Object Placement Planning and Optimization for Robot Manipulators

Joshua A. Haustein[1], Kaiyu Hang[2], Johannes Stork[3] and Danica Kragic[1]

***Abstract*— We address the problem of planning the placement of a rigid object with a dual-arm robot in a cluttered environment. In this task, we need to locate a collision-free pose for the object that *a)* facilitates the stable placement of the object, *b)* is reachable by the robot and *c)* optimizes a user-given placement objective. In addition, we need to select which robot arm to perform the placement with. To solve this task, we propose an anytime algorithm that integrates sampling-based motion planning with a novel hierarchical search for suitable placement poses. Our algorithm incrementally produces approach motions to stable placement poses, reaching placements with better objective as runtime progresses. We evaluate our approach for two different placement objectives, and observe its effectiveness even in challenging scenarios.**

I. Introduction

Pick-and-place is among the most common tasks robot manipulators are applied for today. Grasp planning, which is the process of autonomously selecting grasps, still receives much attention and effort from the robotics community [1]–[3]. In contrast, the problem of placement planning, which is the process of autonomously deciding where and how to place an object with a robot, has received considerably less attention.

An autonomous robot tasked with placing a grasped object can generally not assume to know the environment in advance, rather it faces the following challenges when perceiving the environment for the first time:

1. It needs to identify suitable locations that afford placing. For instance, an object may be placed flat on a horizontal surface, leaned against a wall, placed on a hook, or laid on top of other objects. Determining how and where a particular object can be placed, requires analysis of both the environment's and the object's physical properties.
2. It needs to be able to reach the placement. Placing requires the robot to move close to obstacles, which make it difficult to compute collision-free arm configurations reaching a placement. In addition, the obstacles render planning an approach motion computationally expensive.
3. Not all placements are equally desirable. For many tasks, there exists an objective such as stability, human-preference on location, or clearance from other obstacles that is to be maximized.

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

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

[3]Center for Applied Autonomous Sensor Systems (AASS), Örebro University, Örebro, Sweden, E-mail: johannesandreas.stork@oru.se

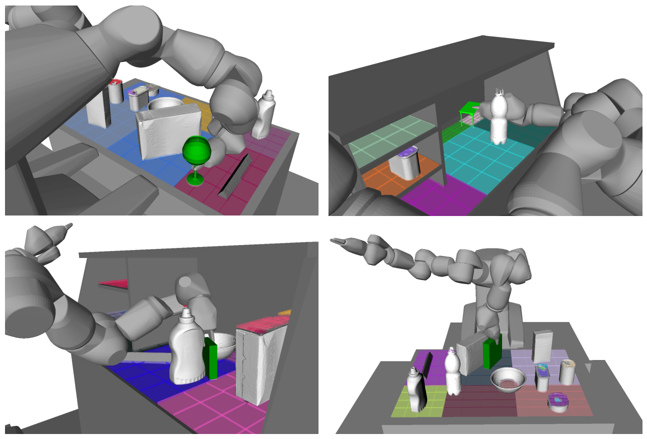


Fig. 1: Our algorithm computes placements for objects as well as corresponding approach motions in cluttered environments. In addition, it optimizes a user-specified objective for the placement pose. In the top row are example placements produced by our algorithm for a wine glass and toy table (*green*) under the objective to maximize clearance from other objects. In the bottom row a small and a large crayons box (*green*) are placed under the objective to minimize clearance.

This work's contribution is an algorithmic framework that addresses these challenges. Our main focus lies on computing reachable placement poses (challenge 2) that maximize a user-specified objective (challenge 3). In particular, we consider placing with a dual-arm robot in difficult to navigate environments, such as shelves and cluttered tabletops, Fig. 1. Our approach addresses challenge 2 under the consideration of different robot arms by integrating a motion planning algorithm with a novel hierarchical search for a placement pose. We address challenge 3 by designing the algorithm such that it finds an initial feasible solution quickly, and then incrementally improves the user-specified objective in an anytime fashion.

II. Related Works

Previous works on placing objects predominantly focus on challenge 1, i.e. searching poses in the environment, where an object can rest stably. A naïve solution consists of identifying horizontal surfaces in the environment and placing the object flat on the surface where there is enough space. This technique is, for instance, commonly employed in manipulation planning works which focus on planning complex sequences of pick-and-place operations rather than individual placements [4]–[8].

The object's orientation for a horizontal placement can be obtained by analyzing the object's convex hull and extracting the faces that support a stable placement [7], [8]. Each of these faces gives rise to a base orientation when aligned with the support surface. Different poses with the same base orientation can be obtained by translating the object along the surface and rotating it around the surface's normal. To locate collision-free and reachable placement poses (challenges 1 and 2) rejection sampling using a collision-checker and inverse kinematics solver is often employed. This is sufficiently efficient, if there are few obstacles and most sampled poses are within reach. If this is not the case, however, a more efficient sampling strategy, such as the one presented in this work, is required.

A more complex approach to locating placement poses (challenge 1) has been proposed by Harada et al. [9]. The approach locates placement poses by matching planar surface patches on the object with planar surface patches in the environment. This allows the approach to locate placements on large, flat surfaces, but also less obvious placements such as a mug hanging on a flat bar. While the work also integrates this algorithm with a motion planner (challenge 2), it does not perform any optimization of an objective (challenge 3).

Locating placement poses (challenge 1) has also been addressed using data-driven methods. Schuster et al. [10] train a classifier to segment a point cloud of a tabletop into clutter and support surfaces. Similarly, Jiang et al. [11] train a classifier to score the placement suitability of candidate poses based on 3D point-clouds of the object and the environment. The classifier evaluates physical feasibility, stability, as well as human placement preference. This enables the approach to produce a variety of placements, such as a plate standing in a dish-rack, a mug hanging on a bar, or a box laying on a flat surface. In order to evaluate the classifier, however, the approach requires a set of reachable candidate poses. Obtaining these in cluttered environments is non-trivial, as random sampling, for instance, has low probability of sampling good candidates.

Ensuring that a collision-free approach motion to a placement exists (challenge 2) requires us to closely integrate the placement search with a motion planning algorithm. This relates our problem to integrated grasp and motion planning [12]–[16]. These works present algorithms that simultaneously compute grasps with corresponding approach motions. The works demonstrate that in cluttered environments separate planning of grasps and approach motions is inefficient, due to collisions or the limited reach of the robot.

