Placing Objects with prior In-Hand Manipulation using Dexterous Manipulation Graphs

Joshua A. Haustein[1], Silvia Cruciani[1], Rizwan Asif[1], Kaiyu Hang[2] and Danica Kragic[1]

***Abstract*— We address the problem of planning the placement of a grasped object with a robot manipulator. More specifically, the robot is tasked to place the grasped object such that a placement preference function is maximized. For this, we present an approach that uses in-hand manipulation to adjust the robot's initial grasp to extend the set of reachable placements. Given an initial grasp, the algorithm computes a set of grasps that can be reached by pushing and rotating the object in-hand. With this set of reachable grasps, it then searches for a stable placement that maximizes the preference function. If successful it returns a sequence of in-hand pushes to adjust the initial grasp to a more advantageous grasp together with a transport motion that carries the object to the placement. We evaluate our algorithm's performance on various placing scenarios, and observe its effectiveness also in challenging scenes containing many obstacles. Our experiments demonstrate that re-grasping with in-hand manipulation increases the quality of placements the robot can reach. In particular, it enables the algorithm to find solutions in situations where safe placing with the initial grasp wouldn't be possible.**

I. Introduction

When a robot is tasked to place an object, the target placement may either be specified explicitly in form of a specific pose, or implicitly in form of a target region. In the latter case, the pose for placing the object must be selected by the robot. Consider Fig. 1, where a robot is tasked to place a grasped object inside a cabinet. To solve this task, the robot needs to overcome multiple challenges:

1) Locate object poses within the shelf that afford a stable placement.
2) Among these stable poses, select the pose that is most suitable. In many tasks there exists a placement preference, such as a preference for certain object orientations or clearance from obstacles.
3) Ensure that the selected placement can be reached among clutter. This process involves ensuring that a collision-free approach motion exists and can be computed.

The grasp that the robot acquires on the object is decisive for the robot to succeed at this task. For example, the grasp shown in Fig. 1 on the left allows the robot to place the depicted object in an upright position, whereas the one on the right does not.

Prior works on pick-and-place [1]–[3] have addressed this issue by integrating the process of grasp selection with the process of placement selection. This integration, however, places significant constraints on the grasps that can be used. This is particularly problematic in scenarios where the initial grasp can not be chosen freely, e.g. due to obstruction by obstacles, or when the object is handed to the robot.

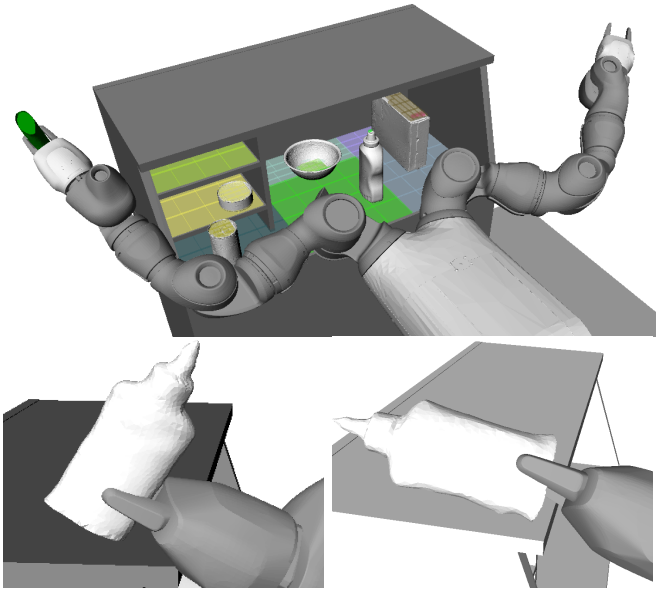


Fig. 1: An example of a placement problem. The robot has to place an object inside the cabinet. The initial grasp on the left allows the robot to place the object in an upright position. The initial grasp on the right does not allow such a placement, as the gripper would collide with the environment.

Enabling the robot to change its grasp can simplify the placement problem, and render infeasible queries feasible. Classical re-grasping [4]–[6] addresses this by placing the object at an intermediate placement, and then picking it up again with a different grasp. This, however, requires locating an intermediate placement first, which is the same problem the robot is trying to solve in the first place. In-hand manipulation, in contrast, allows to change the grasp without placing the object.

In this paper, we integrate Dexterous Manipulation Graphs (DMG) [7], an in-hand manipulation framework, with our recent work on placement planning [8]. The DMG is a tool for in-hand manipulation that enables us to plan sliding paths for a parallel-jaw gripper along an object's surface. Combined with our placement planner, this provides us with the ability to plan placements under the consideration of different grasps that can be reached through in-hand manipulation. Given an initial grasp on an object, and a user-specified preference for placements, our algorithm computes

[1] Division of Robotics, Perception and Learning (RPL), EECS, KTH Royal Institute of Technology, Stockholm, Sweden, E-mail: haustein, cruciani, rasif, dani@kth.se

[2] Department of Mechanical Engineering and Material Science, Yale University, New Haven, USA, E-mail: kaiyu.hang@yale.edu

a sequence of actions that adjust the grasp in-hand, and then transport the object to a placement that maximizes the preference function.

II. Related Works

We first provide an overview of related placement planning works. Thereafter, we discuss various alternatives for planning in-hand manipulation to change a robot grasp without prior placement.

A. Placement Planning

Placement planning focuses on the problem of locating and selecting a placement pose for a given object. To locate such a pose, prior works have proposed both analytic geometry-based methods [8], [9], as well as data-driven methods [10], [11]. Analytic methods locate stable placement poses by matching shape features of the object to shape features of the environment. For example, a common approach is to locate horizontal support surfaces in the environment and to compute the various orientations, in which the object can rest on these [6], [8], [9], [12]. Existing data-driven approaches train classifiers to either discriminate support surfaces from clutter [10], or to evaluate the placement suitability for sampled object poses [11].

A particular challenge in placing are obstacles, as the robot needs to move close to a support surface and other obstacles to reach a placement. Therefore, to guarantee that a placement pose is reachable, previous works integrate the search for a placement pose with sampling-based motion planning [8], [9]. An alternative to transporting an object all the way to its target pose is to drop it in a controlled manner. Holladay et al. [13] formulate the problem of placing as inverse motion planning problem. The approach exploits geometrical features of the environment and the robot to constrain the possible motions a dropped object can follow after it was released. This allows placing at poses that would be infeasible to reach otherwise. If the grasped object is fragile, however, or placement accuracy is of relevance, placing the object directly at the target pose is preferable.

