IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 7 December 2022, accepted 20 December 2022, date of publication 26 December 2022,
date of current version 30 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3232130

RESEARCH ARTICLE

Local and Global Search-Based Planning for Object Rearrangement in Clutter

JINHWI LEE[1,2], (Student Member, IEEE), CHANGJOO NAM[3], (Member, IEEE), JONG HYEON PARK[2], (Member, IEEE), AND CHANGHWAN KIM[1], (Member, IEEE)

[1]Korea Institute of Science and Technology, Seoul 02792, South Korea
[2]Department of Mechanical Engineering, Hanyang University, Seoul 04763, South Korea
[3]Division of Electronic Engineering, Sogang University, Seoul 04107, South Korea

Corresponding author: Changhwan Kim (ckim@kist.re.kr)

This work was supported in part by the Technology Innovation Program and Industrial Strategic Technology Development Program (Development of Service Robot Technologies for Cleaning a Table) under Grant 20018256, and in part by the Korea Institute of Science and Technology (KIST) Institutional Program under Project 2E31560.

ABSTRACT We propose two algorithms based on local and global searches for a Task and Motion Planning (TAMP) problem, which considers a robotic manipulator to rearrange obstacles and grasp a target in clutter. In the problem, no collision-free path for a robotic manipulator is available unless some obstacles blocking a target are relocated. The two algorithms determine the sequence of obstacles to be removed for grasping a target without collisions. The local search algorithm determines the sequence quickly in an online manner but could be suboptimal in the number of removed obstacles. The global search algorithm based on tree search needs upfront computation but runs in polynomial time. In numerical simulation settings, we consider objects in various shapes, which could make reachable directions bounded. From the simulations the planning time of local search algorithm is faster than that of global search, whereas the global search algorithm removes the less number of obstacles than the local search. In addition, we show that the global search algorithm with a heuristic cost is faster than without the cost but the minimum number of removed obstacles is still obtained from the global search algorithm without the heuristic cost. Practical experiments show the applicability of our algorithms in real environments

INDEX TERMS Task and motion planning, object rearrangement, local and global search, manipulation planning.

I. INTRODUCTION

In a workshop or home environment, a robot could perform manipulation tasks in cluttered environments, where movable objects may occlude each other as seen in Fig. 1 (e.g., bin picking in warehouses). When a target object is occluded by other objects (so called obstacles), it is necessary to rearrange the obstacles and grasp the target without collisions.

In such cluttered environments as Fig. 1 overhand grasping may not be available geometrically so that the approaching motions from the side or front are essential. This approaching may request several sub-tasks to remove some obstacles blocking a target object. When a robot wants to move an object into a certain spot, the sequential actions of pick-and-place are necessary. The more objects to move, the more pick-and-place actions are requested. In other words, a smaller number of objects to move could make the number of pick-and-place actions decrease. Eventually, reducing the number of robot actions for removing the obstacles affects the efficiency of rearrangement and target grasping. Due to this, the present work focuses on minimizing the number of obstacles to be removed. The rearrangement problem determines the sequence of obstacles to be removed and is known to be computationally difficult as mentioned in [1]. In addition, the problem could become much harder, if the reachable direction to objects is bounded by the shapes of objects, which is dealt with in this work as well.

The associate editor coordinating the review of this manuscript and approving it for publication was Zheng Chen.

FIGURE 1. An example of cluttered environment. Certain objects need to be removed from the clutter to grasp a target object.

Some previous works propose rearrangement methods based on motion or path planning. Dogar and Srinivasa [2] present a planning framework that generates a straight line path from an end-effector to a target object. Removing the obstacles on the path is planned as sub-tasks and other objects may also be removed to get spaces to place such obstacles removed from the path. Similarly, in [3], the motion of a manipulator is generated first without considering the obstacles and then all the obstacles in collision with the robot are removed. Murali et al. [4] suggest the rearrangement method considering the accessibility between a gripper and a target object. The method is not intended to minimize the rearrangement task to grasp a target object. However, considering the 6-dof of the gripper poses, it finds an obstacle that prevents the manipulator from grasping a target object efficiently in real-world. Sundermeyer et al. [5] also propose a learning based method called Contact-GraspNet to find the grasping points of an object in clutter by considering the object's pose and collisions with other objects. The method calculates the 6-DOF of a gripper with contact points between the gripper and an object, even if an unlearned object is given. Danielczuk et al. [6] propose a method to place an object without collisions in clutter. The method converts objects and an environment scene into point clouds and finds the 6-DOF pose of an object for a gripper to place it safely in the clouds of an environment. Moll et al. [7] suggest the planner that plans a path to a target by pushing all the obstacles on the path aside a bit. The planner generates a path by a random tree expansion whose strategy is a concern about improving the coverage of the tree rather than determining the number of obstacles to be pushed. Papallas and Dogar [8] also suggest the method to grasp a target by pushing objects. When a human commands the start and end points of the end-effector, a robot generates a motion that pushes objects around a target. Ren et al. [9] propose a rearrangement planning algorithm to make a gap between dense blocks and help a finger-gripper grasp the target with ease. The algorithm applies a heuristic function to RRT (Rapidly exploring Random Tree) to reduce planning time. Eitel et al. [10] propose a method of singulating each object so that a finger-gripper can grasp the objects by considering the grasping points of the objects in clutter. The method generates a push plan by learning images using CNN. The images represent the expected state when objects are pushed. Pinto et al. [11] and Yuan et al. [12] propose methods that do not directly optimize energy or time for accomplishing a manipulation task but mainly consider the validity of their plans. Therefore, it is observed that some obstacles may be removed unnecessarily. Most of these approaches focus on finding the obstacles to be relocated, not reducing the number of obstacles seriously. In addition, they do not consider such a constraint as the reachable directions of objects that could affect both of task and motion planning in real world. Thus, our goal is to plan as small numbers of sub-tasks as possible for grasping a target object, considering the reachable directions of objects.

