MOM framework, which is already quite capable, as illustrated by experimental results in Section VIII. It does have several assumptions and simplifications which can be removed, providing a path to even more general and capable systems.

a) Object features: Object categories can play an important role in supporting the inference of latent object properties. For example, recognizing that an object is likely to be an instance of the coffee mug category, based on its shape and appearance, might allow us to make additional inferences about its material, functional properties (can
contain hot liquid), and parts (its opening and handle). It is straightforward to augment the planning model so that if an object is perceived to have some property or class membership, then additional properties are inferred. This capability enables examples, illustrated in Appendix C, in which by recognizing an object as belong to the category bowl, we infer that objects can be dropped into it.

b) Additional manipulation operations: Extending the system to have additional manipulation operations is somewhat more complex, but the work is substantially amortized, again, over object arrangements, shapes, and goals. Added operations can be smoothly combined with the existing prehensile operation to generate a rich class of plans. For example, to add the ability to push an object, it would be necessary to add:

- a pushing controller to the robot description (although open-loop pushing could be accomplished with an existing position controller);
- a push operator description that models the predicted change in object pose after a push;
- a plan-push sampler, which can generate diverse choices of possible paths along which an object can be pushed, subject to some constraints which may include start and target poses.

Similar characterizations can be given for operations such as pouring and scooping [19], opening a child-proof bottle using impedance control [20], moving kinematic objects such as drawers and doors [11], and many more.

c) State estimation: In the basic system, there is no memory; actions are selected based only on the current view, which like many other vision-based manipulation approaches, is assumed to imperfect but sufficient for acting in the world. For robustness, it is critical to integrate observations over time (e.g., to remember objects that were once visible but are now not) and to integrate the predicted effects of actions (e.g., to increase the belief that an object is located in a bowl after the robot drops it there, even if it cannot be observed inside the bowl from the current angle. In addition, it can be beneficial to be able to fuse information from other sensory modalities, including tactile and auditory sensing.

d) Information gathering: The current perception system generates a single hypothesis about the world state, which is used by the planner to select actions as if it were true. Because it does not take into account the degree to which the robot is uncertain about the world state when it selects actions, it cannot decide that in some situations it would be better to do explicit information-gathering rather than pursue its goal more directly given a point estimate of the state. Previous work [9], [10], [11] has provided methods for TAMP in belief space, but addressed uncertainty only in robot base and object instance pose, but not object shape or other properties. Future work involves integrating these approaches with the proposed approach.

X. Conclusion

We have demonstrated an instance of a strategy for designing and building very general robot manipulation systems using a combination of analytical and empirical methods. The system is a closed-loop policy that maps from images to position commands and generalizes over a broad class of objects, object arrangements, and goals. It is able to solve a larger class of open-world sequential manipulation problems than methods that are either purely analytical (using classic hand-built algorithms for perception, planning, and control) or purely empirical (using modern methods for learning goal-conditioned policies).

Acknowledgements

We gratefully acknowledge support from NSF grant 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from the Honda Research Institute; and from MIT-IBM Watson Lab. Caelan Garrett and Aidan Curtis are supported by NSF GRFP fellowships. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.
A. PDDLStream

This appendix provides the PDDLstream definitions for the remaining MOM operators and streams that are not displayed in Figure 3 and Figure 4. Figure 15 gives the PDDL description of the move-holding and pick actions, which are similar to the move and place actions respectively. Figure 16 gives the PDDLstream description of the predict-cfree-grasp stream, which predicts whether object cloud ?oc2 at pose ?p2 does not collide with object cloud ?oc when held at grasp ?g while the robot follows trajectory ?t. A collision-free prediction certifies (CFreeTrajGraspPose ?t ?oc ?g ?oc2 ?p2).

B. Perception module implementation

This appendix provides details on the implementation and testing of the perception modules.

B.1 Segmentation

Algorithm 2 displays the pseudocode for the combined UOIS-net-3D and DBSCAN segmentation algorithm. It assumes a neural network for table segmentation NNt(·) and a neural network for object segmentation NNs(·), which can be implemented by UOIS-net-3D. The algorithm’s inputs are an RGB-D image o, the robot description R which contains the location of the RGB-D sensor, and the current robot configuration q. It returns a set of segmented table point clouds CT and a set of segmented object point clouds CM.