When placing objects at a specific pose, there often exists a preference for certain placements over others. In our previous work [8], we proposed an algorithm to search for placements that maximize a placement objective function. The work focused on locating such placements in the presence of many obstacles. The approach combines motion planning with a Monte-Carlo Tree search based sampling algorithm for placements. The proposed sampling algorithm is adaptive and addresses the presence of obstacles by dynamically focusing its sampling on placements that are likely reachable. The placements the algorithm computes, however, only consider a single initial grasp. Hence, this grasp needs to be carefully selected in advance to ensure that the algorithm can succeed. In this work, we extend this algorithm to consider different grasps by integrating it with an in-hand manipulation framework.

B. In-Hand Manipulation Planning

In-hand manipulation allows a robot to change its grasp without releasing the object. This is in stark contrast to re-grasping by placing [4]–[6], [14], or hand-over grasping with a second manipulator [14]. It is particularly useful in the context of placing, as it does not require locating intermediate placements, nor does it necessarily require a second manipulator.

Many works on in-hand manipulation rely on having access to high fidelity dynamic models of the object. An example is the RRT* based planner in [15] that plans a sequence of pushes of the object against external fixtures to reach a desired grasp configuration. All the friction coefficients between the object, the gripper and the external fixtures are assumed to be known, so that a dynamic solver can properly backtrack the effect of each action. Similarly, various strategies for smaller changes in the grasp configuration (e.g. pivoting, sliding) have been proposed that also rely on complete knowledge of the system's dynamic properties [16]–[18].

In contrast, an example of a planner that relies only on kinematic models has been proposed in [19]. Using trajectory optimization, this method provides a solution to change the grasp configuration of a multi-finger hand. However, the contacts between the object and the fingertips do not slide, making this planner strongly rely on the multiple degrees-of-freedom of the hand. In addition, it is not suitable for large changes in the desired grasp.

In our previous work, we developed a planner for in-hand manipulation based on analysis of only the object's shape: the Dexterous Manipulation Graph (DMG) [7]. This planner provides solutions to move the object inside the gripper as a sequence of pushes against the environment or exerted by a second gripper in a dual-arm robot scenario. Among the possible alternatives for in-hand manipulation planning, we will exploit the DMG to easily determine what grasps are reachable from the initial grasp through in-hand manipulation.

III. Problem Statement

We consider a placement problem as depicted in Fig. 1. A robot grasping a *rigid* object o with an initial grasp g_0, is tasked to place the object on a piece of furniture in front of it. For this, the user specifies a target volume $V \subset \mathbb{R}^3$ describing the permitted object positions, and a placement preference function $\xi : \mathcal{X}^o \rightarrow \mathbb{R}$ that is to be maximized, where $\mathcal{X}^o = V \times SO(3) \subset SE(3)$ is the set of poses in V.

We consider a robot with a parallel-jaw gripper. Accordingly, a grasp $g = (a, {}^eT_o)$ is a tuple of an aperture $a \in [0, \hat{a}]$, and the pose of the object w.r.t. the gripper, ${}^eT_o \in SE(3)$. Let $\mathcal{G} \subsetneq [0, \hat{a}] \times SE(3)$ denote the set of all such grasps on the object o. The initial grasp $g_0 \in \mathcal{G}$ may be poorly suited for placing the object, and therefore we equip the robot with the ability to change its grasp through in-hand manipulation. For this, let $\mathcal{G}(g_0) \subseteq \mathcal{G}$ denote the set of grasps that can be reached through in-hand manipulation from g_0 without

releasing the object. Then, formally, we aim to solve the following constrained optimization problem:

$$\begin{aligned} \underset{\boldsymbol{x}\in\mathcal{X}^o, g\in\mathcal{G}(g_0)}{\text{maximize}} \quad & \xi(\boldsymbol{x}) \\ \text{subject to} \quad & c_f(\boldsymbol{x}) \quad = 1 \\ & s(\boldsymbol{x}) \quad = 1 \\ & r(\boldsymbol{x}, g) \quad = 1, \end{aligned} \tag{1}$$

where the binary predicate $c_f(\boldsymbol{x}) = 1$ states that $\boldsymbol{x}$ is physically feasible, i.e. o does not penetrate any obstacle, $s(\boldsymbol{x}) = 1$ states that o rests stably, and $r(\boldsymbol{x}, g) = 1$ that $\boldsymbol{x}$ is reachable by the robot with grasp g.

The reachability constraint is determined by the robot's kinematics. Let $\mathcal{C}$ denote the configuration space of the robot, and let $O_g(q)\colon \mathcal{C} \rightarrow SE(3)$ denote the pose of the object at configuration $q \in \mathcal{C}$ under grasp g. Depending on the grasp g, the configuration space is partitioned differently in collision and collision-free space, $\mathcal{C} = \mathcal{C}^g_{\text{free}} \cup \mathcal{C}^g_{\text{obst}}$. A configuration $q \in \mathcal{C}$ is colliding, if the robot is colliding or the object at pose $\boldsymbol{x} = O_g(q)$ is colliding.

The reachability constraint, $r(\boldsymbol{x}, g)$, consists of two sub-constraints. First, the pose $\boldsymbol{x}$ must be kinematically reachable, i.e. there must exists a grasp $g \in \mathcal{G}(g_0)$ and a collision-free configuration $q \in \mathcal{C}^g_{\text{free}}$ such that $O_g(q) = \boldsymbol{x}$. Second, given the initial configuration of the robot q_0, the pose must be *path-reachable*, i.e. there must exists a grasp $g \in \mathcal{G}(g_0)$, for which a collision-free continuous path $\tau\colon [0, 1] \rightarrow \mathcal{C}^g_{\text{free}}$ starting from the initial configuration $\tau(0) = q_0 \in \mathcal{C}^g_{\text{free}}$ and ending in a configuration $\tau(1) \in \mathcal{C}^g_{\text{free}}$ with $O_g(\tau(1)) = \boldsymbol{x}$ exists.