Another works aim to optimize the performance of the object rearrangement to be a goal configuration. Han et al. [13] solve an optimization problem for object rearrangement by using a tree search algorithm, which aims at minimizing the number of pick-and-place actions and the travel distance of the end-effector. Krontiris and Bekris [14] and Han et al. [15] also propose the methods that minimize the number of object movements for a rearrangement, where objects are stored in stack-like containers. These methods are similar to the Tower of Hanoi algorithm. Eljuri et al. [16] and Wang et al. [17] also propose rearrangement planning algorithms to relocate objects for a given goal state by checking the motion feasibility of the state. Although their algorithms show high efficiency and solution quality, the problems are not involved with complex motion planning and only considers placing objects on the top of the container. On the other hand, scalability of a planning algorithm is important as cluttered environments have a nontrivial number of objects. Vega-Brown and Roy [18] produce high quality solutions based on a graph search for rearrangement problems, which are to move objects to a designated space by a mobile robot. The environments in the problems are not cluttered due to the small number of objects compared to the size of map, so increasing scalability of them remains unsolved. These rearrangement problems are, however, slightly different from the present one, since the present one is to find what obstacles to be relocated in what order for grasping a target.

We propose two Task and Motion Planning (TAMP) algorithms that decide the obstacles to be relocated. The first one is based on locally searching the obstacles that are most accessible by the end-effector of a robot. The accessibility of obstacles is evaluated by employing Vector Field Histogram+ (VFH+) [19], which finds obstacle-free directions for approaching an obstacle. This algorithm runs fast (less than 0.05 sec to generate a plan for ten objects) and is complete. It could, however, make a suboptimal solution in aspect of the number of obstacles to be removed. We develop another algorithm to find an optimal solution using Breadth-First Search (BFS) of a tree representing possible manipulation actions and resultant states. The optimal solution removes the minimum number of obstacles so that the total execution time can be reduced and the computation time significantly by pruning and ordering as well (e.g., less than

TABLE 1. Nomenclature.

N	the total number of objects
t	target object
v	node of tree in the global search
v_p	parent node of v
O	set of all the objects in robot workspace including t (i.e. $O = \{o_1, o_2, \cdots, o_{N-1}, t\}$)
o_r	an obstacle to be removed
O_p, O_x	sets of objects in the nodes v_p and x, respectively
O_s	set of obstacles to be removed
K	the number of elements of O_s
M	robot configuration
Q	frontier implemented by the node v
U	frontier implemented by $d_{\max}$
$d_{\max}$	the max. radial distance from the center of an object to grasp for computing histogram
H	histogram of the sector in robot workspace
H_{G}	histogram of the sector having minimum magnitude
H_{G}^t	H_{G} of the target object
sector	the angle range surrounding a given object (i.e. $-180°$ to $180°$ in the x-axis of histogram)

0.6 sec with ten objects). The two algorithms deal with the reachable directions of objects, which are bounded by the shapes of objects and end-effector. We run some simulations for the two algorithms and compare the results. We attach a video clip describing the algorithms and the simulation results as well.

II. PROBLEM DESCRIPTION

In this section, we describe the terminologies, the definition of a direction angle, and the assumptions for the present problem in detail. First, the notations and terminologies used in the paper are given as seen in TABLE 1.

A. CONCEPT AND DEFINITIONS OF DIRECTION ANGLES

We define tree types of direction angles of an object for grasping as seen in Fig. 2. *Accessible* directions are the angle range that is not blocked by obstacles so that the end-effector can approach an object in the directions without collisions. *Reachable* directions denote the angle range which is determined according to the shape and size of an object and the workspace of an end-effector. *Reachable* directions can vary depending on a gripper for the same object. *Graspable* directions are then obtained by the intersection of *accessible* and *reachable* directions, in which the end-effector can approach the object and grasp it without collisions.

B. ASSUMPTIONS

We consider the problem of grasping a target object, which requires a robot to rearrange obstacles and avoid collisions. The problem deals with an environment with N objects including a target object t. The basic assumptions are give as follows: (i) The robot knows the configurations and geometries (i.e., pose, shape, size) of all objects including a target. (ii) The end-effector has no collision-free path to a target without relocating some obstacles.[1] (iii) The removed obstacles are placed outside the environment, which is predetermined. (iv) Overhand grasping is not allowed, since the present work focuses on manipulating of objects in such environment as seen in Fig. 1. (v) There is at least one object that has enough space for manipulation such that the finger of a gripper has a graspable direction for the object.[2] (vi) Objects can be grasped in different reachable directions owing to their various shapes. The reachable directions of objects are known as *a priori*.

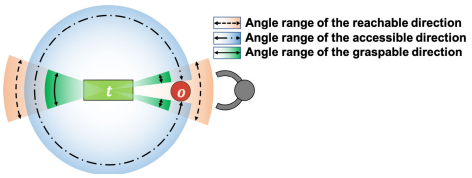


FIGURE 2. The definitions of three types of direction angles for grasping an object (*t, a green box*): *Accessible* directions denote the angle range around on an object except the angle blocked by an obstacle (*o*, a red circle). *Reachable* directions are obtained according to the geometries (herein shape) of an object and an end-effector. The intersection of *accessible* and *reachable* directions represents the graspable directions. If the end-effector approaches the object from one of the *graspable* directions (Three graspable directions are possible in the figure), it can grasp the object without collisions. If none of objects occludes a target, *reachable* directions are identical to *graspable* directions.

Among these assumptions, the last may bring more challenges in establishing a task plan for rearrangement, since grasping motions could be constrained significantly by the limited reachable directions of objects. For example, a cuboid with different side lengths as in Fig. 2 would be grasped only in one side (the shorter), if the size of end-effector is not wide enough. Therefore, task and motion planning needs to consider the reachable directions of objects carefully.

C. PROBLEM STATEMENT

In this work, our goal is to remove as few obstacles as possible and obtain a collision-free path for an end-effector to grasp a target object in clutter. Removing an obstacle assigns an additional grasping task and necessarily yields collision-checking between objects and a robot body, which causes significant computational efforts to find valid paths. Therefore, reducing the number of obstacles to be removed could generate less grasping actions and save time for extra collision checks in motion planning, which finally decreases the total execution time for rearrangement. In formulation, we aim to find a sequence of obstacles to be removed $O_s = \{o_r | r = 1, \cdots, K\}$ from $O = \{o_1, o_2, \cdots, o_{N-1}, t\}$, where O is the set of all N objects including a target t. The objective of problem is to minimize K and find the corresponding O_s, where $K = |O_s|$.

