Abstract—Many methods have been developed for planning the motion of robotic arms for picking and placing, ranging from local optimization to global search techniques, which are effective for sparsely placed objects. Dense clutter, however, still adversely affects the success rate, computation times, and quality of solutions in many real-world setups. The current work integrates tools from existing methodologies and proposes a framework that achieves high success ratio in clutter with anytime performance. The idea is to first explore the lower dimensional end effector’s task space efficiently by ignoring the arm, and build a discrete approximation of a navigation function, which guides the end effector towards the set of available grasps or object placements. This is performed online, without prior knowledge of the scene. Then, an informed sampling-based planner for the entire arm uses Jacobian-based steering to reach promising end effector poses given the task space guidance. While informed, the method is also comprehensive and allows the exploration of alternative paths over time if the task space guidance does not lead to a solution. This paper evaluates the proposed method against alternatives in picking or placing tasks among varying amounts of clutter for a variety of robotic manipulators with different end-effectors. The results suggest that the method reliably provides higher quality solutions to such challenges in a reasonable amount of time.

In problems where solutions exist in workspace clutter, there is a need to effectively guide the search process. The key insight is that in manipulation tasks, goals are typically end-effector centric i.e., grasping configurations. Consequently, heuristic guidance in manipulation tasks should reason about the end-effector. Additionally, the planning algorithm has to be designed to effectively use such guidance to find high-quality solutions quickly.

Towards solving such a challenge, this work proposes the Jacobian Informed Search Tree (JIST) method, which is a heuristic-guided sampling-based search algorithm for prehensile manipulation planning in the presence of dense clutter. The key idea is to plan around the constraints induced on the end-effector by the presence of clutter, so as to guide the motions of the arm towards areas which lead to a solution. The contributions of this work are: 1) developing an effective heuristic for guiding the arm by solving the motion planning problem for just the end effector; 2) applying the pseudo-inverse Jacobian as a steering process which allows the arm to be guided by the heuristic; and 3) incorporating task space guided maneuvers effective in clutter into an asymptotically optimal motion planner.

The performance of JIST is evaluated experimentally through comparison with other state-of-the-art planners for picking and placing tasks, including both trajectory optimization (CHOMP-HMC) and sampling-based methods (Grasp-RRT, IK-CBirRT). The results indicate that JIST, in contrast to the alternatives, can quickly and reliably compute high-quality solutions in the presence of dense clutter in unknown scenes, for a variety of robotic platforms, without requiring parameter tuning.

I. INTRODUCTION

A variety of robotic tasks, such as warehouse automation and service robotics, motivate computationally efficient and high-quality solutions to prehensile manipulation. Typical applications involve the need to pick and place a variety of objects from tabletop or shelf-like storage units. Such manipulation challenges are demonstrated in Fig. 1.

Consider a tabletop and shelf manipulation challenge, as shown in Fig. 2. Prior work [4] has shown that large amounts of clutter can significantly reduce the viable valid grasps on an object. While traditional strategies successfully find solutions in the tabletop setting, more cluttered environments like the shelf results in a massive degradation of performance, as highlighted in Fig. 2. One possible explanation is that the clearance of the solutions found in cluttered scenes is low, where clutter refers to the minimum distance between the robot geometries and the rest of the environment. This work accordingly refers to scenes with low clearance as densely cluttered, which have narrow passages in the workspace that make the motion planning problem harder. Nevertheless, coming up with high-quality motion planning solutions to such challenges in a reasonable amount of time remains an important objective.

II. BACKGROUND AND RELATIVE CONTRIBUTION

The proposed method draws inspiration from both broad categories of approaches for planning arm motion: a) local optimization and b) global search.

Local optimization follows a locally valid gradient towards finding a solution. Artificial potential fields incrementally move the current robot configuration by following such a
Some approaches deal with integrated grasp and motion planning [25][26]. This work focuses on motion planning aspects and uses the output of grasping planners [27][28][29], i.e., grasps to set the goals of the corresponding picking challenges, which are frequently defined as end effector poses relative to a target object.