The aim of our algorithm is to solve the problem in Eq. (1), and provide a grasp $g^* \in \mathcal{G}(g_0)$, as well as a path τ^* that reach the solution $\boldsymbol{x}^*$ of Eq. (1). In addition, the algorithm should provide a sequence of in-hand pushes that transfer the grasp from g_0 to g^*. Solving Eq. (1) exactly, however, is very challenging, as we cannot express its feasible set in closed form. Instead, we will present a sampling-based algorithm that incrementally produces reachable placements that subsequently achieve higher objectives.

A. Practical Considerations and Limitations

To perform the in-hand manipulation, we use a second manipulator to push the grasped object within the grasping gripper while the grasping arm remains in configuration q_0. For this, we assume there is an obstacle-free region in front of the furniture, where this manipulation can be performed safely, without the need to avoid obstacles.

We also assume access to the kinematic and geometric model of the robot, the geometry of the object, the location of its center of mass, and the geometry of the environment in form of surface points $\mathcal{S} \subsetneq \mathbb{R}^3$. Additionally, we assume that the environment is rigid, and define the world's reference frame such that gravity acts antiparallel to the z-axis.

For a pose $\boldsymbol{x} \in SE(3)$ let $\boldsymbol{p_x} = (x, y, z) \in \mathbb{R}^3$ denote its position and $\boldsymbol{o_x} = (e_x, e_y, e_z)$ its orientation w.r.t the world reference frame axes. To optimize the objective function ξ, we assume that the function is differentiable w.r.t. the x, y, e_z components of a pose and compute these gradients numerically.

IV. Method

An overview of our algorithm is shown in Fig. 2. The extensions to our previous work [8] are highlighted in green. The algorithm receives the information listed on the left as input and produces manipulator paths $\tau_i : [0, 1] \rightarrow \mathcal{C}^{g_i}_{\text{free}}$ to stable placements as output. Each path τ_i is associated with a grasp $g_i \in \mathcal{G}(g_0)$ on the object. The algorithm operates in an anytime fashion by iteratively producing new paths τ_i as runtime progresses. Each subsequent path leads to a placement pose with higher objective value than the previous path. All paths start at the same initial arm configuration q_0, at which the in-hand manipulation will be performed prior to following the path.

The algorithm consists of two stages: the pre-processing stage and the optimization stage. In the pre-processing stage the various input geometries are analyzed. From the environment, the algorithm extracts contiguous horizontal surfaces in the target volume on which the object may be placed. We refer to these surfaces as *placement regions*. From the object, the algorithm computes the DMG and the grasps $\mathcal{G}(g_0)$ that can be reached using it. In addition, it computes the different orientations of the object at which it can be placed horizontally. For this, the algorithm extracts different faces of the object's convex hull, that we refer to as *placement faces*.

In the optimization stage, the algorithm uses these quantities to first locate stable, physically feasible and kinematically reachable placement poses. It then provides arm configurations reaching these poses as goals to a motion planning algorithm to verify path-reachability. Subsequently, path-reachable placements are locally optimized using a greedy gradient-descent algorithm. This procedure is repeated until the user requests termination. Whenever a path-reachable placement has been found, the corresponding approach path is returned and the subsequent iterations are constrained to only search for placements achieving a higher objective.

A. Pre-processing

1) Placement Regions and Faces: The computation of placement regions and placement faces is described in detail in our prior work [8]. Briefly, a placement face is a face of the object's convex hull that supports a stable horizontal placement. A face supports such a placement, if the projection of the object's center of mass along the face's normal falls into its interior. Each placement face gives rise to a base orientation of the object, at which it can be placed horizontally. If aligned with the horizontal surface of a placement region, different placements can be achieved by translating the object parallel to the surface, or rotating it around the surface's normal. For a placement face f and placement region r, let $\hat{S}(r, f) \subset \mathcal{X}^o$ denote the set of all object poses that can be achieved this way.

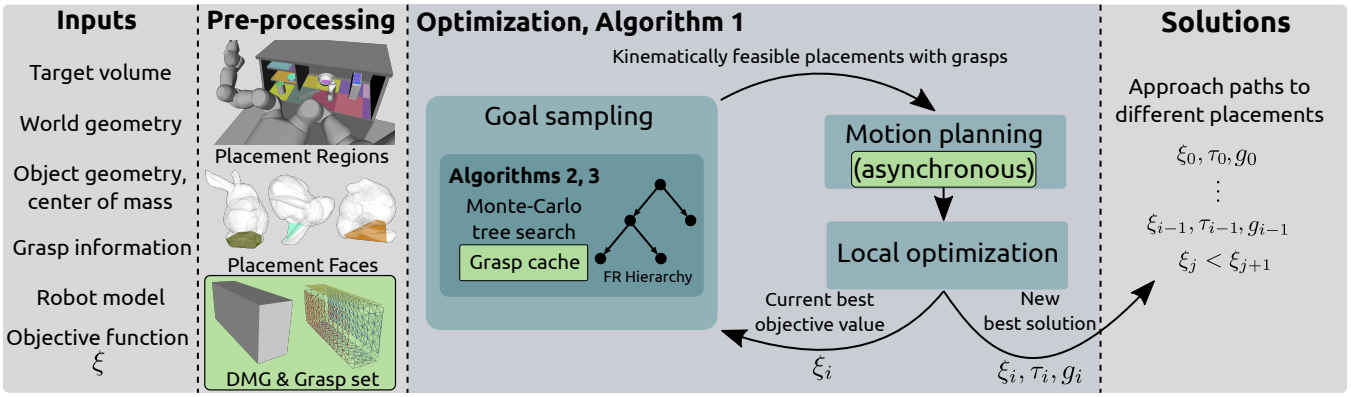


Fig. 2: Our approach consists of two stages. In the pre-processing stage it computes placement regions, faces and the DMG of the object. From the DMG it then computes the set of reachable grasps $\mathcal{G}(g_o)$. In the optimization stage an adaptive sampling algorithm samples kinematically reachable and collision-free stable placement poses under the consideration of different grasps. These are provided to an asynchronous motion algorithm to verify path-reachability and construct an approach motion. Subsequently, a local optimization improves the reached placements locally. Every new solution is made available to the user, and subsequent iterations return better solutions. The additions for re-grasping to our prior [8] work are highlighted in green.