III. METHODS FOR OBSTACLE REARRANGEMENT CONSIDERING GRASPABLE DIRECTIONS

In this section, we describe the two algorithms of LocalSearch and GlobalSearch. Both of them employ Vector Field

[1]When a collision-free path to a target exists, our method is able to give that path without requesting rearrangement.

[2]The violation of assumption (v) means that there is no object to be grasped without collision due to very small gaps between objects. A robot manipulator cannot then grasp any object and the algorithm is terminated.

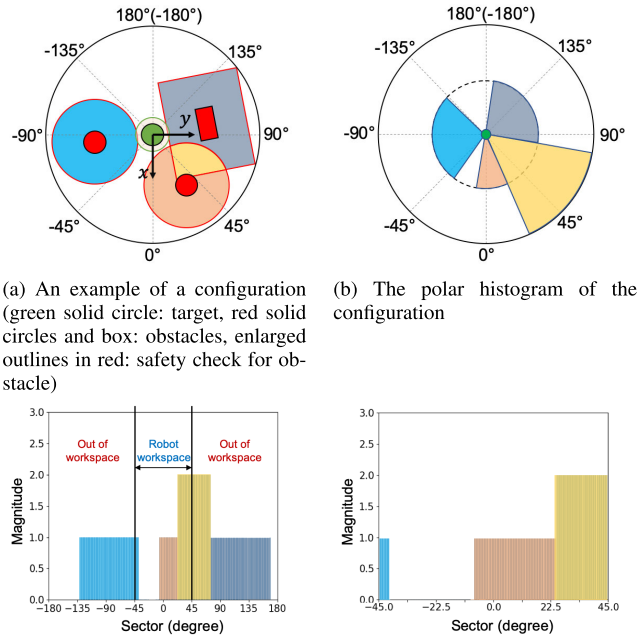
Histogram+ (VFH+) [19], which has been widely used for mobile robots to find obstacle-free directions for autonomous navigation. We modify VFH+ and use it as a subroutine to find a collision-free direction to an object. We first give a brief description on the modified version of VFH+ and then introduce the algorithms.

A. THE MODIFIED VFH+

The modified VFH+ computes a histogram to represent the graspable directions of an object in two steps. First, it finds obstacle-free angle ranges as seen in Fig. 2 (i.e., accessible directions in blue). Next, the modified VFH+ computes the graspable directions (in green as seen in Fig. 2) by computing the intersection between the accessible and the reachable directions. A robot can take one of the resultant graspable directions to grasp an object. We briefly summarize the computation of the accessible directions using VFH+ (The technical details are referred to [20]).

The key idea is to consider the environment in the perspective of a target object, which is the center circle in green as seen in Fig. 3a. In the coordinate system, the direction from the target object to the robot base is determined as the x-axis and 0°. From the configuration of objects, the modified VFH+ computes a polar histogram according to the positions and sizes of objects around the target. Considering the sizes of gripper and objects, we have the outline circle of each object enlarged for collision-checking as seen in Fig. 3a. A collision occurs, when the gripper is in the enlarged circles in red. The histogram range is determined by considering the sizes of enlarged circles. The overall histogram considering all objects is computed by the sum of the polar histogram of each object as seen in Fig. 3b. In Fig. 3b, there are three angle ranges in the polar histogram (i.e., -134°∼(-42°), -9°∼80°, 24°∼172°), where non-zero magnitudes of histogram are computed. A single obstacle occludes the target in some ranges so that the magnitude of histogram in the ranges is one. Multiple obstacles could block the target so that the magnitude of histogram is larger than one (e.g. it is two in the range of 24°∼80° in the figure). The angle range with zero magnitude means that there is no object blocking the target in the range. Due to geometrical limitations, the workspace of robot arm is bounded so that the sector of histogram is partially and practically available. For the example in Fig. 3c, the sector of angle ranges from -45°∼45° is the workspace such that the robot cannot reach outside that sector. The valid histogram H of the sector is computed as shown in Fig. 3d. From H, a set of accessible directions can be obtained. If the target object is reachable from any direction (like a cylindrical object), all the accessible directions become graspable directions. If the object has limited reachable directions (which is known as a priori), the graspable directions could change according to the pose of the object.

GraspabilityCheck determines the minimum magnitude of the histogram (H_G) out of the histogram (H) and its corresponding sector for a given object. The histogram (H)



(a) An example of a configuration (green solid circle: target, red solid circles and box: obstacles, enlarged outlines in red: safety check for obstacle)

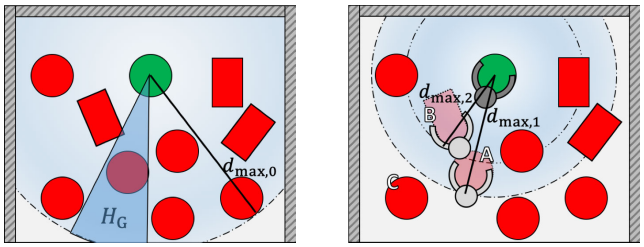
(b) The polar histogram of the configuration

(c) The normal histogram corresponding to (b)

(d) The partial histogram on the robot workspace (-45°∼45°)

FIGURE 3. An example of a configuration and its histogram from the modified VFH+: In (a), the enlarged outlines in red surrounding the objects denote the safety areas for collision-checking with the robot gripper. Through the histograms in (b) and (c),the partial histogram in (d) on the robot workspace is finally obtained.

for a given object is obtained by the modified VFH+ as mentioned before. The inputs of GraspabilityCheck are a configuration of objects O, a target object t, a robot configuration M (i.e., robot kinematics and position), and a parameter $d_{\max}$. The parameter $d_{\max}$ is used to determine the area for obstacle detection, which is described in Sec. III-B in detail. Within the reachable directions, the sector having the minimum magnitude of the histogram is obtained as H_G, which is the output of GraspabilityCheck. GraspabilityCheck is complete, because the histogram must have at least one minimum magnitude. There are two cases depending on the minimum magnitude of histogram: (i) If the magnitude is zero, there is an obstacle-free direction to the target. Thus, the algorithm can find graspable directions to grasp a target. (ii) If the magnitude is nonzero, there exists at least one obstacle blocking a target. For the example in Fig. 3d, the minimum magnitude of the histogram, H_G, is zero. This means that no obstacle blocks the target in the sector so that the target can have a graspable direction in that sector. If the target has a limitation on its reachability in the shape like a box, the sector could shrink more. A time complexity of GraspabilityCheck is linear. GraspabilityCheck calls the modified VFH+ once to compute a histogram H. The modified VFH+ runs on N objects at most in the order of $O(N)$ in [20]. Then GraspabilityCheck runs to find the graspable directions with the minimum magnitude of the histogram. The time complexity depends on the resolution of the histogram sector. If we divide