Our work addresses the analogous challenge for placing (challenge 2), with the addition of optimizing an objective function on the placement (challenge 3). We follow a similar idea as our prior work on integrated grasp and motion planning [12] and use a hierarchical sampling algorithm to better cope with the presence of clutter. Regarding challenge 1, we follow aforementioned previous works and place objects on horizontal support surfaces.

III. Problem Definition

We consider a dual-arm robot equipped with two manipulators, $\mathcal{A} = \{a_1, a_2\}$, that is tasked to place a *rigid* object o in a user-defined target volume $V \subset \mathbb{R}^3$. We assume that the object can be grasped by either arm with a grasp known a priori, and it's the algorithm's task to choose which arm to place with. The target volume V is a set of permitted positions for the object o, and restricts the search space for placement poses to $\mathcal{X}^o = V \times SO(3) \subset SE(3)$. Obviously, not all poses in $\mathcal{X}^o$ facilitate a stable placement, since for many of these the object might be, for example, in midair or intersecting obstacles. We denote the constraint that a pose $\boldsymbol{x} \in \mathcal{X}^o$ facilitates the stable placement of the object as binary mapping $c_s(\boldsymbol{x})$, that is 1 if $\boldsymbol{x}$ is a stable placement and 0 otherwise. Additionally, we denote the constraint that a pose $\boldsymbol{x}$ is physically feasible, i.e. that there is no intersection of the interior of the object with any obstacle, as binary predicate $c_f(\boldsymbol{x})$.

A placement pose must be reachable by the robot. For this, let $\mathcal{C}^a = \mathcal{C}^a_{\text{free}} \cup \mathcal{C}^a_{\text{obst}}$ denote the configuration space of arm $a \in \mathcal{A}$, and let $O(q)\colon \mathcal{C}^a \to SE(3)$ denote the pose $\boldsymbol{x} \in SE(3)$ of the object when grasped with arm a in configuration $q \in \mathcal{C}^a$. We say a pose $\boldsymbol{x} \in \mathcal{X}^o$ is *path-reachable*, $c_r(\boldsymbol{x}) = 1$, if for some arm $a \in \mathcal{A}$ there exists a known collision-free continuous path $\tau\colon [0,1] \to \mathcal{C}^a_{\text{free}}$ starting from the initial configuration of the robot $\tau(0) = q_0 \in \mathcal{C}^a_{\text{free}}$ and ending in a configuration $\tau(1) = q_g \in \mathcal{C}^a_{\text{free}}$ such that it reaches $\boldsymbol{x}$, i.e. $O(q_g) = \boldsymbol{x}$.

With these constraints and a user-provided objective function $\xi : \mathcal{X}^o \to \mathbb{R}$, we formalize our task as the following constrained optimization problem:

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathcal{X}^o}{\text{maximize}} \quad & \xi(\boldsymbol{x}) \\ \text{subject to} \quad & c_f(\boldsymbol{x}) &= 1 \\ & c_s(\boldsymbol{x}) &= 1 \\ & c_r(\boldsymbol{x}) &= 1 \end{aligned} \tag{1}$$

Independently of the objective function, the optimization problem is challenging to solve due to the constraints. The collision-free constraint, $c_f(\boldsymbol{x})$, renders the problem non-convex. The stability constraint, $c_s(\boldsymbol{x})$, is difficult to model, as it is a function of the physical properties of the object and the local environment. Lastly, the path-reachability constraint, $c_r(\boldsymbol{x})$, requires a motion planning algorithm to compute an approach path, which is generally computationally expensive.

Note that after releasing an object at a placement pose, the robot might not be able to retreat without colliding with the placed object. Hence, in principle, there is the additional constraint that a collision-free retreat motion must be possible. In this work, however, we choose to exclude this constraint from our problem definition and instead assume that a collision-free retreat is always possible.

Assumptions on prior information: We 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 $\mathcal{S} \subsetneq \mathbb{R}^3$. Furthermore, we assume that the environment is rigid, and that gravity acts antiparallel to the z-axis of the world's reference frame. Let ${}^{g}T^{a}_{o} \in SE(3)$ for $a \in \mathcal{A}$ denote the grasp transformation matrices from the object's frame to the respective gripper frames. We assume that these grasps

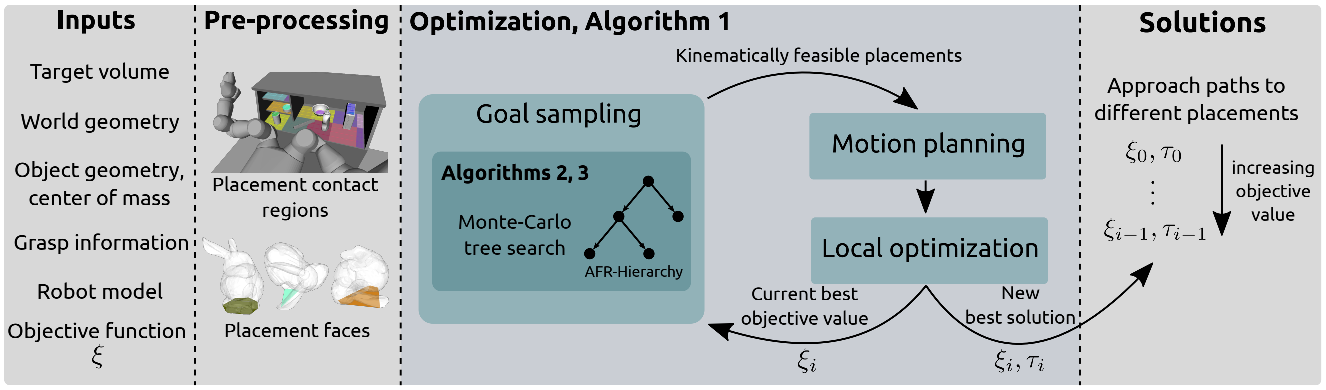


Fig. 2: Our approach consists of two stages. In a pre-processing stage we first extract placement contact regions and faces that help locating us stable object poses. In the optimization stage a sampling algorithm is employed to locate kinematically reachable and collision-free stable placement poses. These are provided to a motion planning algorithm to verify path-reachability and construct an approach motion. Subsequently a local optimization algorithm is employed to improve the placement locally. Any found solution is made available to the user, and subsequent iterations search for better solutions.

are selected such that a stable placement pose can be acquired without releasing the object.