All placement faces and regions together define a parameterized search space for valid placement poses:

$$\hat{S} = \bigcup_{r \in \mathcal{R}, f \in \mathcal{F}} \hat{S}(r, f) \subset \mathcal{X}^o, \tag{2}$$

where $\mathcal{R}$ is the set of all placement regions and $\mathcal{F}$ the set of all placement faces. The set $\hat{S}$ is a lower dimensional subset of $\mathcal{X}^o$ that is likely to contain stable poses, and serves as our search space. Not all poses within $\hat{S}$, however, are stable, i.e. poses located at the edge of a region. In addition, many of the poses may be physically infeasible or not reachable.

2) Dexterous Manipulation Graph: To enable the robot to change its grasp using in-hand manipulation, we compute the DMG of the object. The DMG is a disconnected undirected graph that represents how one finger of a parallel-jaw gripper can move along the object's surface. The DMG is obtained by analyzing the object's shape, and the process of deriving it is detailed in our previous work [7].

A DMG node is a tuple $n{=}\langle \mathbf{p}_n, \Phi_n \rangle$, where $\mathbf{p}_n \in \mathbb{R}^3$ is the position of the contact point between the fingertip and the object, and Φ_n is a continuous set of orientations that the finger can assume when in contact at $\mathbf{p}_n$. Two DMG nodes n and m are adjacent in the graph, if it is possible for the fingertip to translate between $\mathbf{p}_n$ and $\mathbf{p}_m$, and $\Phi_n \cap \Phi_m \neq \emptyset$. The possible translations and rotations of the finger along the object depend on the object's shape; for instance, a finger cannot slide across sharp edges. Therefore, the DMG is a disconnected graph.

3) Reachable Grasps: We use the DMG to compute in advance the grasps $\mathcal{G}(g_0)$ that the robot can reach from its initial grasp g_0 using in-hand manipulation. A grasp does not directly correspond to a single node in the DMG. A DMG node represents a set of possible orientations that a single finger can assume at the node's contact position. Therefore, each grasp is associated with two nodes as well as the actual orientation the fingers assume for the grasp — one node and orientation per finger.

With this in mind, the DMG implicitly defines a graph of grasps. In this graph of grasps each node represents a grasp, and the adjacency relation describes the robot's ability to locally slide from one grasp to another through a rotation or a translation of the object. Each grasp node is a tuple $\langle n_1, \phi_1, n_2, \phi_2 \rangle$, where n_i is the DMG node and $\phi_i \in \Phi_i$ the orientation of the respective finger.

The adjacency relation is obtained from the DMG according to the two following rules. Two grasp nodes $a = \langle n_1^a, \phi_1^a, n_2^a, \phi_2^a \rangle, b = \langle n_1^b, \phi_1^b, n_2^b, \phi_2^b \rangle$ are translationally adjacent, if

- finger 1 can slide: n_1^a is adjacent to n_1^b
- finger 2 can slide: n_2^a is adjacent to n_2^b
- finger 1's orientation is valid: $\phi_1^a \in \Phi_{n_1^b}$ and $\phi_1^a = \phi_1^b$
- finger 2's orientation is valid: $\phi_2^a \in \Phi_{n_2^b}$ and $\phi_2^a = \phi_2^b$.

This adjacency means the robot can translate the object in-hand between the two grasps. Additionally, two grasp nodes $a = \langle n_1, \phi_1^a, n_2, \phi_2^a \rangle, b = \langle n_1, \phi_1^b, n_2, \phi_2^b \rangle$ that share the same contact point and only differ in orientation are rotationally adjacent. This adjacency means the robot can rotate the object around the grasp axis to a grasp with different orientation.

Similar to the DMG, the grasp graph is a disconnected graph. Accordingly, the set of reachable grasps $\mathcal{G}(g_o)$ is only a subset of all grasps on the object. We employ Dijkstra's algorithm to explore the connected component originating from g_0. As a result, we obtain the reachable grasp set $\mathcal{G}(g_o)$, and the shortest sequence of in-hand object translations and rotations to move from g_0 to any $g \in \mathcal{G}(g_0)$. Note that the grasp set $\mathcal{G}(g_0)$ is a discrete set of grasps, whose cardinality is determined by the resolution of the DMG.

B. Optimization Stage

The optimization algorithm is shown in Algorithm 1. As aforementioned, the algorithm is sampling-based as we can not model the set of valid placement poses in closed form. Instead, we confirm the validity of sampled placement poses through collision-checking, inverse kinematics computation and motion planning.

The algorithm alternates between sampling collision-free arm configurations that reach candidate placement poses,

Algorithm 1: Placement Optimization

1 $P_s \leftarrow \emptyset$ // Goal sampler state
2 $\tau_{\text{best}}, g_{\text{best}}, \xi_{\text{best}}, \mathcal{Q} \leftarrow \bot, \bot, -\infty, \emptyset$
// Initialize asynchronous motion planning
3 $M \leftarrow$ InitMotionPlanner(q_0)
4 **while not** Terminate()
// Goals are tuples (q, g), $q \in \mathcal{C}^g_{\text{free}}$, $g \in \mathcal{G}(g_0)$
5 $\mathcal{Q}_n, P_s \leftarrow$ SampleGoals(ξ_{best}, P_s)
6 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathcal{Q}_n$
7 **if** $|\mathcal{Q}| > 0$
8 $\tau, g_\tau \leftarrow$ SynchronizeGoals($\mathcal{Q}, M$)
9 **if** $\tau \neq \bot$
10 $\tau \leftarrow$ OptimizeLocally(τ, g_τ)
11 $\tau_{\text{best}}, g_{\text{best}} \leftarrow \tau, g_\tau$
12 $\xi_{\text{best}} \leftarrow \xi(O_{g_\tau}(\tau(1)))$
13 $\mathcal{Q}_o = \{(q, g) \in \mathcal{Q} \mid \xi(O_g(q)) \leq \xi_{\text{best}}\}$
14 $\mathcal{Q} = \mathcal{Q} \setminus \mathcal{Q}_o$
15 **publish** $\tau_{best}, g_{best}, \xi_{best}$
16 **return** $\tau_{best}, g_{best}, \xi_{best}$

synchronization with a motion planning algorithm, and local optimization. The arm configurations are sampled in the sub-algorithm SampleGoals. Each sampled arm configuration q is associated with a grasp g, for which the arm configuration reaches a stable and physically feasible placement pose. All such sampled tuples (q, g) form the set Q, which serves as goal set for the motion planning algorithm.