(a) The initial $d_{\max,0}$ to include all objects to compute a histogram by GRASPABILITYCHECK

(b) The updates of $d_{\max,0}$ to exclude obstacles that do not have to be considered anymore

FIGURE 4. An illustration $d_{\max}$ computation, which is updated after each removal of an obstacle. (a) Initially, $d_{\max,0}$ is the Euclidean distance between the target and the farthest object such that all the objects are considered in GraspabilityCheck to compute the histogram. (b) Once Obstacle A is removed, $d_{\max,1}$, which is the distance between the target and Obstacle A, is used by GraspabilityCheck to find the next obstacle to remove. Once Obstacle B is removed, $d_{\max,2}$ is used.

the sector into L segments (i.e., angle intervals), the time complexity of GraspabilityCheck is $O(N + L)$.[3]

B. LOCAL SEARCH ALGORITHM

LocalSearch algorithm in Alg. 1 aims to find the obstacles to remove one by one through iteration. If multiple obstacles block a target object, the algorithm recursively runs until it finds graspable obstacles by chaining obstacles from the one near to the target to the one that is graspable. Once the robot removes a graspable obstacle that is not the target, the algorithm is called to find the next obstacle for removing with the updated object configuration (O), which the current obstacle is excluded from. The algorithm is called repeatedly until the target t is removed from O.

The quantity $d_{\max}$ determines what obstacles are included for computing the histogram. Initially, $d_{\max}$ is given as the Euclidean distance from the target to the farthest obstacle from it. After each step (after removing an obstacle), $d_{\max}$ is updated as the Euclidean distance between the target and the obstacle that is most recently removed (line 6). By this update, the obstacles in other directions but the one the robot is currently "digging" can be excluded. Figs. 4 shows how to update $d_{\max}$. In the beginning, the Euclidean distance from the target to its farthest object in O is computed ($d_{\max,0}$ in Fig. 4a) so that all the objects in O are initially considered. Once Obstacle A is removed, the distance from the target to Obstacle A (i.e., $d_{\max,1}$) is used by GraspabilityCheck to find the next obstacle to remove (Fig. 4b). After removing Obstacle B, $d_{\max,2}$ is computed in the same way. If $d_{\max}$ is not updated as shown above, the robot could have removed more obstacles. For example, after removing Obstacle A in Fig. 4b, Obstacle C could be the next, if the algorithm keeps using $d_{\max,0}$. This is due to the fact that Obstacle C is still included for computing the histogram.

[3]In our experiments, the number of segments L is 90, when the angle range of total sector is 90° (i.e., the range of robot workspace) and the resolution of the sector is 1°.

Algorithm 1 LocalSearch

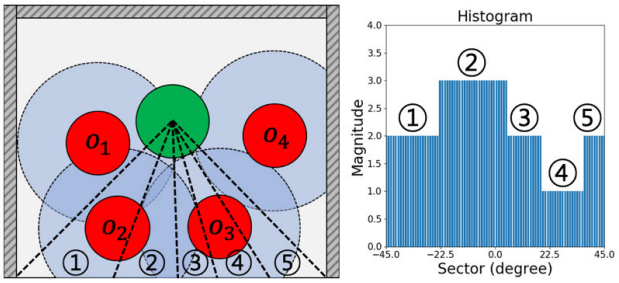
Input: The original target t, current target t_c (initial $t_c = t$), configuration O that is set of all objects remaining in the workspace including t, robot configuration M, distance from a target to a farthest object $d_{\max}$

Output: The set of an obstacle to be removed O_s, updated $d_{\max}$

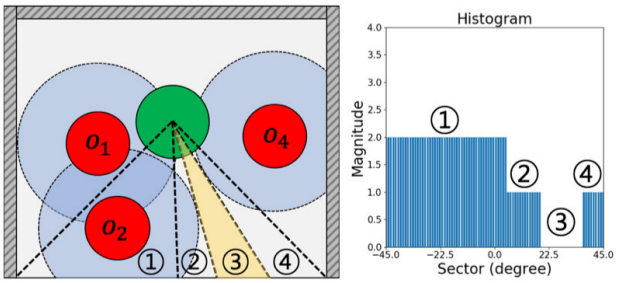
1: $O_s = \phi$
2: H_G = GraspabilityCheck($O, t_c, M, d_{\max}$)
3: **Choose** o_r which is in the sector of H_G and the closest to the target
4: **if** $o_r == t_c$ **then**
5: **if** MotionPlanning(o_r) == True **then**
6: $d_{\max}$ = EuclideanDist(o_r, t)
7: $O_s = \{o_r\}$
8: **return** $O_s, d_{\max}$
9: **else**
10: **Choose** other o_r which is in the sector of H_G and the next closest to the target
11: **end if**
12: **end if**
13: $t_c = o_r$
14: $(O_s, d_{\max})$ = LocalSearch($t, t_c, O, M, d_{\max}$)

The minimum magnitude of histogram H_G is computed by GraspabilityCheck in line 2. The magnitude of H_G represents the minimum number of obstacles to be removed in the direction angles of H_G. Therefore, the robot can remove the minimum number of obstacles. If there are multiple obstacles in the sector of H_G, the algorithm chains the obstacles from the closest to the target to the farthest. The chain of obstacles begins from the target, since the histogram indicates which obstacle blocks the target. Thus, the closest obstacle o_r is chosen for removal in line 3. If the chosen obstacle is not the current target t_c, it means other obstacles still remains to be chained. Thus, the algorithm recursively finds the next obstacle in the chain by considering o_r as a temporary target (it denotes the current target t_c lines 13–14). If the obstacle o_r is the current target t_c, it means the magnitude of H_G is zero. It is then evaluated whether a motion planning to grasp o_r is feasible. If the motion planning is feasible, o_r is graspable and the set of O_s with a single element of o_r is returned (lines 5–8). When the motion planning is infeasible, a new o_r that is the next closest to the target in the sector of H_G is selected (line 10). This recursion occurs until the algorithm finds a graspable obstacle, which is at the end of the chain. Once the graspable obstacle is removed, Alg. 1 is called to find the next obstacle until the original target (t) becomes graspable. Due to this, the algorithm solves a new removal problem after removing an obstacle, which could be favorable to dealing with such dynamic situations as the poses of objects change or new objects appear.