For a pose $\boldsymbol{x} \in SE(3)$, let $\boldsymbol{p_x} = (x, y, z) \in \mathbb{R}^3$ be its position and $\boldsymbol{o_x} = (e_x, e_y, e_z)$ its orientation expressed in rotation angles around the world's x, y and z axis respectively. Our algorithm treats the objective function $\xi(\boldsymbol{x})$ as a black-box. If ξ is differentiable w.r.t. the x, y, e_z components of $\boldsymbol{x}$, however, we obtain these partial derivatives numerically and exploit them for local optimization.

IV. Method

We address the problem in Eq. (1) with the algorithmic framework shown in Fig. 2. The framework receives the information listed on the left as input and produces paths $\tau_i : [0, 1] \rightarrow \mathcal{C}^a_{\text{free}}$, each associated with an arm $a \in \mathcal{A}$, as output. The final configuration of each path $\tau_i(1)$ represents a placement solution using a particular arm, and places the object at a stable and collision-free placement pose $\boldsymbol{x} = O(\tau_i(1))$.

The framework consists of a pre-processing stage and an optimization stage. The pre-processing stage analyzes the world and object geometry to extract surface information for potential placements. The optimization stage operates in an anytime fashion and iteratively produces new paths τ_i that reach placements with better objective $\xi_i = \xi(O(\tau_i(1))) > \xi_j$ than the previous paths $\tau_j, j < i$.

The base idea of our approach is to decompose the problem in Eq. (1) into a search for feasible placement poses that fulfill all constraints, and only subsequently optimize the objective. In general, we can not model all constraints in Eq. (1) in closed form. However, for a particular pose $\boldsymbol{x} \in \mathcal{X}^o$ we can verify whether it fulfills the constraints. Therefore, the optimization stage operates in a sampling-based manner. For each constraint in Eq. (1) our framework has one component designed to verify it, or to provide samples from its satisfying set:

Stable placement. A necessary condition for a pose $\boldsymbol{x} \in \mathcal{X}^o$ to be stable, $c_s(\boldsymbol{x}) = 1$, is that the object is in contact with the environment. In the pre-processing stage, we therefore extract surfaces in the target volume and on the object that afford placing. With these surfaces we obtain an approximation $\hat{S} \subset \mathcal{X}^o$ of the set of stable placement poses that serves as search space for our optimization.

Physically and kinematically feasible placement. Within the set $\hat{S}$ we need to locate object poses that are physically feasible, $c_f(\boldsymbol{x}) = 1$. In addition, we need to verify that these poses can be reached by collision-free arm configurations $q \in \mathcal{C}^a_{\text{free}}$ for at least one arm. Sampling of such poses $\boldsymbol{x} \in \hat{S}$, and verification of $c_f(\boldsymbol{x})$ and $c_s(\boldsymbol{x})$ is performed in the optimization stage in a subprocess that we refer to as goal sampling.

Reachable placement. To verify path-reachability of sampled candidate poses, $c_r(\boldsymbol{x}) = 1$, we need to construct approach paths to them. For this, the optimization stage employs sampling-based motion planning [17] towards the arm configurations computed by the goal sampling algorithm.

Preferred placement. The actual optimization of the objective function is achieved through two concepts. First, we employ a greedy local optimization algorithm on the poses that have been verified to satisfy all constraints. Second, whenever an approach path to a new placement pose $\boldsymbol{x}$ has been found, we constrain subsequent iterations of the algorithm to only consider poses $\boldsymbol{x}'$ that achieve a better objective $\xi(\boldsymbol{x}') > \xi(\boldsymbol{x})$.

A. Pre-processing: Defining Potential Contacts

Modeling the set $S = \{\boldsymbol{x} \in \mathcal{X}^o \mid c_s(\boldsymbol{x}) = 1\}$ of stable placement poses is challenging, due to the large variety of possible placements. We approximate this set by the set of poses at which the object is placed on horizontal surfaces. For this, we extract a discrete set of *placement contact regions*, $\mathcal{R} = \{r_i\}_{i=1}^{i=m}, r_i \subset \mathcal{S} \cap V \subsetneq \mathbb{R}^3$, from the surface geometry $\mathcal{S}$ in the target volume V. A placement contact region is a contiguous set of surface points that share the same height, see Fig. 3. In our implementation, we extract these from an occupancy grid of the environment. Other techniques, however, could also be employed.

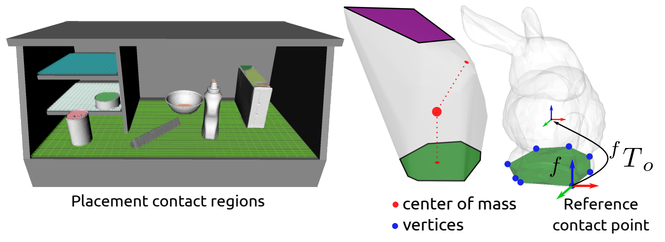


Fig. 3: Placement faces and contact regions. *Left:* We extract contiguous horizontal surfaces in the environment that provide us with contact locations to place the object on. *Right:* The Stanford bunny model is shown with its convex hull and two of the hull's faces are highlighted. The *green* face is a placement face, as the projection of the center of mass (*red*) along the faces' normal falls into the face. This is not the case for the *purple* face. To align the object with a horizontal contact region, we define for each placement face a transformation matrix ^fT_o that describes the object frame relative to a frame rooted at a reference vertex with the face's normal as z-axis.

To determine the orientation in which the object should make contact, we extract contact points from the object's surface. We follow a similar approach as in aforementioned works [7], [8] and select faces from the object's convex hull to place the object on. The convex hull of a point cloud of the object is a convex polyhedron. A face of this polyhedron supports a stable placement on a horizontal surface, if the projection of the object's center of mass along the face's normal falls into this face, see Fig. 3. We refer to the $k \in \mathbb{N}$ number of faces for which this is the case as *placement faces*, $\mathcal{F} = \{f_i\}_{i=1}^{i=k}$.