The motion planning algorithm operates asynchronously to Algorithm 1. It explores the various free spaces $\mathcal{C}^g_{\text{free}}$ of the robot arm for the different grasps $g \in \mathcal{Q}$ to compute approach paths to the respective goal placements. The SynchronizeGoals function informs the motion planner about the current set of sampled goals $\mathcal{Q}$. If the motion planner has found a new path since the last call of SynchronizeGoals, it returns this path τ and its associated grasp g_τ. If multiple paths have been found, the path leading to the placement with highest objective value is returned.

Whenever a new path has been found, the path is locally extended by an arm motion that greedily moves the object to a placement with higher objective. This operation is performed in OptimizeLocally, and uses the pseudo-inverse of the arm's Jacobian to follow the gradient of the objective function. For more details on this operation, we refer to [8].

After this local optimization step, the goal set $\mathcal{Q}$ is updated to only contain goals that achieve higher objective values than the newly found solution. In addition, in subsequent iterations of the algorithm, the SampleGoals function will only return goals that achieve higher objective values. This way the algorithm achieves optimization and limits the verification of path-reachability only to placements poses that improve upon the objective.

1) Goal Sampling: The SampleGoals function is shown in Algorithm 2. The purpose of this algorithm is to locate stable, physically feasible, and kinematically reachable placement poses $\boldsymbol{x} \in \hat{S}$ with objective value $\xi(\boldsymbol{x}) > \xi_{\text{best}}$. This can be very challenging in narrow spaces as it is the case inside shelves and other furniture. Uniform random sampling has low probability of sampling collision-free poses and arm configurations in such environments.

Algorithm 2: SampleGoals: Sampling of pose, grasp and arm configuration

Input: Best achieved objective value ξ_{best}, state storage P_s
Constants: Number of maximal iterations $i_{\max}$,
Output: Feasible placement configurations $\mathcal{Q}_n$, updated state storage P_s
1 $\mathcal{Q}_n \leftarrow \emptyset$
2 **for** $i \leftarrow 1, \ldots, i_{max}$
3 $n \leftarrow$ SelectFRNode(P_s)
4 $\boldsymbol{x} \leftarrow$ SamplePose(n)
5 $g \leftarrow$ SelectGrasp($\boldsymbol{x}, n, P_s$)
6 $q \leftarrow$ IKSolver($\boldsymbol{x}, g$)
// If $\boldsymbol{x}$ is valid, add (q, g) to $\mathcal{Q}_n$
7 **if** $s(\boldsymbol{x}) = 1 \wedge c_f(\boldsymbol{x}) = 1 \wedge \xi(\boldsymbol{x}) > \xi_{best} \wedge q \in \mathcal{C}^g_{free}$
8 $\mathcal{Q}_n \leftarrow \mathcal{Q}_n \cup \{(q, g)\}$
9 Update($n, P_s, \boldsymbol{x}, q, \xi_{best}, g$)
10 **return** $\mathcal{Q}_n, P_s$

To alleviate this, we proposed in our previous work to use an adaptive sampling algorithm. The algorithm is based on Monte Carlo Tree Search, and focuses its sampling on regions of $\hat{S}$ that are likely to contain valid placement poses. These operations are performed in line 3 and 4 of Algorithm 2, and will be described in more detail shortly.

After sampling a pose $\boldsymbol{x}$, the algorithm selects a grasp $g \in \mathcal{G}(g_0)$ to compute an arm configuration to reach $\boldsymbol{x}$. The arm configuration is computed using a numerical inverse kinematics solver [20] starting at a random seed. If the solver succeeds, and the resulting arm configuration is collision-free, the pose is considered kinematically reachable with grasp g. If the pose $\boldsymbol{x}$ is also stable, $s(\boldsymbol{x}) = 1$, physically feasible, $c_f(\boldsymbol{x}) = 1$, and improving upon the objective, $\xi(\boldsymbol{x}) > \xi_{\text{best}}$, the algorithm found a new goal for the motion planner. The new goal is then added to the set of goal samples $\mathcal{Q}_n$. In the following the validity of the sample is reported in a data structure, P_s, that is used for the adaptive sampling and grasp selection. These operations are repeated for a constant number of iterations before the algorithm returns to Algorithm 1.

Exploiting Spatial Similarities: Both the pose sampling and the grasp selection procedure utilize a hierarchical partitioning of $\hat{S}$ from Eq. (2) to exploit spatial similarities between placement poses. The hierarchy encodes these similarities in dependence of the parameters f, r, x, y, θ. Here, the parameters f, r are the categorical choices of placement face and region in Eq. (2). The parameters x, y, θ denote the continuous translation and rotation of the object w.r.t the horizontal placement region r.

The first level of the hierarchy determines the base orientation of the object by selecting a placement face. The second level determines the placement region r. Accordingly, the nodes on the second level represent the pose sets $\hat{S}(r, f)$. From the third level on, each node represents a subset of $\hat{S}(r, f)$ by recursively partitioning the set. The range of positions are partitioned like a quadtree, and the range of orientations around the support surface's normal are partitioned in four sub-intervals on every level.

With this hierarchy, the algorithm tracks for each subset the number of samples obtained from the set and their validity. This information is stored in the variable P_s, and

Algorithm 3: SELECTGRASP: Choosing a grasp for a sampled placement pose.