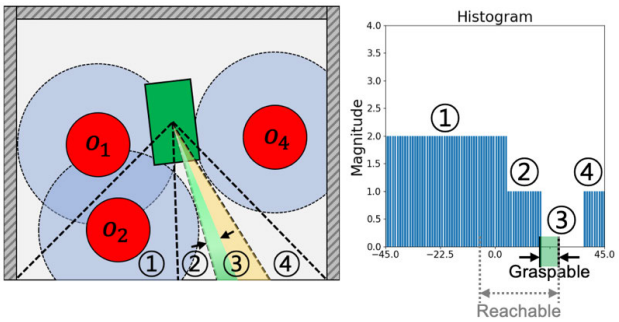
Three examples of the algorithm are shown in Figs. 5. In Fig. 5a, a sector range of ④ has the minimum magnitude



(a) The case where no obstacle-free direction exists: The histogram on the right does have no sectors with zero magnitude. Thus, the end-effector cannot grasp the target without removing obstacles.



(b) The case where there is an obstacle-free direction: The histogram on the right shows the sector of ③ with zero magnitude. Therefore, the end-effector can grasp the target directly through the sector of ③.



(c) The case where the target has limited reachable direction angles: The intersection of the accessible directions (the sector ③) and the reachable directions denotes the graspable directions as seen in the right figure.

FIGURE 5. Examples illustrating the histograms (in the right figures) computed by the modified VFH+ for the cluttered configurations on the left. The numbers in circles on the left represent areas in the configurations, which matches the same numbers in the histograms on the right. The positive angle of the histogram is measured counterclockwise from the base line, which is vertically from the target in green to the robot base (just under the bottom line of the figures on the left). The histograms on the right are shown partially from −45° to 45°, which is the workspace of robot manipulator.

for the histogram H_G. Within the sectors, the obstacle o_3 at the bottom is chosen to remove. In the example of Fig. 5b, the sector ③ has the zero magnitude of H_G, which means that the target has graspable directions in that sector. If only the target has limited reachable directions in the same configuration, the graspable directions could shrink as seen in Fig. 5c (graspable directions in green on the left and the corresponding histogram on the right).

It is noticed that Alg. 1 is complete and runs in polynomial time as shown in the following theorems.

Theorem 1: Alg. 1 is complete and a robot manipulator eventually grasps a target.

Proof: By Sec. III-A, GraspabilityCheck is complete so that it always returns an object to be grasped (either the target or an obstacle). Therefore, Alg. 1 can always find a graspable object returned by GraspabilityCheck. In each recursive call of Alg. 1, one object is removed. After the recursion is repeated for all $N-1$ objects in the worst case (i.e., all obstacles are blocking the target so removed), the last remaining object, which is the target, is grasped. Thus, Alg. 1 guarantees that the target is grasped. □

Theorem 2: Alg. 1 runs in polynomial time.

Proof: Initially, Alg. 1 needs the maximum among all the Euclidean distances from the target to all obstacles. The maximum operation takes $O(N)$ but only once. Then, Alg. 1 is called recursively at the most N times for all N objects in O. The subroutine GraspabilityCheck is with $O(N+L)$ as proven in Sec. III-A. Therefore, the time complexity of Alg. 1 is $O(N+N(N+L)) = O(N^2+NL)$. □

C. GLOBAL SEARCH ALGORITHM

Alg. 1 runs in polynomial time and could take relatively low time cost to find obstacles to remove. However, the solution may be suboptimal in the number of removed obstacles as the algorithm chains obstacles locally without considering global optimum. Thus, we develop a global search algorithm using a tree search method.

We define a tree to search a sequence of obstacles to be removed. As shown in Fig. 6, a node in the tree represents a state of objects (i.e., an object configuration). A goal node represents a configuration, where the target has graspable directions without collision. If object o is removed from the current configuration O, a child node is generated to represent the configuration where o does not exist anymore, i.e., $O \setminus o$. Thus, the depth of the tree denotes the number of removed obstacles. A tree search algorithm traverses over the tree structure until the goal node is found. In each node, multiple child nodes can be generated, since multiple objects need to be removed. A frontier stores the nodes that will be searched while expanding the tree. Among the nodes in the frontier, the next node is determined by the search strategy for expansion.

We should employ one of the uninformed search strategies, as we do not have any information regarding the goal state so that evaluating the expansion of the tree is not possible. We choose the Breadth-First Search (BFS) to find the optimal solution among many possible ones. BFS searches all the nodes in the same depth and moves to the next depth. If a goal node is found by BFS, the node is at the shallowest depth compared to all the other solutions [21]. In our problem, a deeper depth means removing more obstacles such that the goal at the shallowest depth indicates the minimum number of removed obstacles.

However, BFS has exponential time complexity so that it may run slow to find a goal node. To reduce the practical runtime, we use a pruning and an ordering method. We prune the tree, if a node to be generated has the same state with one of existing nodes. On the other hand, the order of nodes in the frontier could affect the search efficiency significantly.

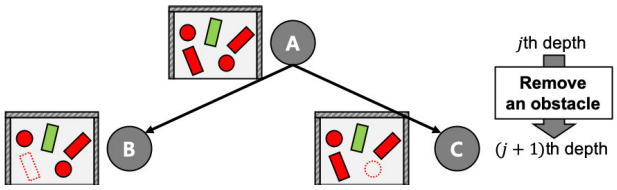


FIGURE 6. The concept of a tree structure: Each node represents an object configuration. A child node is generated by removing an object from the configuration. The depth of tree denotes the number of obstacles to be removed.