We compare all three segmentation methods on the ARID-20 subset of the object clutter indoor dataset (OCID) [31] and GraspNet-1Billion [32] datasets, which contain several image distributions that vary object model complexity and the amount of clutter. The free subset (390 images) of the ARID-20 dataset contains cases where objects are relatively well-separated from each other. The touching and stacked subsets (650 images in total) contain cluttered scenes that include objects with simple geometries. The novel object subset (7680 images) of the GraspNet-1Billion [32] dataset contains objects with geometries that are more complicated than simple boxes or cylinders. 960 images are sampled for testing purpose. We used the default parameters for UOIS-net-3D a distance-from-neighbors parameter of D = 0.01 meters for DBSCAN. The ground-truth non-object foreground mask was provided to DBSCAN in order to segment out tables. For a fair comparison, we also provided the ground-truth foreground mask to UOIS-net-3D and the combined method as well. The results are shown in Figure 17. Results are reported using F-measure.

B.2 Surfaces

We use the table segmentation predicted by UOIS-net-3D to identify the set of points to be considered in Random Sample Consensus (RANSAC) [25] plane estimation. Once the plane with the fewest outliers is found, we reclassify points in close proximity to the estimated plane as part of the table to account for segmentation errors.

Algorithm 3 displays the pseudocode for the version of the shape estimation algorithm that first combines the visible, MSN-predicted, and projected points, second filters visible points, and third returns a convex decomposition of the concave hull of the points.

We experimented with the following six shape estimation strategies, all of which ultimately produce a mesh by taking the concave hull of one of following sets of points: a) the visible points only (V), b) the union of the visible points and their vertical projection onto the table plane (VP), c) the points from VP after filtering out points that are inconsistent with the observed depth image (VPF) d) the union of the visible points and the learning based MSN shape-completed (L) point cloud (VL), e) the points from VL after filtering (VLF), and f) the filtered union of visible, projected, and MSN-predicted point clouds (VLFP).

MSN was trained on a dataset comprised of 14 YCB object models rendered by PyBullet. Although we use YCB objects for training, we are aiming to learn a general object-shape prior rather than how to complete these specific objects. Testing is conducted on a much larger variety of objects. Each object model was rendered individually from approximately 200 randomly-sampled view points. During training, the depth image was transformed to 3D space to produce the visible partial point cloud. The model was trained using the the Earth Mover’s Distance (EMD) as the training loss function [33].
\begin{itemize}
  \item \textbf{:action} move-holding
  \item \textbf{:parameters} (?q1 \textsf{?t} ?q2 \textsf{?oc} ?g)
  \item \textbf{:precondition} \{ (\textsf{Motion} \ ?q1 \textsf{?t} \ ?q2) (\textsf{Grasp} \ ?oc \ ?g) (\textsf{AtGrasp} \ ?oc \ ?g) (\textsf{AtConf} \ ?q1) \}
  \item \textbf{:effect} \{ (\textsf{AtConf} \ ?q2) (\textsf{not} (\textsf{AtConf} \ ?q1)) \}
\end{itemize}

\begin{itemize}
  \item \textbf{:action} pick
  \item \textbf{:parameters} (?q ?oc \textsf{?g} ?oc2 \textsf{?p2})
  \item \textbf{:precondition} \{ (\textsf{Grasp} ?oc \ ?g) (\textsf{Kin} \ ?q \ ?g \ ?p) (\textsf{Stable} \ ?oc \ ?p \ ?oc2 \ ?p2) \}
  \item \textbf{:effect} \{ (\textsf{AtConf} \ ?q1) (\textsf{HandEmpty}) (\textsf{AtPose} \ ?oc \ ?p) \}
\end{itemize}

Fig. 15. A PDDL stream description of \textit{move-holding and place} actions. The underlined predicate is estimated by perceptual modules.

\begin{itemize}
  \item \textbf{:stream} predict-conf-free-grasp
  \item \textbf{:inputs} (?t \textsf{?oc} \textsf{?g} \textsf{?p2} \textsf{?p2})
  \item \textbf{:domain} \{ (\textsf{Traj} \ ?t) (\textsf{Grasp} \ ?oc \ ?g) (\textsf{Pose} \ ?oc2 \ ?p2) \}
  \item \textbf{:certified} \{ (\textsf{CFreeTrajPose} \ ?t \ ?oc2 \ ?p2) \}
\end{itemize}

Fig. 16. A PDDL stream description of the \textit{predict-traj-grasp} stream. The underlined predicates denote properties estimated by perceptual modules.

\begin{tabular}{|c|c|c|c|c|c|c|c|}
\hline
Dataset & DBSCAN & UOIS-net-3D & COMBINED \\
\hline
OCID Uncluttered & 98.7 & 95.5 & 97.2 \\
OCID Cluttered & 68.7 & 87.5 & 89.6 \\
GraspNet-1Billion & 69.1 & 80.1 & 82.6 \\
\hline
\end{tabular}

Fig. 17. Comparison of the segmentation approaches in terms of F-measure.