Input: Sampled pose $\boldsymbol{x}$, Hierarchy node n in which the sampled pose lies, state storage P_s
Output: Feasible placement configurations $\mathcal{Q}_n$, state storage P_s

```
1  G ← ∅ // Grasp cache is an ordered set
2  p ← n
   // Retrieve ordered grasp cache
3  while p ≠ ROOT(P_s)
4      G ← G ∪ CACHEDGRASPS(p, P_s)
5      p ← PARENT(p)
6  if |G| = 0
7      G ← {g_0}
8  for g ∈ G
9      if GRIPPERCOLLISIONFREE(g, x)
           // accept the grasp with probability
              p_accept
10         if SAMPLEUNIFORMLY([0, 1]) ≤ p_accept
11             return g
12 if |G(g_0) \ G| = 0
13     return SAMPLEUNIFORMLY(G(g_0));
14 return SAMPLEUNIFORMLY(G(g_0) \ G);
```

maintained throughout the whole optimization process in Algorithm 1. In Algorithm 2, the function SELECTFRNODE uses this information to select a subset of $\hat{S}$ to sample from. The function operates in a similar way as a single iteration of Monte-Carlo Tree search using UCB1 [21] as in-tree policy. The Monte-Carlo rollout is performed in line 4 of Algorithm 2 by randomly sampling a pose $\boldsymbol{x}$ from the selected subset. For more details about this procedure we refer to [8].

Selecting a Grasp: The grasp set $\mathcal{G}(g_0)$ can encompass a large variety of grasps, making it detrimental to the algorithm's performance to try them all for every pose. Instead, the SELECTGRASP function samples a grasp from $\mathcal{G}(g_0)$. Rather than sampling $\mathcal{G}(g_0)$ uniformly at random, however, the SELECTGRASP function biases its grasp selection to grasps that likely enable the robot to reach the selected pose.

For this, the algorithm utilizes the hierarchical partitioning of $\hat{S}$, and stores for each sampled subset an ordered set of grasps in P_s. Initially these sets are empty. Whenever a kinematically reachable pose $\boldsymbol{x}$ is sampled from a subset associated with node n from the hierarchy, the corresponding grasp g is stored in the grasp set of n. In addition, the grasp g is propagated up in the hierarchy and stored for every ancestor of n.

The grasp selection process is shown in Algorithm 3. The algorithm receives the sampled hierarchy node n, the sampled pose $\boldsymbol{x}$ and the state variable P_s as arguments. Starting from n, the algorithm first collects all previously stored grasps in an ordered set G by ascending the hierarchy. The order of grasps in G is the order, in which the grasps are added. This allows the algorithm to first try the grasps that have been successful closest in the hierarchy to n. If no grasp has been stored before, G is initialized with the initial grasp g_0. This gives preference to place the object with the initial grasp if possible. Next, the algorithm iteratively tests for each grasp in G whether the robot's end-effector would be collision-free at the given object pose $\boldsymbol{x}$. If this is the case, the algorithm returns the tried grasp with a high probability.

A collision-free end-effector is not a sufficient condition for a placement pose to be reachable with a particular grasp. The end-effector pose may be out of the reachable workspace, or force the arm to penetrate an obstacle. For this reason, the algorithm only returns a collision-free grasp with probability p_{accept}. If none of the grasps in G were selected, the algorithm samples a grasp uniformly at random. In case G does not encompass all reachable grasps $\mathcal{G}(g_0)$, it randomly samples a grasp from $\mathcal{G}(g_0) \setminus G$. Otherwise, it re-samples any grasp from $\mathcal{G}(g_0)$.

C. Motion Planning

Each grasp results in a slightly different obstacle space $\mathcal{C}^g_{\text{obst}}$. Hence, to validate path-reachability we can not trivially employ a single motion planning algorithm. We address this problem by running separate motion planners for each grasp in $\mathcal{Q}$ in parallel. As algorithm we employ a modification of OMPL's [22] bidirectional RRT algorithm [23]. The modification allows us to add and remove backward trees while the algorithm is running. A backward tree is removed when the respective goal is removed from $\mathcal{Q}$, due to its objective being smaller than ξ_{best}.

V. EXPERIMENTS

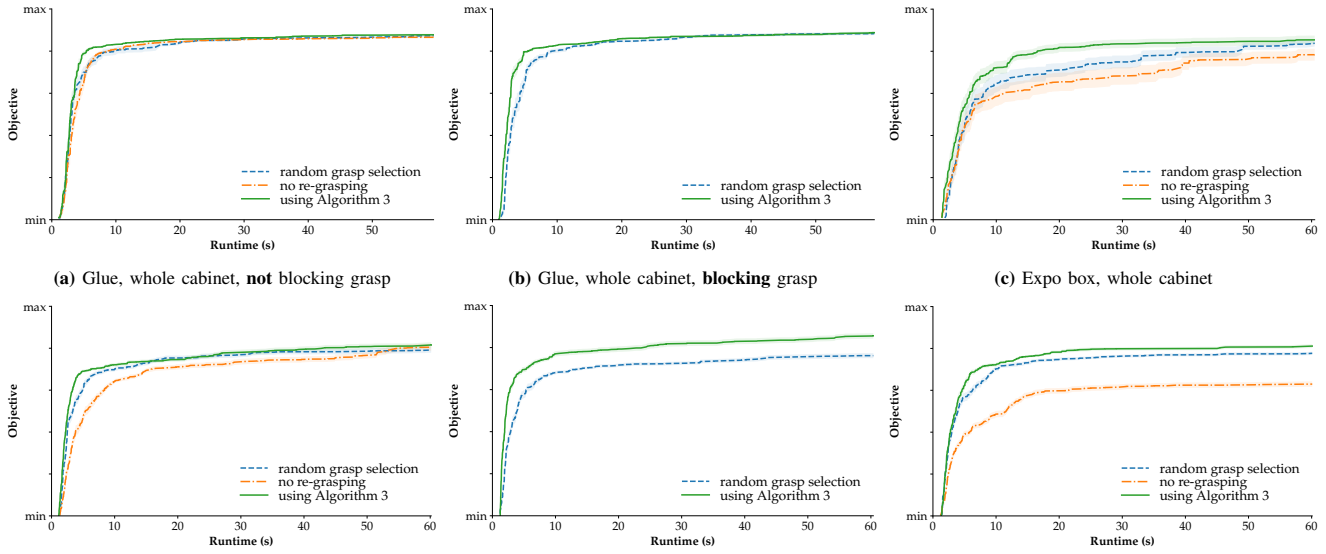
We implemented the presented framework in Python and C++ using OpenRAVE [24]. We generated the DMG for two considered objects: the Elmer's glue bottle and the Expo Dry Eraser box from the Amazon Picking Challenge 2016 objects set. In our experiments, we evaluate the performance of the planning algorithm for different initial grasp choices, and how enabling the robot to change its grasp affects the optimization performance. All evaluations were run on an Intel Core i7-4790K CPU @ 4.00GHz×4 with 16GB RAM running Ubuntu 18.04. For a video showing an execution on a real robot, we refer to our website[1].