### III. Problem Setup

Consider a robot arm with \( d \) degrees of freedom, which must solve pick and place tasks involving a target object \( o \) in a workspace with a set of static obstacles \( \mathbb{O} \).

**Assumption 1:** The geometric models of object \( o \) and obstacles \( \mathbb{O} \) are known. Online, before calling the planner, a perception module provides the 6D pose \( p_o \in SE(3) \) of \( o \), as well as the 6D poses of obstacles \( \mathbb{O} \).

This setup can be easily extended to point cloud representations, permitting the utilization of recent work on 6D pose estimation [30]. Note that this does not allow precomputation that reasons over the object or obstacle poses, as this information is available only online.

#### A. Goal Task Space Motion Planning

**Definition 1 (C-space):** An arm’s configuration \( q \) is a point in the robot’s C-space: \( \mathbb{C}_{\text{full}} \subset \mathbb{R}^d \). The colliding C-space \( (\mathbb{C}_{\text{obs}} \subset \mathbb{C}_{\text{full}}) \) corresponds to configurations where the arm collides with \( \mathbb{O} \) or \( o \) in their detected poses. The free C-space is then defined as: \( \mathbb{C}_{\text{free}} = \mathbb{C}_{\text{full}} \setminus \mathbb{C}_{\text{obs}} \).

The motion planning problem to be solved in manipulation setups is frequently not defined directly in \( \mathbb{C}_{\text{free}} \). Instead, the task relates to the arm’s end effector pose.

**Definition 2 (Pose Space):** The end effector’s pose space \( \mathbb{E} \subseteq SE(3) \) corresponds to the reachable subset of poses for the arm’s end effector.

The pose space \( \mathbb{E} \) can be computed given the forward kinematics of the robotic arm \( FK: \mathbb{C}_{\text{full}} \rightarrow \mathbb{E} \). The inverse kinematics function corresponds to the inverse mapping: \( IK: \mathbb{E} \rightarrow \mathbb{C}_{\text{full}} \). For redundant manipulators, \( dim(\mathbb{E}) < dim(\mathbb{C}) \), so \( IK \) describes a mapping to self-motion manifolds of the manipulator [24], composed of states, which all have the same end effector pose.

**Definition 3 (Goal-Constrained Task Space (GCTS)):** Given a set of goal poses \( \mathbb{E}_{\text{goal}} \subset \mathbb{E} \) for the end effector, the arm’s goal task space \( \mathbb{Q}_{\text{goal}} \subset \mathbb{C}_{\text{full}} \) denotes the robot configurations that bring the end effector to a pose in the set \( \mathbb{E}_{\text{goal}} \): \( \mathbb{Q}_{\text{goal}} = \{ q \in \mathbb{C}_{\text{full}} | FK(q) \in \mathbb{E}_{\text{goal}} \} \).

For a redundant arm, even a discrete pose set \( \mathbb{E}_{\text{goal}} \) gives rise to a continuous submanifold of solutions \( \mathbb{Q}_{\text{goal}} \). Given the above definition, the objective is to solve the following:

**Definition 4 (Motion Planning with a GCTS):** Given a start arm configuration \( q_{\text{start}} \in \mathbb{C}_{\text{free}} \) and a set of goal poses for the end effector \( \mathbb{E}_{\text{goal}} \subset \mathbb{E} \), a solution path is a continuous curve \( \pi : [0, 1] \rightarrow \mathbb{C}_{\text{free}} \), so that \( \pi(0) = q_{\text{start}} \) and \( \pi(1) \in \mathbb{Q}_{\text{goal}} \). Given the space of all solution paths \( \mathbb{P} \), the objective is to find the path \( \pi^* = \arg\min_{\pi \in \mathbb{P}} C(\pi) \), which minimizes a cost function \( C \) defined over \( \mathbb{P} \).
B. Defining Goal Poses for Pick and Place Instances

For picking, the goal is to move the end effector to a pose that allows for a stable grasp.