Each placement face is a polygon, and only its vertices are guaranteed to be part of the object's actual surface. For each face $f \in \mathcal{F}$, we select one of these vertices as reference contact point and define a transformation matrix ^fT_o as shown in Fig. 3. With this, the combination of a placement contact region $r \in \mathcal{R}$ and a placement face $f \in \mathcal{F}$ defines a class of object poses

$$\hat{S}(r,f) = \{T\big(R_z(\theta), \begin{pmatrix} x \\ y \\ z_r \end{pmatrix}\big)\, ^fT_o \mid \begin{pmatrix} x \\ y \\ z_r \end{pmatrix} \in r, \theta \in [0, 2\pi)\},$$

where z_r is the z-coordinate of the placement contact region, $R_z(\theta)$ the rotation matrix around the z-axis by angle θ, and $T(\cdot,\cdot)$ an operator that combines a translation vector and rotation matrix to a transformation matrix.

The poses in $\hat{S}(r,f)$ vary in x, y translation within the contact region r and rotation by θ around the z-axis going through f's reference contact point located at x, y, z_r. For all poses in $\hat{S}(r,f)$ the reference contact point is guaranteed to be in contact. Depending on the size of f and r, the other vertices are also likely in contact but contact is not guaranteed. Therefore, when sampling $\hat{S}(r,f)$ we need to verify stability of the sampled poses by ensuring that the remaining vertices are also in contact. Here, it is sufficient for these vertices to contact any placement contact region $r \in \mathcal{R}$, allowing placements where an object bridges a gap between regions. Finally, the union $\hat{S} = \bigcup_{r \in \mathcal{R}, f \in \mathcal{F}} \hat{S}(r,f)$ of all $\hat{S}(r,f)$ constitutes a parameterized approximation of S, that serves as our search space for feasible placements.

Algorithm 1: Optimization stage

```
1  $M_s, G_s \leftarrow \emptyset, \emptyset$   // Storage of internal state
2  $\tau, \xi_{\text{best}}, \mathcal{G} \leftarrow \perp, -\infty, \emptyset$
3  while not TERMINATE()
4      $\mathcal{G}_{\text{new}}, G_s \leftarrow$ SAMPLEGOALS($g_{max}, \xi_{best}, G_s$)
5      $\mathcal{G} \leftarrow \mathcal{G} \cup \mathcal{G}_{\text{new}}$
6      if $|\mathcal{G}| > 0$
7          $\tau, M_s \leftarrow$ PLANMOTION($m_{max}, \mathcal{G}, M_s$)
8          if $\tau \neq \perp$
9              $\tau \leftarrow$ OPTIMIZELOCALLY($\tau$)
10             $\tau_{\text{best}} \leftarrow \tau$
11             $\xi_{\text{best}} \leftarrow \xi(O(\tau(1)))$
12             $\mathcal{G}_o = \{g \in \mathcal{G} | \xi(O(g)) \leq \xi_{\text{best}}\}$
13             $\mathcal{G} = \mathcal{G} \setminus \mathcal{G}_o$
14             publish $\tau_{best}, O(\tau_{best}(1))$
15 return $\tau_{\text{best}}, O(\tau_{\text{best}}(1))$
```

B. Sampling-based Optimization

The optimization stage is formalized in Algorithm 1. The algorithm alternates between executing the sub-algorithms SAMPLEGOALS, PLANMOTION and OPTIMIZELOCALLY until termination is requested by the user. In each iteration, SAMPLEGOALS samples $\hat{S}$ to compute a finite set of collision-free arm configurations $\mathcal{G}_{\text{new}} = \{(q,a) \mid a \in \mathcal{A}, q \in \mathcal{C}^a_{\text{free}}\}$ reaching the sampled poses. For each returned $q \in \mathcal{G}_{\text{new}}$ the stability constraint, $c_s(O(q)) = 1$, and the physical feasibility constraint, $c_f(O(q)) = 1$, at the respective object pose $\boldsymbol{x} = O(q)$ are satisfied. Furthermore, the placements achieve a better objective than the best solution found so far, $\xi(O(q)) > \xi_{\text{best}}$ for all $q \in \mathcal{G}_{\text{new}}$. Initially, ξ_{best} is set to $-\infty$ and is updated as the algorithm succeeds at verifying path-reachability of placements.

In each iteration, the algorithm stores all sampled goals with an objective greater than ξ_{best} in a set $\mathcal{G}$. This set is provided as goal set to a motion planner in PLANMOTION. Whenever this algorithm succeeds at computing a new path, OPTIMIZELOCALLY is executed to improve the solution locally. Subsequently ξ_{best} and $\mathcal{G}$ are updated accordingly.

SAMPLEGOALS and PLANMOTION receive parameters $g_{\text{max}}, G_s, m_{\text{max}}, M_s$ respectively. The presence of the parameters G_s, M_s emphasizes that both sub-algorithms maintain an internal state across all iterations of the algorithm. This is crucial for the efficiency of the overall approach and will be detailed in the following sections. The parameters $g_{\text{max}}, m_{\text{max}}$ limit the computation time budget for each function, to balance the computational burden of sampling new goals and planning paths.

C. Goal sampling

The function SAMPLEGOALS needs to solve a constraint satisfaction problem:

$$\begin{aligned} \text{find} \quad & a \in \mathcal{A}, q \in \mathcal{C}^a \\ \text{such that} \quad & \xi(O(q)) > \xi_{\text{best}} \\ & c_s(O(q)) = 1 \\ & c_f(O(q)) = 1 \\ & q \in \mathcal{C}^a_{\text{free}} \end{aligned} \tag{2}$$

This subproblem by itself is challenging to solve. Given the approximate parameterized set $\hat{S}$, we could employ uniform

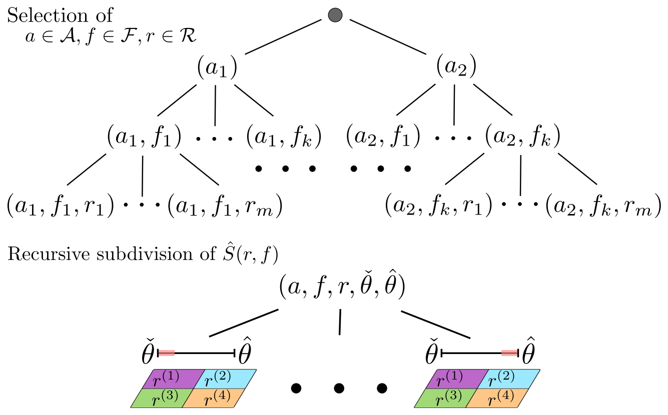


Fig. 4: The AFR-hierarchy consists of two different parts. On the first three level, the hierarchy represents choices of an arm $a \in \mathcal{A}$, a placement face $f \in \mathcal{F}$ and a region $r \in \mathcal{R}$. On the level at greater depths, the hierarchy recursively subdivides the region r and the range of orientations $[\check{\theta}, \hat{\theta})$ within a pose set $\hat{S}(r, f)$

sampling of $\hat{S}$ and reject invalid samples using an inverse kinematics solver and collision checker. This, however, can be rather inefficient in the presence of obstacles, where the probability of randomly sampling a collision-free object pose $\boldsymbol{x}$ and arm configuration $q \in \mathcal{C}^a$ reaching $\boldsymbol{x}$ is low. We remedy this by employing a sampling procedure that adapts its sampling and focuses on regions of $\hat{S}$ that are likely to satisfy all constraints.