\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
Clutter & Complex & V & VP & VPF & VL & VLF & VLF_P \\
\hline
No & No & 0.489 & 0.639 & 0.678 & 0.633 & 0.649 & 0.703 \\
Yes & No & 0.477 & 0.536 & 0.589 & 0.565 & 0.573 & 0.612 \\
No & Yes & 0.497 & 0.568 & 0.604 & 0.530 & 0.615 & 0.620 \\
Yes & Yes & 0.398 & 0.464 & 0.483 & 0.446 & 0.503 & 0.497 \\
\hline
\end{tabular}

Fig. 18. Comparison of the shape-estimation strategies.

We evaluated the performance of these methods in four different domains, each on 2000 images taken from a randomly-sampled camera pose. Each domain either contains imagines with a single object (No Clutter) or with 10-14 objects (Clutter). The objects in each domain are either all primitive shapes (boxes, cylinders, and spheres) with randomized dimensions (Not Complex) or a subset of 922 objects in 8 categories from the ShapeNet [43] dataset, which include cameras, bowls, mugs, headphones, etc (Complex). To initialize the scene, objects are dropped sequentially on the table from a randomly-generated pose and simulated until arriving at a resting state. For scoring, the estimated meshes were converted to surface voxels and evaluated using boundary volumetric Intersection-over-Union (IoU). IoU scores close to one indicate that the estimated mesh covered most of the ground truth boundary voxels. Boundaries were dilated to tolerate a small amount of misalignment that arises from the voxelization process.

Figure 18 shows the results of the experiment. Using only the visible points (V) performed the worst in each domain. The fully combined method (VLPF) in general performed the best across the domains, and filtering (F) improved the accuracy of the predictions produced by both L and P.

Our experiments were conducted in scenes where objects are resting stably on a flat surface and the camera has an above viewing angle, which likely contributes to the projection approach (VP) outperforming the learned approach (VL) before filtering. The learned approach is a more general and flexible prior compared to the projection method, which is a fixed heuristics. However the input (observed point cloud) will have small drift after going through the network, making the output inconsistent with the observed depth image. The filtering process can fix this problem so VLPF outperforms VP in complex domains. In scenes where the top of an object could not be observed or the objects have complex geometries, we expect the learned approach to be more competitive.

\begin{algorithm}
\caption{The combined shape estimation algorithm}
\begin{algorithmic}[1]
\Require Shape completion neural network $NN_{sc}(\cdot)$
\Require Minimum number of points $m \geq 3$
\Require Alpha shape parameter $\alpha$
\Procedure{estimate-mesh}{$c, T, o, R, q$}
\State $P_o \leftarrow \text{camera-pose}(R, q)$
\For{($p_t, h_t$) $\in T$}
\State $c_t \leftarrow \text{TFORM}(p_t^{-1}, c)$
\State $b_t \leftarrow \{[v_x, v_y, 0] \mid v \in c_t, [v_x, v_y] \in h_t, v_z \geq 0\}$
\State $c'_t \leftarrow c_t \cup b_t \cup NN_{sc}(c_t)$\Comment Completed point cloud $c'_t$
\State $c''_t \leftarrow \{v \in \text{TFORM}(p_t^{-1}, c'_t) \mid v_z \geq o[v_x, v_y]\}$
\EndFor
\If{$|c''_t| < n$}
\State continue
\EndIf
\State $M_o \leftarrow \text{CONCAVE-HULL}(c''_t)$\Comment concave mesh $M_o$
\State $M_o \leftarrow \text{CONVEX-DECOMPOSITION}(M_o)$
\State $M \leftarrow \text{TFORM}(P_o, M_o)$\Comment Convex meshes $M$
\State return $M$
\EndProcedure
\end{algorithmic}
\end{algorithm}