Assumption 2: A grasping planner produces a set of stable, collision-free grasps for the arm’s end effector and the target object \( o \) at the object’s detected pose \( p_o \).

A database of grasps can be computed in the object’s frame [31][27][28][29] and evaluated in terms of their stability offline [32]. Online, the grasps are transformed given the object pose \( p_o \) and collision checked, given the end effector’s geometry and the obstacles \( \mathcal{O} \). Nevertheless, states that correspond to grasps are on the boundary of \( C_{\text{free}} \) with \( C_{\text{obs}} \), which result in very hard motion planning problems. For this reason, the goal set for picking instances is defined for “pre-grasp” poses: each grasp returned by the grasping planner is transformed by moving the end effector along a vector away from the object as shown in Fig. 3. A solution returned by the planner that brings the end effector to a “pre-grasp” pose, needs to be appended with a motion so that the end effector achieves the actual grasp.

![Fig. 3. a) Pre-grasp pose, b) post-grasp state, c) object pre-placement.](image)

A similar setup is used for placing, where in the initial state the target object is immobilized by the end effector. In this case, the arm-object system can be considered as the state the target object is immobilized by the end effector. In this case, the arm-object system can be considered as the state the target object is immobilized by the end effector. The current work proceeds tied to the motions of the end effector, since the goals and poses are again extended so that the robot moves the object from the grasp state to the actual goal placement.

For a goal object placement, the pose is transformed so that the robot moves the object is again raised away from the surface. Planning for placement, a task planner produces a set of stable object placements on a support surface.

Assumption 3: For placement, a task planner produces a set of stable object placements on a support surface.

Again, it is helpful to avoid defining problems that bring the robot close to the boundary of \( C_{\text{free}} \). Thus, if the initial state has the object on a resting surface, the state is transformed so that the object is raised away from the surface. For a goal object placement, the pose is transformed so that the object is again raised away from the surface. Planning solutions from “post-grasp” states to “pre-placement” goal poses are again extended so that the robot moves the object from the grasp state to the actual goal placement.

C. Distance and Cost Function

It has been established that the problem at hand is closely tied to the motions of the end effector, since the goals and heuristic can be described by it. The current work proceeds to define a distance function appropriate for this setting, that minimizes end effector displacement. Another benefit of this is that paths that minimize end effector motion tend to look more natural to human observers who pay more attention to the end effector rather than the rest of the arm [33].

Towards this objective, this work employs the \( \text{DISP} \) distance function, which is a model-aware distance metric for \( SE(3) \), which can be approximated efficiently by the \( C \sim \text{DIST} \) algorithm [34]. Given the convex-hull \( H \) of the end-effector’s model, the \( \text{DISP} \) distance \( D \) of two poses \( e_i, e_j \in SE(3) \) is:

\[
D(e_i, e_j) = \max_{p \in H} ||p(e_i) - p(e_j)||_2
\]

With slight abuse of notation, the expression \( D(q, e) \) between \( q \in C_{\text{full}} \) and end effector pose \( e \in \mathcal{E} \) will denote the distance \( D(FK(q), e) \). It follows that the distance between \( q_i, q_j \in C_{\text{full}} \) is \( D(FK(q_i), FK(q_j)) \). Then, the cost of a path \( \pi \) is:

\[
C(\pi) = \sum_{i=0}^{k} D(\pi(i), \pi(i+1))
\]

where \( k \) is a discretization along \( \pi \). For the remainder of the paper, unless otherwise stated, distances will refer to Eq. [1] and costs will refer to Eq. [2].

IV. JACOBIAN INFORMED SEARCH TREE

This section describes a process for exploring the end effector’s space \( \mathcal{E} \) and how the corresponding information is used to guide the exploration of the arm’s space \( C_{\text{full}} \).