1) AFR-Hierarchy: Sampling a pose from $\hat{S}$ involves choosing a placement contact region $r \in \mathcal{R}$ and a face $f \in \mathcal{F}$. In addition, to compute an arm configuration reaching a sampled pose, we need to select an arm $a \in \mathcal{A}$. While there is an overlap between the poses that each arm can reach, some may be more easily reached by one than the other. Whether a particular placement face f is a good choice to place the object on depends on the grasp, and thus on the arm that is selected. Similarly, whether a placement contact region allows a stable and penetration-free placement strongly depends on the placement face, as this determines the footprint and the base orientation of the object. Furthermore, if a pose $\boldsymbol{x} \in \hat{S}(r, f)$ for a particular region r and face f is reachable by an arm a, it is likely that poses in close proximity are also reachable by the arm. Hence, there exists a spatial correlation of feasibility within a set $\hat{S}(r, f)$, as well as between different sets of $\hat{S}(r, f)$ with similar categorical choices for $r \in \mathcal{R}, f \in \mathcal{F}$ and arms $a \in \mathcal{A}$.

This observation leads us to the definition of the AFR-hierarchy shown in Fig. 4. On the first level of this hierarchy, an arm $a \in \mathcal{A}$ is selected, on the second level a placement face $f \in \mathcal{F}$, and on the third a placement contact region $r \in \mathcal{R}$. From the third level on, each node in the hierarchy defines all quantities that we require to sample poses and compute arm configurations. Subsequent level of the hierarchy recursively partition the sets $\hat{S}(r, f)$, and every node represents a subset of the $\hat{S}(r, f)$ it descends from.

Let $n = (a, r, f, \check{\theta}, \hat{\theta})$ denote a node at level $i \geq 3$. The values a, r, f denote the arm, the placement face and the placement contact region the node n is associated with. The values $\check{\theta}, \hat{\theta}$ define a range of orientations $[\check{\theta}, \hat{\theta}) \subseteq [0, 2\pi)$ around the z-axis. On level $i = 3$, it is $\check{\theta} = 0$ and $\hat{\theta} = 2\pi$ and the nodes represent the full $\hat{S}(r, f)$ for the respective regions and faces. For nodes at level $i > 3$, the interval $[\check{\theta}, \hat{\theta}) \subset [0, 2\pi)$ defines a subset of orientations, and r denotes only a subset of the original contact region of its ancestor at level $i = 3$. On any level $i \geq 3$, the children of a node n arise from partitioning its region r and interval $[\check{\theta}, \hat{\theta})$. The region r is divided into four subregions $r = r^{(1)} \cup r^{(2)} \cup r^{(3)} \cup r^{(4)}$ by splitting it along its mean x and y positions. The interval $[\check{\theta}, \hat{\theta})$ is split into four equally sized sub-intervals. The 4×4 children of n then result from combining each subregion with each interval of orientation angles. This subdivision is recursively continued until some user-specified minimal region area and orientation range.

Algorithm 2: SampleGoals: Sampling algorithm based on Monte Carlo tree search

```
Input: Number of maximal iterations $g_{\max}$, best achieved objective
    value $\xi_{\text{best}}$, state storage $G_s$
Output: Feasible placement configurations $\mathcal{G}_{\text{new}}$, state storage $G_s$
1  $\mathcal{G}_{\text{new}} \leftarrow \emptyset$
2  for $i \leftarrow 1, \ldots, g_{max}$
3      $n \leftarrow$ SelectAfrNode($G_s$)
4      $\boldsymbol{x} \leftarrow$ Sample($n$)
5      if $c_s(\boldsymbol{x}) = 1 \wedge c_f(\boldsymbol{x}) = 1 \wedge \xi(\boldsymbol{x}) > \xi_{best}$
6          $q \leftarrow$ IKSolver($\boldsymbol{x}, a_n$)
7          if $q \in \mathcal{C}^{a_n}_{free}$
8              $\mathcal{G}_{\text{new}} \leftarrow \mathcal{G}_{\text{new}} \cup \{(q, a_n)\}$
9      Update($n, G_s, \boldsymbol{x}, q, \xi_{best}$)
10 return $\mathcal{G}_{\text{new}}, G_s$
```

2) Goals Sampling with Monte Carlo Tree Search: To obtain samples that satisfy Eq. (2), we exploit the spatial correlation modeled by the AFR-hierarchy, and employ a sampling algorithm based on Monte Carlo tree search (MCTS) [18]. The algorithm is shown in Algorithm 2 and Algorithm 3, where Algorithm 2 is the SampleGoal procedure called by Algorithm 1.

The key idea of the algorithm is to incrementally construct a tree of nodes in the AFR-hierarchy, and store for each node the proportion of valid samples obtained from its subbranch. This information is then used to focus sampling on branches of the hierarchy that are likely to contain more valid samples while maintaining some exploration. The tree is stored in the variable G_s, and thus steadily constructed across all executions of SampleGoal in Algorithm 1.

Every time Algorithm 2 is executed, it attempts to produce $g_{\max}$ goal samples (q, a), $q \in \mathcal{C}^a_{\text{free}}$, $a \in \mathcal{A}$. For each sample, the algorithm first selects a node n from the AFR-hierarchy using Algorithm 3. Algorithm 3 ensures that this node fully specifies a set $\hat{S}(r, f)$ or a subset thereof. It then randomly samples a pose $\boldsymbol{x}$ from this set and evaluates whether the pose constraints $c_f(\boldsymbol{x}), c_s(\boldsymbol{x})$ and $\xi(\boldsymbol{x}) > \xi_{\text{best}}$ are fulfilled. If this is the case, it employs an inverse kinematics solver to compute an arm configuration $q \in \mathcal{C}^{a_n}$ for the arm a_n specified by the AFR node n. If such a configuration exists and it is collision-free, a new goal has been obtained and can be provided to the motion planning algorithm.