\subsection{B.4 Grasps}

Algorithm 5 displays the pseudocode for the EMA grasp generator. It takes as input the robot description $R$ and an estimated mesh $m$. It yields a possibly infinite sequence of grasp poses. We performed a real-world experiment to compare the success rates of GPD, GraspNet, and EMA.
Algorithm 5 The EMA grasp generation algorithm

Require: Antipodal normal tolerance $\delta \approx 0$ in radians

1: procedure GENERATE-GRASPS($R, m$)
2: \[E \leftarrow \text{END-EFFECTOR}(R)\] \hspace{1em} ▷ Extract the end effector
3: while True do
4: \[v_1, \hat{n}_1 \leftarrow \text{SAMPLE-SURFACE}(m)\] \hspace{1em} ▷ Point $v_1$, normal $\hat{n}_1$
5: \[v_2, \hat{n}_2 \leftarrow \text{SAMPLE-SURFACE}(m)\]
6: \[\hat{d}_{21} \leftarrow \frac{(v_2 - v_1)}{|v_2 - v_1|}\]
7: if $(-\hat{n}_1 \cdot \hat{d}_{21}) < \cos \delta$ or $(\hat{n}_2 \cdot \hat{d}_{21}) < \cos \delta$ then continue
8: \[\theta \leftarrow \text{SAMPLE-UNIFORM}(0, 2\pi)\]
9: \[R \leftarrow \text{AXISANGLE}(\hat{d}_{21}, \theta)\]
10: \[p \leftarrow \text{POSE}(R, (v_1 + v_2)/2)\] \hspace{1em} ▷ Grasp pose $p$
11: if not \text{COLLISION}(E, p, m, I_4) then
12: \[\text{yield } p\] \hspace{1em} ▷ Generate the grasp

We used a simple definition of success where a grasp is successful if, after the robot performs a pick action with the grasp and then lifts its gripper, the robot is unable to fully close its fingers, which indicates that an object is in between the fingers. Measuring the degree to which the object has changed pose relative to the robot would provide a more nuanced evaluation, but is beyond our current scope. Each grasping trial was formulated as a call to the overall system with goal $\exists \text{obj}. \text{Holding(left-hand, obj)}$. In order to ensure that the planner does not always try to grasp the same object, we added a randomized cost to each individual pick action that biases the planner to select the object with lowest-cost feasible pick. We performed two experiments, one in an uncluttered setting where objects are well-separated and one in a cluttered setting where objects are in contact with each other. We performed 35 grasping trials in each setting.

Figure 19. Left: a grasp generated by EMA in the uncluttered grasp experiment. Right: a grasp generated by EMA in the cluttered grasp experiment.

Figure 20 shows the results of the grasp experiments, which are illustrated in Figure 19. In the uncluttered case, the GPD and EMA methods both outperformed GraspNet and were able to produce a successful grasp on 27 out of 35 attempts. A commonality between GPD and EMA is that they both identify candidate grasp transforms using geometric procedures, which operate either on a point cloud (GPD) or on a mesh (EMA), whereas GraspNet samples candidates directly from a neural network, which might not retain geometric invariants. GraspNet was also more computationally expensive than GPD and EMA, likely due to the inference overhead when using PointNet [44].

| Method   | Uncluttered Successes | Time | Cluttered Successes | Time |
|----------|-----------------------|------|---------------------|------|
| GPD      | 27/35                 | 8.7s | 23/35               | 6.8s |
| GraspNet | 21/35                 | 48.3s| 16/35               | 21.7s|
| EMA (ours) | 27/35             | 10.3s| 27/35               | 7.3s |

Fig. 20. Grasp success rates in uncluttered and cluttered settings.
C. Demonstrations

In addition to the multi-trial full-system experiments, we also tested our system with the following individual problems to qualitatively demonstrate the system’s behavior; this appendix provides brief descriptions of these tasks. See [https://tinyurl.com/open-world-tamp](https://tinyurl.com/open-world-tamp) for full videos of M0M solving these problems.

1) Packing three objects: The goal in this task is for all objects to be on a red target region, which corresponds to the following logical formula:

\[ \forall \text{obj. } \exists \text{region. On(obj, region)} \land \text{Is(region, red)}. \]

Figure 21 displays still images from the video of our system solving this task, which is available at [https://youtu.be/qGbTOBECUXA](https://youtu.be/qGbTOBECUXA).