A. Exploring the End Effector’s Pose Space

A simple way to estimate how far the end effector is at pose \( e \in \mathcal{E} \) from the set \( \mathcal{E}_{\text{goal}} \) is to consider the minimum \( \text{SE}(3) \) distance from \( e \) to all goal poses \( \mathcal{E}_{\text{goal}} \). Nevertheless, in the presence of clutter, these distances are not informative of the end effector’s path cost so as to reach goal poses.

To take clutter into account, this work proposes the offline construction and online search of a roadmap \( R_{ee}(V_{ee}, E_{ee}) \) in \( \mathcal{E} \) as shown in Fig. 4. The vertices \( V_{ee} \) store reachable end effector poses given the arm’s kinematics, while the edges store local connection paths for the free-flying end effector. The goal poses \( \mathcal{E}_{\text{goal}} \) and the start pose \( e_{\text{start}} = FK(q_{\text{start}}) \) are attached to \( R_{ee} \), which is then searched online for a path from \( \mathcal{E}_{\text{goal}} \) to \( e_{\text{start}} \). Performing the search in this direction produces a multi-start search tree rooted at \( \mathcal{E}_{\text{goal}} \). During the online search, only collisions between the end effector, the obstacles \( \mathcal{O} \), and the object \( o \), but not the arm, are taken into account. This allows for quick estimations of the costs to reach \( \mathcal{E}_{\text{goal}} \). These costs are used later as heuristics to guide a search procedure in the arm’s \( C\) - space.

![Fig. 4. (left) Construction of \( R_{ee} \); (right) Online search of \( R_{ee} \).](image)

Offline Construction of \( R_{ee} \): The end effector reachability roadmap \( R_{ee}(V_{ee}, E_{ee}) \) is constructed offline without knowledge of the workspace and reflects the robot’s reachability. A sampling-based process is followed similar to \( \text{PRM}^+ \) [19]. A random arm state \( q_r \in C_{\text{full}} \) is sampled and the corresponding end effector pose \( FK(q_r) \in \mathcal{E} \) is stored as a vertex. Each vertex is then connected with an edge to the closest \( \log|V_{ee}| \) vertices in terms of \( D(q_r, e) \). Each edge stores an interpolation in \( \text{SE}(3) \) between the two vertex poses according to a discretization defined by an estimation of the end effector’s maximum velocity.

Online Computation of Distances to Goal Poses: Given knowledge of the workspace, \( R_{ee} \) is searched for collision-free end effector paths from the goal poses \( \mathcal{E}_{\text{goal}} \) to the start
pose $e_{start}$. The poses $E_{goal}$ and $e_{start}$ are added as vertices on $R_{ee}$ and are connected to their closest $\log |V_{ee}|$ neighbors.

Then, a multi-start, multi-objective $A^*$ (MSMO $A^*$) search procedure is performed on $R_{ee}$. Upon initialization, the priority queue of the search includes all goal poses $E_{goal}$. Then, each search node corresponds to a path from a pose in $E_{goal}$ to a vertex $u \in V_{ee}$. The priority queue sorts nodes so as to minimize the following two ordered objectives:

- $f_1(u)$, the number of colliding poses along the path from $E_{goal}$ to $u$, given the roadmap’s edge discretization;
- $f_2(u) = g_2(u) + h_2(u)$, where $g_2$ is $C(\pi(E_{goal}, e_u))$ and $h_2$ is $D(e_u, e_{start})$.

Once a vertex $u$ has been expanded, it is added to the “closed list” $L_{closed}$. If there is a collision-free end effector path from $E_{goal}$ to $e_{start}$, the above search will report the shortest path. Then, the $g_2$ cost of vertices $u$ in $L_{closed}$ corresponds to an estimate of the distance along the shortest collision free path from $u$ to $E_{goal}$, i.e., $g_2$ is a discrete approximation of a navigation function in $E_{goal}$ given the presence of obstacles. The proposed approach uses these $g_2$ values to heuristically guide the exploration in $C_{full}$. In dense clutter, if the roadmap’s resolution does not permit collision-free paths, the algorithm returns the path with the minimum number of collisions given its discretization. In this case, the $g_2$ values of vertices in $L_{closed}$ will guide the arm exploration along the shortest, least colliding solutions for the end effector.