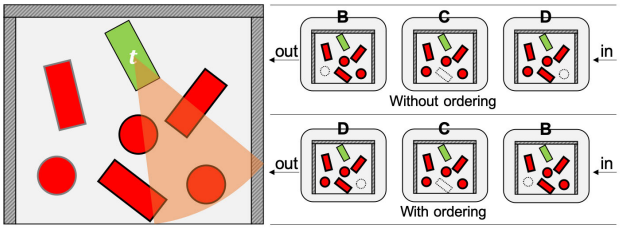


FIGURE 7. An example of ordering the frontier: The fan shape in the left figure shows the reachable directions of the target in green. The first row of the right figure shows the frontier without ordering for the configuration of objects in the left figure. The nodes are searched in the order of B, C, and D. We consider the reachable directions of the target for ordering. Since the end-effector can approach an object through its reachable directions, it is more important to search first the objects within the directions. After ordering, the algorithm searches D–C–B as seen in the bottom row of the right figure.

Since BFS explores all the nodes in the same depth, it could work faster if the nodes that are most likely the goal are searched first [22]. After every expansion of the tree, we order the nodes in the frontier (implemented by a First-In-First-Out queue) to remove the obstacle that likely clears the path to the target. The details of the pruning and the ordering method are described below with the pseudocode shown in Alg. 2.

Alg. 2 has a similar input and output with Alg. 1. Since Alg. 1 works in an online manner, it is called repeatedly to remove objects (obstacles) one by one and grasp a target. Thus, Alg. 1 needs to return $d_{\max}$ for the next call in addition to the set O_s indicating which object to remove. Since Alg. 2 finds the full sequence of the objects to be removed, $d_{\max}$ is used and updated inside the algorithm. In addition, an initial node v_0 is an additional input argument to Alg. 2.

In the beginning, the frontiers Q and U and the set of obstacles to be removed O_s are initialized. Then the root node v_0 and the initial $d_{\max}$ are inserted into the frontiers of Q and U. Before the target is grasped, the following loop repeats (lines 4–34). First, the node v and $d_{\max}$ to be expanded are dequeued (line 5-6). For each object in O in the state v, GraspabilityCheck checks if each of the objects has graspable directions (line 8). If an object has such graspable directions (i.e., H_G has zero magnitude), the object can be removed. Thus, a new node v^+ is generated and saved to the frontier (lines 10–11). The quantity $d^+_{\max}$ is obtained in the same way as done in Alg. 1 for the next computation of GraspabilityCheck. Pruning is done in line 32. When a tree is generated, multiple nodes representing an identical state can be generated. For example, if two graspable objects are removed in different orders, they result in the same state. Duplicating of nodes may cause unnecessary searching efforts and make the algorithm slow. To check the duplication in node generation, we compare all the nodes in the same depth. Since the nodes in other depths have the different numbers of removing objects, those nodes never share the same state.

In line 33, the nodes in the frontier are ordered. As described before, the frontier stores the nodes to search, which are generated by removing an obstacle. Thus, the nodes in the frontier represent the state where each of graspable objects is removed. In Fig. 7, the top row of the right figure shows the frontier without ordering for the configuration of objects on the left. The nodes are searched in the order of B, C, and D. To order the nodes, we consider the reachable directions of the target. Since the end-effector approaches an object in its reachable directions, it is important to consider the objects within the directions first. In each expansion of the tree, we find the reachable directions of the target. In the left figure of Fig. 7, the reachable directions of target are represented in the fan shape. The first node into the frontier represents the configuration, where the corresponding obstacle occludes the target most and is selected for removal (determined by the histogram). The next nodes are determined according to the same manner (i.e., dependent on how much the corresponding obstacle occludes the target). The nodes outside the reachable directions are then inserted into the frontier by the lexicographical order. The bottom row of the right figure in Fig. 7 shows the ordered frontier. The nodes are searched in the order of D, C, and B after ordering.

Once a child node is generated, the algorithm checks if the node is a goal state. In other words, if the corresponding object to be removed is not the target, the same process runs to find another object to remove. If the object to be removed is the target, the algorithm saves the sequence of all the objects(i.e. obstacles) to be removed so far. This is done by chaining the parents nodes back to the root (lines 16–21). The objects in the chain are saved to O_s in the reverse order as seen in line 22, (i.e., the order of obstacles from the robot to the target). Then, the algorithm evaluates whether the motion planning for the set O_s is feasible. If it is feasible, the algorithm terminates and returns the set O_s (lines 23–24). If not feasible, O_s becomes empty, and the last elements of frontier Q and U are deleted (lines 25–27). And replanning proceeds until a new goal node is found. Fig. 8 shows an execution example of Alg. 2. In the example, a goal node is found at the depth level 3 so that the two objects as obstacles are removed and the target is grasped in the node order of B–D–G.

It is noted that Alg. 2 is complete and runs in exponential time as shown in the following a theorem.

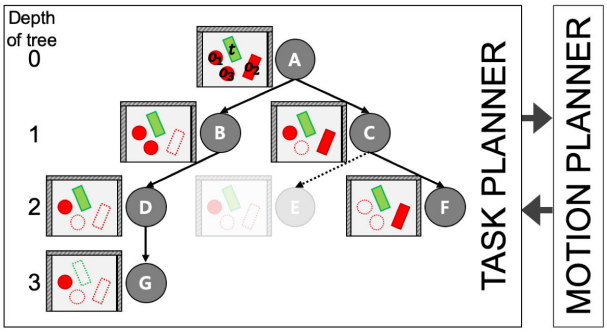


FIGURE 8. An execution example of Alg. 2: In the beginning (Node A), the obstacles to be removed are found by GraspabilityCheck. Since o_2 and o_3 are removable and o_1 and t are not graspable in Node A, two nodes B and C are generated from Node A. Their order is determined by the ordering method described above. Since o_2 blocks more widely the reachable directions to the target than o_3, it is searched earlier. After the two nodes are generated from the root node, the search goes to the next depth because of no more obstacles to be removed in the current depth. It is noticed that Node E is pruned in the next depth, since it is identical to Node D. Finally, the target is grasped in the bottom depth. If motion planning for nodes B–D–G is feasible, the sequence is returned.