Tree maintenance: After each sample step, the tree stored in G_s is updated according to whether we successfully obtained a new goal sample or not. For each sampled node n, we store the following information in G_s:

- $v(n)$, the number of samples obtained from n or any of its descendants,
- $r(n)$, the sum of all rewards obtained for sampling n or any of its descendants,
- $Ch(n, G_s)$, the children of n that have been added to G_s.

The numbers $v(n)$ and $r(n)$ are updated by the UPDATE function, whereas $Ch(n, G_s)$ is updated within Algorithm 3.

After sampling n we obtain a reward

$$\Delta r(n) = \begin{cases} H(\boldsymbol{x}, q, \xi_{\text{best}}) & \text{if n is not a leaf of AFR} \\ 1 & \text{if } c_\xi(\boldsymbol{x}, \xi_{\text{best}}) = 1 \ \wedge c_f(q) = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $c_f(q) = 1$ if $q \in \mathcal{C}^a_{\text{free}}$, and $c_\xi(\boldsymbol{x}, \xi_{\text{best}}) = 1$, if $c_s(\boldsymbol{x}) = c_f(\boldsymbol{x}) = 1$ and $\xi(\boldsymbol{x}) > \xi_{\text{best}}$. This reward is binary, if n is a leaf of the AFR-hierarchy and there are no further subdivisions of the pose set. If n is not a leaf, the reward is a heuristic value $H(\boldsymbol{x}, q, \xi_{\text{best}}) \in [0, 1]$ that also rewards samples only satisfying some of the constraints. In any case, the reward is recursively propagated to n's ancestors, n', to update their respective $v(n'), r(n')$. The number of samples, $v(n')$, is always increased by one as we acquired a single sample, whereas the accumulated reward $r(n')$ is updated by the reward $\Delta r(n)$ obtained at the sampled node n.

Tree exploitation and extension: The decision on which node to sample is made in the SELECTAFRNODE function, Algorithm 3. The algorithm always starts at the root of the AFR-hierarchy and descends to a node in the hierarchy that it decides to sample. Since we can produce samples only for nodes at depths $i \geq 3$, the algorithm always needs to descend at least to depth 3 before returning any node. For nodes n at depth $i > 3$ the algorithm descends to its children, as long as n is not a leaf and has been sampled before.

Initially, G_s only contains the root of the AFR-hierarchy. Hence, the only option the algorithm has is to select a child in the AFR-hierarchy that is not in G_s yet. This is done by the ADDCHILD operation, which selects a random child from the AFR-hierarchy and adds it to G_s. Let n now be any selected AFR-node that is already stored in G_s. We distinguish between its children $Ch(n, G_s)$ that are also stored in G_s and its total set of children $Ch(n)$ as defined by the hierarchy. If n has children in G_s, the algorithm may either descend to one of them, or add a new child to G_s, if $|Ch(n, G_s)| < |Ch(n)|$. The decision on what to do is based on the UCB1 policy [19], which is common to employ in Monte Carlo tree search and shown in Algorithm 3. It allows the algorithm to balance between re-sampling branches (exploitation) that have led to valid samples before and exploring new branches. The score u' for adding a new child is based on the conservative assumption that any unsampled node is as good as the average of its siblings.

Algorithm 3: SELECTAFRNODE: Selecting which AFR-node to sample

```
Input: State storage $G_s$
Output: node $n$ in $G_s$ that defines a tuple $(a, f, r, \check{\theta}, \hat{\theta})$
1  Function SELECTCHILD($n, G_s$)
2      for $i \in Ch(n, G_s)$
3          $u_i \leftarrow \frac{r(i)}{v(i)} + c\sqrt{\frac{2\ln(v(n))}{v(i)}}$
4      $u' \leftarrow -\infty$
5      if $|Ch(n, G_s)| < |Ch(n)|$
6          $u' \leftarrow \frac{1}{j}\sum_{i=1}^{j} \frac{r(i)}{v(i)} + c\sqrt{\frac{2\ln(v(n))}{j}}$
7      if $\forall i \in Ch(n, G_s) : u' > u_i \vee |Ch(n, G_s)| = 0$
8          return ADDCHILD($n, G_s$)
9      return $\arg\max_{i \in Ch(n, G_s)} u_i$
10 $n \leftarrow$ ROOT($G_s$)
11 for $d \leftarrow 1 \ldots 3$
12     $n \leftarrow$ SELECTCHILD($n, G_s$)
13 while $v(n) > 0 \wedge \neg$ISLEAF($n$)
14     $n \leftarrow$ SELECTCHILD($n, G_s$)
15 return $n$
```

D. Motion Planning

The subalgorithm PLANMOTION plans motions for each arm separately, as we assume that only one of the arms is used to perform the placement, while the other arm remains in a resting position. In principle, any motion planning algorithm could be employed for this subalgorithm. The only requirement on the algorithm is the possibility to efficiently add and remove goal configurations from the goal set $\mathcal{G}$, desirably without loosing information, e.g. samples in a search tree, that could be beneficial for planning paths to future goals.

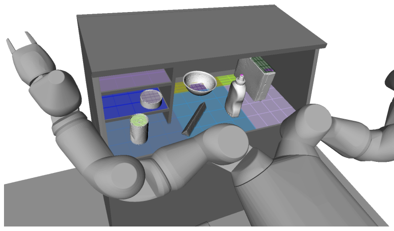
In our implementation, we employ a modification of OMPL's [20] bidirectional RRT algorithm [21]. The algorithm constructs a single forward tree and one backward tree for each goal in $\mathcal{G}$. It maintains these search trees throughout all iterations of Algorithm 1. Whenever the RRT algorithm succeeds in connecting the forward tree with a backward tree, the two trees are merged and success is reported. When $\mathcal{G}$ is modified, the backward trees rooting in goal configurations that have been removed are kept, as they may still prove valuable to reach other goals. When connecting to any of these, however, the algorithm no longer reports success and only merges it into the forward tree.