### B. Generating Arm Paths

This section describes how to generate paths for the arm given the vertices $u \in V_{ee}$ stored in the “closed list” $L_{closed}$ and the cost estimates $g_2$ from $u$ to $E_{goal}$. An illustration of the steps of the algorithm can be seen in Figure 5.

The algorithm is outlined in Alg. 1 which is inspired by a search strategy [20] capable of utilizing heuristic guidance. It receives as input the start arm configuration $q_{start}$, the goal poses $E_{goal}$, the end effector reachability roadmap $R_{ee}$, the number of iterations $N$, and the parameter of the maximum number $\kappa$ of greedy edge expansions per iteration. The first step of the method is to execute the multi-start, multi-objective $A^*$ search over $R_{ee}$ outlined in the previous section, in order to acquire the closed list $L_{closed}$. This contains the vertices of $R_{ee}$ visited during the search and the corresponding cost estimates to the goal poses $E_{goal}$. The method then builds a search tree whose nodes correspond to arm configurations. The tree is rooted at the start configuration $q_{start}$ (line 3) and is expanded towards configurations that bring the end effector to goal poses.

Each iteration of the algorithm starts by selecting a node of the search tree so as to expand it (lines 6-9). If a node was added during the previous iteration, and its heuristic value is better than its parent on the tree, then the newly added node is selected for expansion (line 6-7). The heuristic value $h(q, L_{closed})$ of a configuration $q$ corresponds to an estimate of the cost to reach $E_{goal}$ approximated using $L_{closed}$.

If there was no progress made in the previous iteration towards reaching $E_{goal}$, the SearchSelection subroutine (line 9) instead uses a probabilistic selection process, where the probability of selecting a node depends on the node’s corresponding sum of cost from the root and heuristic cost-to-go. All nodes are guaranteed to have a non-zero probability of being selected; however, nodes with better costs and heuristic sum have a greater probability.

Once a configuration $q_{new}$ is selected for expansion, the set $A_{cand}$ represents a set of actions that can be used to extend the tree out of $q_{new}$, which can correspond either to target poses for the end effector or joint velocities for the arm. The first time that node $q_{new}$ is selected for expansion (line 10), the subroutine GreedyEdges is used to generate target poses for extending the tree out of $q_{new}$ (line 11).

**JIST** iterates over the generated actions, which are prioritized in terms of the lowest $h$ in terms of the resulting pose and the best one $q_{best}$ is considered for addition at each iteration (line 14). $Steer$ uses an appropriate steering method to generate a trajectory where the state reached at the end corresponds to $q_{new}$ (line 16). The configuration $q_{new}$ must satisfy two conditions so as to be added as a new node (line 17): (a) A branch and bound process (BaB) ensures that $q_{new}$ has smaller path cost relative to the length of the best solution found $\pi^*$; (b) A collision checker $CC$ verifies whether the path from $q_{sel}$ to $q_{new}$ is in collision. If the node $q_{new}$ passes these checks (line 19), then it is added to the tree $T$. If a better path to the goal is discovered with the addition of $q_{new}$, it is recorded as $\pi^*$ (line 20).