Theorem 3: Alg. 2 is complete so that it eventually gives a plan to grasp a target object.

Proof: By Sec. III-A, GraspabilityCheck is complete so it always returns an object to be grasped (either a target or an obstacle). Alg. 2 always expands the search tree by removing the object indicated by GraspabilityCheck. Since we use BFS that is shown to be complete, the solution is always found. Thus, Alg. 2 guarantees a sequence of objects(obstacles) to be removed including a target. □

D. HEURISTIC SEARCH

The Breadth-First Search (BFS) in Alg. 2 determines a node v in the first-in-first-out (FIFO) order from Q in dequeue as in [21] (line 5-6). Instead we may apply the heuristic search proposed in [23] to dequeue and select a node by employing a heuristic cost for efficiency as in [22]. The cost for each node is defined as the sum of two H^t_{G} values of the current node and its parent node. H^t_{G} is H_{G} value of the original target t, which is computed from GraspabilityCheck at each node. The heuristic search determines a node with a minimum cost among the nodes in the frontier since the H^t_{G} of the goal node is 0. The simulation results of Alg. 2 using BFS and the heuristic search are compared and investigated in Sec. IV-A4.

Algorithm 2 GlobalSearch

Input: The original target t, robot configuration M, distance from the target t to its farthest object $d_{\max,0}$, node v_0 representing the initial configuration that has the set of all objects in the workspace including t

Output: O_s that includes the obstacles to be removed and the target t

1: $O_s = \phi$; $Q = \phi$; $U = \phi$
2: Enqueue(Q, v_0)
3: Enqueue(U, $d_{\max,0}$)
4: **while** $t \notin O_s$ **do**
5: v = Dequeue(Q)
6: $d_{\max}$ = Dequeue(U)
7: **for** each $o \in v$ **do**
8: H_{G} = GraspabilityCheck(v, o, M, $d_{\max}$)
9: **if** $H_{\mathrm{G}} == 0$ **then**
10: **Generate** node v^+ which represents v except o
11: Enqueue(Q, v^+)
12: $d^+_{\max}$ = EuclideanDist(o, t)
13: Enqueue(U, $d^+_{\max}$)
14: **if** $o == t$ **then**
15: $x = v^+$
16: **while** $x \neq v_0$ **do**
17: v_p = GetParent(x)
18: $o' = O_p \setminus O_x$
19: $O_s = O_s \cup \{o'\}$
20: $x = v_p$
21: **end while**
22: O_s = ReverseOrder(O_s)
23: **if** MotionPlanning(O_s) == True **then**
24: **return** O_s
25: **else**
26: $O_s = \phi$
27: **Delete** the last elements of Q and U
28: **end if**
29: **end if**
30: **end if**
31: **end for**
32: (Q, U) = Pruning(Q, U)
33: (Q, U) = OrderFrontier(Q, U)
34: **end while**

IV. EXPERIMENTS

We run simulations and practical experiments to validate our algorithm. In the simulations, we verify the characteristics of each algorithm and compare the proposed algorithms with another algorithm. In the practical experiments, we show that the proposed algorithms could be applicable to real environments.

A. SIMULATIONS

We run numerical simulation in a virtual environment using the dynamic simulator V-REP [24]. In the simulation, we use Kinova JACO1, a 6-DOF manipulator anchored at the base location. The simulation instances of the problem are composed of cluttered and constrained environments as seen in Fig. 9. In the environments, three types of objects which are cylinder, cup and box are used. Considering a typical two-finger gripper as we used, we set the reachable directions of each object to 360° for a cylinder (all surface), 60° for a cup (round surface except the handle part) and 30° for a box (two narrow sides). It is not assumed that the manipulator can approach objects from the top. Obstacles are supposed to be placed on the predefined spots of a shelf next to the robot.

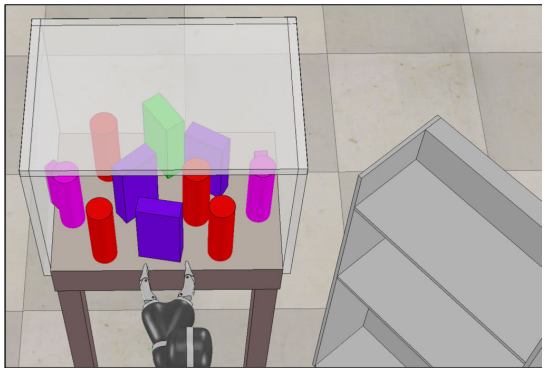


FIGURE 9. An instance of a simulated environment with V-REP. The shapes of objects are cylinder (red), cup (magenta), and box (purple). The target object is the green box.

All the simulations were run on an Intel i7-7700 3.6 GHz with 16 GB memory.

1) RUNTIME OF GlobalSearch

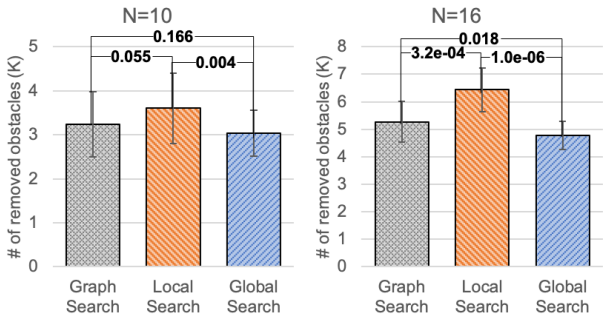
First, we investigate the runtime of Alg. 2. The algorithm is implemented in python2.7 and Moveit! [25] and RRTconnect [26] of Open Motion Planning Library (OMPL) [27] are used for motion planning. The algorithm is performed with the 30 random object configurations for each case of the varying numbers of objects $N = 7, 10, 13$ and 16 as seen in TABLE 2. The tree search time increases in proportion to the number of objects N. The motion planning time is also proportional to the tree search time, since the number times of motion planning is in proportion to the depth of the tree. As the number of objects N increases, the probability of failure in motion planning increases. This failure causes more searching efforts to find a node where motion planning is valid such that the runtime of algorithm increases. The reasonable computation time may be obtained by pruning and ordering efficiently. Since this planning is done in advance, no other computation is required during robot's execution. In a practical application, the number of objects would not be prohibitively large, since an environment is bounded, dense, and confined. Therefore, Alg. 2 can be applicable to real-world instances.