![Figure 21](image1)

2) Changing the resting face: The goal in this task is for all objects to be on a red target region, which corresponds to the following logical formula:

\[ \forall \text{obj. } \exists \text{region. On(obj, region)} \land \text{Is(region, red)}. \]

Figure 22 displays still images from the video of our system solving this task, which is available at [https://youtu.be/Er_m0CSsX7E](https://youtu.be/Er_m0CSsX7E).

![Figure 22](image2)

3) Dispersing three objects: The goal in this task is for all objects to be on a target region, which corresponds to the following logical formula:

\[ \forall \text{obj. } \exists \text{region. On(obj, region)}. \]

In contrast to other tasks, both a red and a blue target region are on the table. Figure 23 displays still images from the video of our system solving this task, which is available at [https://youtu.be/dWU8S8TuD5Y](https://youtu.be/dWU8S8TuD5Y).

![Figure 23](image3)

4) Manipulation in clutter: The goal in this task is for a mustard object to be on a target region, which corresponds to the following logical formula:

\[ \exists \text{obj. } \exists \text{region. On(obj, region)} \land \text{Is(obj, mustard)}. \]

Figure 24 displays still images from the video of our system solving this task, which is available at [https://youtu.be/UG8V0E1064A](https://youtu.be/UG8V0E1064A).

![Figure 24](image4)

5) Occluded objects: The goal in this task is for all objects to be on a blue target region, which corresponds to the following logical formula:

\[ \forall \text{obj. } \exists \text{region. On(obj, region)} \land \text{Is(region, blue)}. \]
Figure 25. *Occluded Objects*: the goal is for all objects to be on a blue target region.

6) **Obstructed place**: The goal in this task is for a mustard bottle to be on a blue target region, which corresponds to the following logical formula:

$$\exists \text{obj. } \exists \text{region. } \text{On} (\text{obj}, \text{region}) \land \text{Is} (\text{obj}, \text{mustard})$$

Figure 26 displays still images from the video of our system solving this task, which is available at [https://youtu.be/ipW0GA3M0yw](https://youtu.be/ipW0GA3M0yw).

Fig. 26. *Obstructed Place*: the goal is for a mustard bottle to be on a blue target region. The robot move the sugar box off of the blue target region to make room for the mustard bottle.

7) **Moving a destructible object**: The goal in this task is for all objects to be on a blue target region, which corresponds to the following logical formula:

$$\forall \text{obj. } \exists \text{region. } \text{On} (\text{obj}, \text{region}) \land \text{Is} (\text{region}, \text{blue})$$

Figure 27 displays still images from the video of our system solving this task, which is available at [https://youtu.be/ZpkzoQmVJ9S](https://youtu.be/ZpkzoQmVJ9S).

Fig. 27. *Moving a Destructible Object*: the goal is for all objects to be on a blue target region. The Lego object is made up of a set of weakly-attached bricks and thus can break when manipulated. When the robot attempts to pick up the Lego, it breaks into three pieces, effectively creating two new objects. Still, the robot is able to perceive these new objects and re-plan to ensure that each one of them ends up on the blue target region.

7) **Moving a destructible object**: The goal in this task is for all objects to be on a blue target region. The robot move the sugar box off of the blue target region to make room for the mustard bottle.

8) **Sorting**: The goal in this task is for each object to be in the bowl that is closest in color to it, which corresponds to the following logical formula:

$$\forall \text{obj. } \exists \text{bowl. } \exists \text{color. } \forall \text{bowl2} \neq \text{bowl}. \text{In} (\text{obj}, \text{bowl}) \land \text{Is} (\text{obj}, \text{color}) \land \text{CloserInColor} (\text{bowl}, \text{bowl2}, \text{color})$$

Figure 28 displays still images from the video of our system solving this task, which is available at [https://youtu.be/Z6VL7jbsnU](https://youtu.be/Z6VL7jbsnU).

Fig. 28. *Re-grasp*: the goal is for an object to be in the robot’s left hand. The spam can is initially on the far right side of the table. Because its left arm is not able to initially reach it, the robot plans to pick up the spam with its right arm and place it in reach of the left arm.
Figure 29 displays still images from the video of our system solving this task, which is available at [https://youtu.be/rjIvd9tRpg](https://youtu.be/rjIvd9tRpg).

Fig. 29. *Sorting*: the goal is for each object to be in the bowl that is closest in color to them. The robot is able to successfully solve a long-horizon task that requires picking closely-placed objects.