**Greedy Edge Generation:** These greedy edges are guided

<table>
<thead>
<tr>
<th>Algorithm 1: JIST ($q_{start}, E_{goal}, R_{ee}, N, \kappa$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $L_{closed} \leftarrow$ MSMO $A^*$($R_{ee}, q_{start}, E_{goal}$);</td>
</tr>
<tr>
<td>2 $\pi^* \leftarrow \emptyset$;</td>
</tr>
<tr>
<td>3 $T \leftarrow {q_{start}};$</td>
</tr>
<tr>
<td>4 $q_{new} \leftarrow q_{start}$;</td>
</tr>
<tr>
<td>5 for $N$ iterations do</td>
</tr>
<tr>
<td>6 if $q_{new} \neq \emptyset$ and $h(q_{new}, L_{closed}) &lt; h(\text{parent}(q_{new}), L_{closed})$ then</td>
</tr>
<tr>
<td>7 $q_{sel} \leftarrow q_{new}$;</td>
</tr>
<tr>
<td>8 else</td>
</tr>
<tr>
<td>9 $q_{sel} \leftarrow \text{SearchSelection}()$;</td>
</tr>
<tr>
<td>10 if $A_{cand}(q_{sel}) = \text{NULL}$ then</td>
</tr>
<tr>
<td>11 $A_{cand}(q_{sel}) \leftarrow \text{GreedyEdges}(q_{sel}, L_{closed}, \kappa)$;</td>
</tr>
<tr>
<td>12 else if $A_{cand}(q_{sel}) = \emptyset$ then</td>
</tr>
<tr>
<td>13 $A_{cand}(q_{sel}) \leftarrow \text{FallbackEdges}(q_{sel})$;</td>
</tr>
<tr>
<td>14 $q_{best} \leftarrow \arg \min_{a \in A_{cand}} h(a, L_{closed})$;</td>
</tr>
<tr>
<td>15 $A_{cand}(q_{sel}) \leftarrow A_{cand}(q_{sel}) \setminus q_{best}$;</td>
</tr>
<tr>
<td>16 $q_{new} \leftarrow \text{Steer}(q_{sel}, q_{best})$;</td>
</tr>
<tr>
<td>17 if BaB($q_{new}, \pi^*$) or $CC(q_{sel} \rightarrow q_{new})$ then</td>
</tr>
<tr>
<td>18 $q_{new} \leftarrow \emptyset$;</td>
</tr>
<tr>
<td>19 if $q_{new} \neq \emptyset$ then</td>
</tr>
<tr>
<td>20 $\pi^* \leftarrow \text{AddEdge}(T, q_{sel} \rightarrow q_{new})$;</td>
</tr>
<tr>
<td>21 $A_{cand}(q_{new}) \leftarrow \text{NULL}$;</td>
</tr>
<tr>
<td>22 return $\pi^*$;</td>
</tr>
</tbody>
</table>
The objective of the Jacobian-based steering process is to compute the joint controls which solve $\Delta e = J^* \Delta q$. The method uses the pseudo-inverse of the Jacobian, $J^+$, so as to minimize $||J^+ \Delta q - \Delta e||^2$, where $\Delta q = J^+ \Delta e$. To account for singularities, damping [35] and clamping are also employed. Thus, at time $t$, for the configuration $q_t$ and the corresponding end effector pose $e_t$, the control update rule for $J^+ \text{steering}$ is:

$$\Delta q_t = J^+ (q_t) \cdot (e_{\text{target}} - e_t)$$

(3)

Since the objective is to minimize $\text{DISP}$, the gradient followed by the above rule reduces the distance of the end effector from the target pose. Any configuration $q_{\text{goal}} \in Q_{\text{goal}}$ discovered through $J^+ \text{steering}$ is kept track of. A benefit to using $J^+ \text{steering}$ is that it satisfies the necessary properties of the projection operator [24] needed to ensure coverage of $Q_{\text{goal}}$. $J^+ \text{steering}$ is used in the connection of “pre-grasps”, “post-grasps”, and “pre-placements” as described in Section III.

V. Experiments

JIST is compared against other planners for solving manipulation challenges. Recent work [26] has utilized CBIRRT [1] for solving grasping problems. Accordingly, this work compares against an IK-based variant of CBIRRT, which uses IK on the $E_{\text{goal}}$ to construct roots of the goal tree. Grasp-RRT corresponds to an RRT variant, which utilizes $J^+ \text{steering}$ to discover grasp states during the goal biasing phase [3]. CHOMP-HMC corresponds to an OpenRave [36] prob. complete implementation of CHOMP [2].