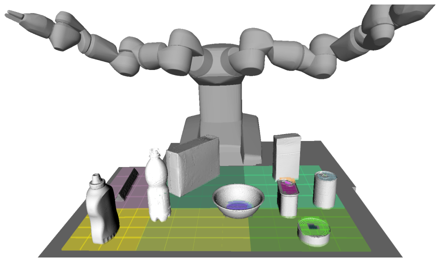
E. Local Optimization

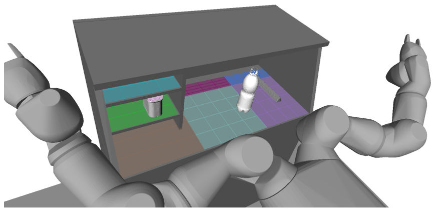
If the objective function ξ is differentiable w.r.t x, y, e_z, we exploit its gradient to locally optimize the placement that a solution path τ_i reaches. For this, let $q = \tau_i(1)$ be the final configuration of the path that reaches a placement pose $O(q) = \boldsymbol{x}$. We can locally improve the solution using the following update rule:

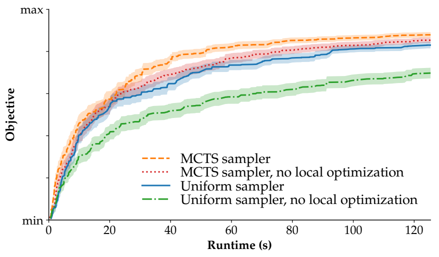
$$\Delta q \leftarrow J^\dagger \boldsymbol{v}(\frac{\partial \xi}{\partial x, y, e_z}(O(q)))$$
$$q \leftarrow q + \mu \Delta q,$$

where $J^\dagger$ is the pseudo-inverse of the arm's Jacobian at q, $\mu \in \mathbb{R}^{>0}$ a step size, and $\boldsymbol{v}(x, y, \theta) = (x, y, 0, 0, 0, \theta)^T$

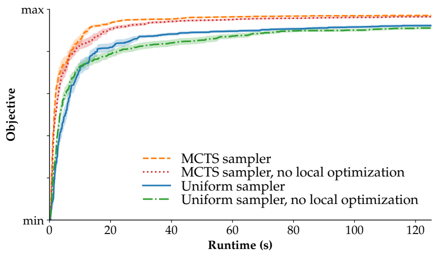

(a) Scene 1


(b) Scene 2

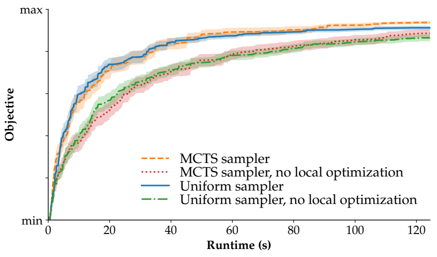

(c) Scene 3



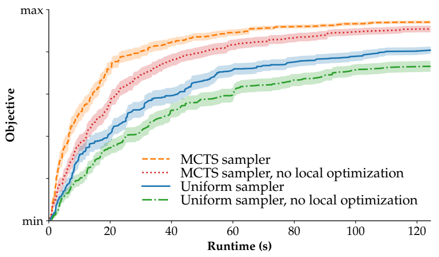
(d) Average optimization performance for all objects when **minimizing** clearance in scene 1.



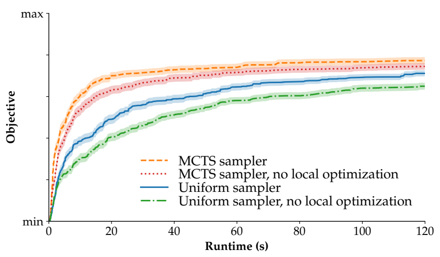
(e) Average optimization performance for all objects when **minimizing** clearance in scene 2.



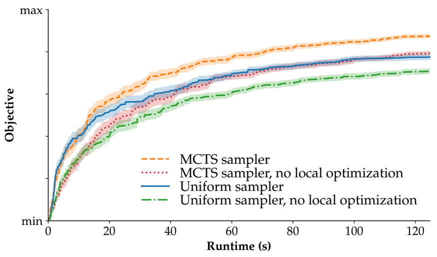
(f) Average optimization performance for all objects when **minimizing** clearance in scene 3.



(g) Average optimization performance for all objects when **maximizing** clearance in scene 1.



(h) Average optimization performance for all objects when **maximizing** clearance in scene 2.



(i) Average optimization performance for all objects when **maximizing** clearance in scene 3.

Fig. 5: Experiment scenes and optimization performance of different variants of our algorithm. In scene 1 and 3 the target volume encompasses the interior of the cabinet ($|V| = 0.13m \times 0.42m \times 0.29m$); in scene 2 the volume covers most of the table surface ($|V| = 0.8m \times 0.49m \times 0.2m$). The plots in **(d)** - **(i)** show the average optimization performance across all test objects achieved by each algorithm variant. The plots show the mean objective value as a function of planning time. The shaded area shows the standard error of the mean. In order to make the objective values comparable, we normalize the achieved objective values for each scene and object into the range of minimal and maximal objective observed throughout all executions.

lifts the three dimensional gradient to a six dimensional end-effector velocity. As long as the updated q is collision-free and $O(q)$ is not violating any constraints, we append the new configuration to the path τ, and thus obtain an improved feasible solution.

V. Experiments

We implemented our approach in Python using OpenRAVE [22] and the Open Motion Planning Library [20]. We evaluate the algorithm in three environments with different degrees of clutter for four objects, see Fig. 1 and Fig. 5. The objects differ in size, shape and number of placement faces. As robot we employ ABB's dual-arm robot Yumi, where each arm has 7 DoFs. For the AFR-hierarchy we choose a minimal placement contact region area of $0.005m \times 0.005m$, and minimal orientation interval of 0.025rad. The MCTS sampler uses an exploration parameter of $c = 0.5$ and the reward heuristic $H(\boldsymbol{x}, q, \xi_{\text{best}}) = \frac{1}{4}[c_f(q) + c_s(\boldsymbol{x}) + c_f(\boldsymbol{x}) + c_\xi(\boldsymbol{x}, \xi_{\text{best}})]$, where $c_f(\boldsymbol{x}), c_s(\boldsymbol{x}), c_f(q), c_\xi(\boldsymbol{x}, \xi_{\text{best}})$ are the binary constraint indicators used throughout Sec. IV. All experiments were run on an Intel Core i7-4790K CPU @ 4.00GHz$\times$4 with 16GB RAM.