In our first set of experiments, we query the algorithm to compute placements for the glue bottle shown in Fig. 1. We evaluate the algorithm on two different environments: the cabinet shown in Fig. 1, and the cluttered table top shown in Fig. 4a. In both cases the placement objective is to maximize the average distance to obstacles within the target volume: $\xi(\boldsymbol{x}) = D(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_o(\boldsymbol{x})|} \sum_{\boldsymbol{p'} \in \mathcal{B}_o(\boldsymbol{x})} d_{\mathcal{S}}(\boldsymbol{p'})$, where $\mathcal{B}_o(\boldsymbol{x})$ is a finite collection of points approximating the object at pose $\boldsymbol{x}$. The obstacle distance function $d_{\mathcal{S}} : \mathbb{R}^3 \to \mathbb{R}$ measures the distance to obstacles in the target volume V that are above or on the side of the object.

We ran the algorithm for the two different initial grasps shown in Fig. 1. The first grasp allows the object to be placed without changing the grasp, whereas the second one requires re-grasping. We ran the algorithm 50 times on each problem and recorded the objective values of the found solutions as a function of runtime. Fig. 3 (a,b,d,e) show the development of the mean objective values as time progresses.

The green solid line shows the optimization performance of the algorithm as presented in Sec. IV. The blue dashed

[1] https://joshuahaustein.github.io/plcmnt-web/



(a) Glue, whole cabinet, **not** blocking grasp

(b) Glue, whole cabinet, **blocking** grasp

(c) Expo box, whole cabinet

(d) Glue, table, **not** blocking grasp

(e) Glue, table, **blocking** grasp

(f) Expo box: table

Fig. 3: Planning and optimization performance of the different variations of our algorithm. All plots show the evolution of the mean objective value of computed placement solutions as a function of runtime. The shaded areas around the lines indicate the mean standard error. The objective values *min* and *max* are the minimal and maximal objective values that we ever observed for the respective scene.

line shows the performance of the algorithm without using Algorithm 3. Instead, the algorithm always samples a random grasp from $\mathcal{G}(g_0)$. The orange dashed-dotted line shows the performance of the algorithm without the ability to change its grasp.

In all test cases the algorithm using Algorithm 3 performs as good or better than the other versions. In the case where the initial grasp allows a placement, we do not observe any degradation in performance in comparison to the algorithm that does not consider different grasps.

Unsurprisingly, for the initial grasp that prohibits placing the glue bottle, the algorithm without re-grasping fails at finding any solution. Hence, there is no orange dashed-dotted line in Fig. 3 (b, e). In contrast, enabling re-grasping leads to similar optimization performance as with the non-blocking initial grasp. The benefit of Algorithm 3 is neglectable in these test cases except for the table scene with blocking grasp, Fig. 3 (e).

In the next set of experiments, we evaluate the optimization performance for the box-shaped Expo Dry Eraser object shown in Fig. 4b. In contrast, to the glue this object has six different placement faces always allowing a placement independent of the initial grasp. Again, we ran the algorithm for the two environments and recorded the optimization performance, Fig. 3 (c, f). For both environments the algorithms with re-grasping outperform the algorithm without re-grasping. This indicates that re-grasping not only renders infeasible queries feasible, but also eases the optimization problem, as a larger variety of placement poses becomes accessible.

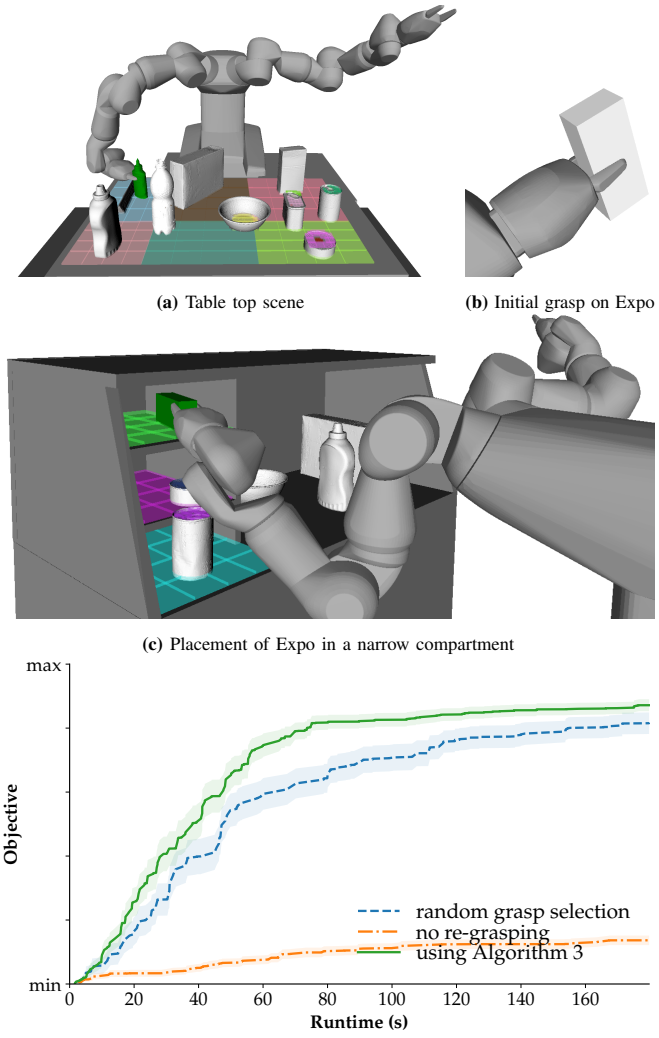
In our last set of experiments, we highlight another benefit of re-grasping. In this experiment, we query the algorithm to place the Expo Dry Eraser object with the initial grasp shown in Fig. 4b in the compartments on the left side of the cabinet, see Fig. 4c. We modify the objective function to $\xi(\boldsymbol{x}) = y + D(\boldsymbol{x})$, where y denotes the object's $y-$position. In the cabinet, the y-axis points into the cabinet. Thus this objective rewards placement poses that are deeper inside of the cabinet.