Setup: A common planning software was used for the sampling-based planners [37]. All methods were evaluated on a single Intel Xeon E5-4650 processor with 8 GB of RAM. Experiments were conducted on a variety of robotic manipulators: Kuka LBR iiwa (7 DOF), Rethink Baxter (14 DOF), and Motoman SDA10F (16 DOF). Three types of end effectors were evaluated: the ReFlex hand, a parallel-jaw, and a vacuum suction-cup. Each benchmark involved computing a trajectory for one of the arms to a set of goal end-effector poses. For pick benchmarks, the poses correspond to pre-grasps 2 cm away from the actual grasps. For place benchmarks, the goal poses corresponded to pre-placements for the object 5 cm above the resting surface.

Metrics: The number of planning successes and solution quality were measured over time, and an average over 50 runs was reported at each time interval. A success was counted if the planner computed a collision-free trajectory from the start state to the goal end-effector pose within the corresponding time. Solution quality was measured using $\text{DISP}$.

Initialization: Both JIST and CHOMP required some additional initialization prior to attempting to solve any of the benchmarks. JIST required $R_{ee}$ to be first computed offline (without knowledge of the scene). For the experiments, the size of $R_{ee}$ was 25,000 vertices, and the maximum
number of maneuvers $\kappa$ was set to 20. For CHOMP each environment had a signed-distance field constructed for it, and also had its parameters tuned per benchmark, since no single set of parameters sufficed for all. For CHOMP and IK-CBirRT, reachable, and collision-free arm configurations were computed through IK at goal poses.

**Pick Benchmarks:** The results for success rate and solution cost over time are shown in Figure 6. Across all experiments, JIST converged to a 100% success rate within the time limit of 30 seconds, with an average initial solution time breakdown shown in Figure 7 (left), while also providing the lowest cost solutions in all cases. The end effector heuristic was shown to be effective in guiding the search to produce high quality initial solutions. An interesting side-note here regarding CHOMP is that when initialized with solution states found by JIST and with some additional parameter tuning, the performance of CHOMP improved significantly. This indicates that solutions out of JIST can serve as better initializations for trajectory optimization methods.

**Place Benchmarks:** The results for the place benchmarks are shown in Figure 8 with average solution times shown in Figure 7. On average, JIST spent longer in the tree expansion portion of the method, relative to the pick benchmarks due to the attached target object causing more collisions. Despite this initial slowdown, JIST still managed to converge to a 100% success rate before all other methods, while also still providing the lowest cost solutions.

**Remarks:** The average clearance (i.e. shortest distance between the robot and the obstacles) of solutions found by all methods in the table environment was 5.7 cm, whereas in the shelf environment it was 2.2 cm. Accordingly, the table benchmarks were the easiest for all methods to solve. This is primarily due to the fact that overhand grasps over the table were viable solutions. However, in the shelf benchmarks, JIST maintains its high performance despite the clutter severely reducing the overall clearance and viable grasps.
This paper presented the Jacobian Informed Search Tree (JIST) algorithm, which is an asymptotically optimal, informed sampling-based planner for controlling an arm in densely cluttered scenes so as to achieve desired end-effector configurations. The method optimizes a cost function representing the displacement of the end-effector, and uses a heuristic computed by effectively searching the end-effector’s task space. JIST employs Jacobian-based steering to bias the expansion of the tree towards end-effector poses that appear promising given the heuristic guidance. The method was shown to have high success rate, even in densely cluttered scenes, with fast initial solution times and high quality solutions.

There are several directions for further exploration. For instance, goal constraints can be introduced for different arm links, such as a camera attached to the arm. One of the benefits of using Jacobian-based steering is that it allows for the satisfaction of secondary objectives in the null-space of the primary objective to reach the target pose. In this way, JIST could also incorporate additional constraints during task execution, such as maintaining certain orientations of the grasped object or confining the end-effector to a particular workspace region. Recent work in more complex IK-solvers could also be utilized to more efficiently guide the search tree [38]. Another direction is the consideration of alternative cost functions, especially time-based and multi-objective functions. Finally, it is interesting to evaluate the performance of JIST without knowledge of the obstacle geometries by operating directly over point clouds [39].

REFERENCES