As objective functions we employ two variations of clearance to obstacles. We define the clearance to obstacles as: $C(\boldsymbol{x}) = \frac{1}{|\mathcal{B}_o(\boldsymbol{x})|} \sum_{\boldsymbol{p'} \in \mathcal{B}_o(\boldsymbol{x})} d_{\mathcal{S}}(\boldsymbol{p'})$, where $\mathcal{B}_o(\boldsymbol{x})$ denotes a finite set of points approximating o when located at pose $\boldsymbol{x}$. The function $d_{\mathcal{S}} : \mathbb{R}^3 \to \mathbb{R}$ denotes the distance in x, y and positive z direction to the environment's surface within the target volume V. Maximizing this function, i.e. $\xi(\boldsymbol{x}) = C(\boldsymbol{x})$, leads to placements distant to obstacles. This is useful, for example, if the robot is tasked to manipulate the object further after placing. Minimizing this function, i.e. $\xi(\boldsymbol{x}) = -C(\boldsymbol{x})$, in contrast, is a useful heuristic for packing multiple objects into a limited volume. This objective is particularly interesting, as close proximity to obstacles renders placements difficult to reach.

As can be seen in Fig. 1, and better so in the accompanying video, our algorithm succeeds at computing placements with high objective values for all test cases. To evaluate the effectiveness of our algorithm design choices, we compare our algorithm to three variations. In the first variant, we omit the local optimization from Algorithm 1. In the second variant, we instead replace the Monte Carlo tree search-based sampling algorithm, Algorithm 2, with a naïve uniform random sampler. In the fourth variant, we also omit the local optimization when using the uniform sampler.

We ran each variant for each objective, object and scene 20 times for 2 minutes and recorded the objective values of the found solutions. The progress of the average objective values as a function of runtime is shown in Fig. 5.

In all test cases all variants compute initial solutions within a few seconds, and succeed at locating better solutions as time progresses. We observe that using both MCTS and local optimization performs better than, or as good, as all baselines. Even without local optimization, the MCTS sampling outperforms the uniform sampler with local optimization in most test cases. Only in scene 3 is the uniform sampler competitive when minimizing clearance. This simple scene has the least amount of obstacles, and thus sampling a feasible placement has high probability. Here, the sampler has little impact on the performance, and local optimization is most decisive for achieving better performance.

In all cases, the mean objective values increase quickly in the beginning before slowing down as they approach the maximum objective ever observed in the respective scene. This is likely due to the fact that the probability of locating poses that improve the objective declines the higher the best achieved objective is.

VI. Discussion & Conclusion

We presented an algorithmic framework that computes robot motions to transport a rigid object to a stable placement that optimizes a user-provided objective. Our approach achieves this also in environments cluttered with obstacles. The approach considers all available robot arms to reach the best placement and operates in an anytime-fashion, computing initial low-objective solutions quickly and improving as computational resources allow.

We combine sampling-based motion planning and local optimization with a hierarchical sampling algorithm based on MCTS. A key novelty lies in the AFR hierarchy and applying MCTS as sampling algorithm. While our experiments already demonstrate the advantage of this approach, we believe it has more potential. For instance, in this work the grasp, that we place the object with, is determined by the chosen arm. An interesting future extension is to incorporate different choices of grasps into the hierarchy. In addition, the sampling algorithm could prune branches of the hierarchy, that can not contain any solutions.

Further, we believe the approach can be extended to different types of placements. The different combinations of placement faces and regions constitute disjoint contact classes of object poses. In future work, we intend to extend the AFR hierarchy to more diverse contact classes, such as an object leaning against a wall.

Lastly, one of the weaknesses of the sampling-based optimization is the decreasing convergence rate observed in our experiments. To remedy this, we intend to investigate whether we can exploit gradient information not only in the local optimization step, but also in the goal sampling algorithm.

Acknowledgments

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

References

[1] A. Bicchi and V. S. A. Kumar, "Robotic grasping and contact: a review," in *ICRA*, 2000, pp. 348–353.

[2] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis – a survey," *TRO*, vol. 30, no. 2, pp. 289–309, 2013.

[3] M. Roa and R. Suárez, "Grasp quality measures: review and performance," *Auton. Robots*, vol. 38, no. 1, pp. 65–88, 2015.

[4] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *ICRA*, 2017, pp. 3426–3432.

[5] Z. Xian, P. Lertkultanon, and Q. Pham, "Closed-chain manipulation of large objects by multi-arm robotic systems," *IEEE RA-L*, vol. 2, no. 4, pp. 1832–1839, 2017.

[6] 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, pp. 104–136, 2018.

[7] W. Wan, H. Igawa, K. Harada, H. Onda, K. Nagata, and N. Yamanobe, "A regrasp planning component for object reorientation," *Autonomous Robots*, pp. 1–15, 2018.

[8] P. Lertkultanon and Q. Pham, "A certified-complete bimanual manipulation planner," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, pp. 1355–1368, July 2018.

[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, pp. 1463 – 1477, 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*, 2010, pp. 152–159.

[11] Y. Jiang, M. Lim, C. Zheng, and A. Saxena, "Learning to place new objects in a scene," *IJRR*, vol. 31, no. 9, pp. 1021–1043, 2012.

[12] J. A. Haustein, K. Hang, and D. Kragic, "Integrating motion and hierarchical fingertip grasp planning," in *ICRA*, 2017, pp. 3439–3446.

[13] J. Rosell, R. Suárez, and A. Pérez, "Path planning for grasping operations using an adaptive PCA-based sampling method," *Auton. Robots*, vol. 35, pp. 27–36, 2013.

[14] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner, "Grasp planning in complex scenes," in *Humanoids*, 2007, pp. 42–48.

[15] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Simultaneous Grasp and Motion Planning: Humanoid Robot ARMAR-III," *IEEE Robotics Automation Magazine*, pp. 43–57, 2012.

[16] J. Fontanals, B. A. Dang-Vu, O. Porges, J. Rosell, and M. A. Roa, "Integrated grasp and motion planning using independent contact regions," in *Humanoids*, 2014, pp. 887–893.

[17] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[18] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[19] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, 2002.

[20] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012, http://ompl.kavrakilab.org.

[21] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *ICRA*, 2000, pp. 995–1001.

[22] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, 2010.