While it is possible with the initial grasp to place the object in front of the compartments, high objective placements are only reachable if the robot changes its grasp to reduce the object's footprint. This way it can reach deeper into the small compartments. The result of this experiment are shown in Fig. 4d. The problem is more difficult than the previous ones, due to the narrow space. Here, we observe a clear benefit of re-grasping. We also observe again a small advantage of using Algorithm 3.

VI. Discussion & Conclusion

We presented a placement planning algorithm that given an initial grasp on an object and a placement objective function, computes a sequence of in-hand pushing actions to adjust the grasp, and then transport the object to a placement maximizing the objective. For this, we integrated our prior works on Dexterous Manipulation Graphs [7] and placement planning [8]. Our experiments demonstrate that the resulting algorithm succeeds in computing high-quality placements even in challenging environments. The ability to change the grasp through in-hand manipulation can not only render infeasible placement queries feasible, but also benefits the algorithm's ability to locate high-quality placements.

In-hand manipulation is very well suited as preparatory manipulation for placing. In contrast to re-grasping with pick-and-place, it does not require to locate any intermediate placements. This allows us to address the placement problem largely isolated from the picking problem. While the DMG-based in-hand manipulation does not allow the robot to change to every possible grasp, the reachable grasp set is usually sufficiently diverse to allow horizontal placements.



(a) Table top scene **(b)** Initial grasp on Expo

(c) Placement of Expo in a narrow compartment

(d) Expo, compartments in cabinet

Fig. 4: Table top experiment scene, initial grasp on Expo Dry Eraser box and placement after re-grasping, and optimization performance for placing the Expo object deep inside the cabinet.

While our algorithm is capable of computing reachable placements also in difficult to navigate environments, its performance degrades the narrower the open spaces are. This is due to the sampling-based nature of the algorithm. The freedom of choosing different grasps further increases the burden on the motion planning component, as every grasp leads to a different collision space. The grasp cache is therefore in part designed to keep the total number of considered grasps low.

In future work, we plan to investigate whether the similarities between the motion planning problems for different grasps can be exploited to make this process more resource efficient. In addition, the assumption that the robot can perform in-hand manipulation in a safe region with a second arm currently limits the approach to dual arm robots. An interesting extension is to utilize a fixture in the environment to achieve the desired grasp change.

Finally, our algorithm currently only verifies that an approach motion to a target exists. In narrow spaces it can occur that a collision-free retreat for the robot arm is not possible after placing. Hence, in future work this additional constraint on the placement pose needs to be incorporated.

Acknowledgments

This work has been supported by the Swedish Foundation for Strategic Research and the Knut and Alice Wallenberg Foundation.

References

[1] T. Siméon, J. P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," in *IJRR*, vol. 23, no. 7-8, 2004.
[2] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *IJRR*, vol. 37, no. 1, 2018.
[3] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *ICRA*, May 2017.
[4] P. Tournassoud, T. Lozano-Perez, and E. Mazer, "Regrasping," in *ICRA*, March 1987.
[5] J. Ma, W. Wan, K. Harada, Q. Zhu, and H. Liu, "Regrasp planning using stable object poses supported by complex structures," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 11, no. 2, 2019.
[6] W. Wan, H. Igawa, K. Harada, H. Onda, K. Nagata, and N. Yamanobe, "A regrasp planning component for object reorientation," *Autonomous Robots*, July 2018.
[7] S. Cruciani, C. Smith, D. Kragic, and K. Hang, "Dexterous manipulation graphs," in *IROS*, 2018.
[8] J. A. Haustein, J. Stork, K. Hang, and D. Kragic, "Object placement planning and optimization for robot manipulators," in *IROS*, 2019.
[9] K. Harada, T. Tsuji, K. Nagata, N. Yamanobe, and H. Onda, "Validating an object placement planner for robotic pick-and-place tasks," *Robotics and Autonomous Systems*, vol. 62, no. 10, 2014.
[10] M. J. Schuster, J. Okerman, H. Nguyen, J. M. Rehg, and C. C. Kemp, "Perceiving clutter and surfaces for object placement in indoor environments," in *Humanoids*, Dec 2010.
[11] Y. Jiang, M. Lim, C. Zheng, and A. Saxena, "Learning to place new objects in a scene," *IJRR*, vol. 31, no. 9, 2012.
[12] P. Lertkultanon and Q. Pham, "A certified-complete bimanual manipulation planner," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, July 2018.
[13] A. E. Holladay, J. Barry, L. Pack Kaelbling, and T. Lozano-Perez, "Object placement as inverse motion planning," in *ICRA*, 2013.
[14] W. Wan, K. Harada, and F. Kanehiro, "Preparatory manipulation planning using automatically determined single and dual arms," *IEEE Transactions on Industrial Informatics*, 2019.
[15] N. C. Dafle, R. Holladay, and A. Rodriguez, "In-hand manipulation via motion cones," in *RSS*, 2018.
[16] J. Shi, J. Z. Woodruff, P. B. Umbanhowar, and K. M. Lynch, "Dynamic in-hand sliding manipulation," *T-RO*, vol. 33, no. 4, 2017.
[17] F. E. Viña, Y. Karayiannidis, C. Smith, and D. Kragic, "Adaptive control for pivoting with visual and tactile feedback," in *ICRA*, 2016.
[18] A. Sintov, O. Tslil, and A. Shapiro, "Robotic swing-up regrasping manipulation based on the impulse-momentum approach and clqr control," *T-RO*, vol. 32, no. 5, 2016.
[19] B. Sundaralingam and T. Hermans, "Relaxed-rigidity constraints: In-grasp manipulation using purely kinematic trajectory optimization," in *RSS*, 2017.
[20] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," in *Humanoids*, 2015.
[21] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2, 2002.
[22] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, December 2012, http://ompl.kavrakilab.org.
[23] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *ICRA*, 2000.
[24] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, 2010.