TABLE 2. The runtime of GlobalSearch. The tree search time and the runtime with motion planning increase in proportion to the number of objects N.

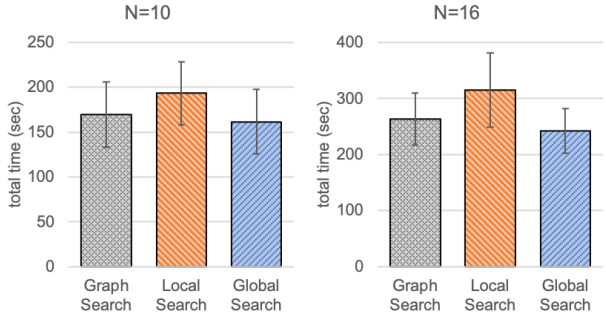
N (the number of objects)	7	10	13	16
practical tree search time (s)	0.033	0.425	1.558	4.810
including motion planning (s)	2.105	6.021	6.548	10.320

2) COMPARING OF LocalSearch AND GlobalSearch

LocalSearch basically uses the same method as in [20] to determine which object to be relocated. The method in [20] was compared with other previous works in [2] and [28], providing the fewer number of removed obstacles. In this paper, we compare the algorithms of LocalSearch and GlobalSearch



(a) The average K for $N = 10, 16$. The values above the graph are the P-values of the t-test.



(b) The total time for $N = 10, 16$

FIGURE 10. Simulation results: (a) The average number of removed obstacles (K) for $N = 10, 16$. (b) The average total time (in seconds) for $N = 10, 16$.

with a graph search method of [29]. The graph search method finds an obstacles sequence by creating paths between adjacent objects. We test the environments with the number of objects $N = 10$ and 16 and generate 30 random instances for each. The objects are placed randomly on a $0.6(m) \times 0.4(m)$ planar area. The results are summarized in Figs. 10. The number of removed obstacles (K) from GlobalSearch is always same as or smaller than one from LocalSearch and the graph search method. Since the graph search method is a global searcher but has fewer search cases than GlobalSearch, results of the method are less than LocalSearch but more than GlobalSearch. The difference in K among the methods becomes larger as N grows, because LocalSearch and the graph search method have more chances to meet local minima if there are more objects. In Fig. 10a, the number above the graph is the P-value of the t-test. When $P < 0.05$, results of two methods are interpreted as statistically different. At $N = 10$, the results of LocalSearch and GlobalSearch are statistically different. At $N = 16$, there is a statistically significant difference between the results of the three methods. On the other hand, GlobalSearch has longer planning time and the algorithm needs to search again when motion planning is infeasible. However, the total time, which denotes the sum of planning time and robot execution time, for GlobalSearch is smaller than other methods as seen in Fig. 10b. The reduction comes from finding the global solutions, which have smaller K so that the execution of relocating the K objects takes less time. On average, GlobalSearch reduces 24% in the number of

TABLE 3. Algorithm runtime for the successful tasks.

(a) The success rate of each algorithm

	N (the number of objects)	7	10	13	16
Success rate (%)	method in [20]	100	93	73	60
	LocalSearch	100	100	100	100

(b) Algorithm runtime for the successful tasks

	N	7	10	13	16
Runtime (s)	method in [20]	0.038	0.045	0.056	0.067
	LocalSearch	2.357	3.445	4.951	6.167

removed obstacles such that 23% in the total time is reduced, comparing with LocalSearch.

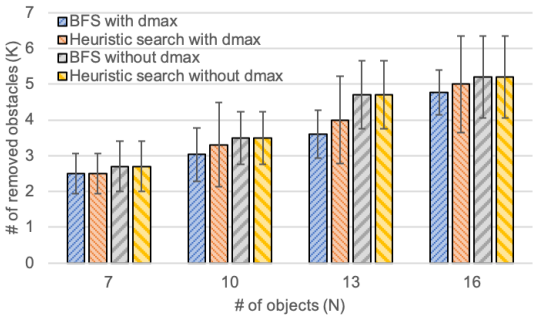
3) PERFORMANCE OF LocalSearch WITH MOTION PLANNING

LocalSearch is based on the method in [20] and is integrated by adding motion planning in this paper so that some of the searched results might fail in motion planning. The motion planning is employed to avoid collisions so that LocalSearch is the advanced version of the method in [20]. We compare the results of the two algorithms in this section. For balanced comparison, the method in [20] is modified to consider the reachable directions of the object.

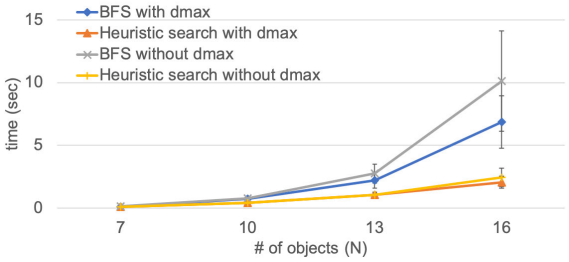
The success rate of task execution and the algorithm runtime are compared in TABLE 3. The simulation was conducted 30 times for each case of $N = 7, 10, 13$ and 16. If any other object but the removing obstacles falls down due to collision while a robot executes the task of grasping a target, the task fails. The algorithm runtime was measured only for the successful cases and is summarized in (b) of TABLE 3. When the task is successfully executed (i.e., no collision in motion planning), the number of obstacles to remove is same for both algorithms. This is because the two methods use the same method to determine which object to be relocated. The algorithm runtime of LocalSearch is 82 times longer than that of the method in [20] on average because of the computing time for motion planning. The task executions always succeed in LocalSearch, whereas the failures of the method in [20] happens more frequently as the number of objects increase. In LocalSearch the object to be removed is determined by considering only the gripper and motion planning then determines whether the object can be grasped without full-link collision. When motion planning fails, LocalSearch searches another object. The success rate of the method in [20] decreases for denser environments, because full-link collision is not checked.

4) GLOBALSEARCH WITH BFS AND HEURISTIC SEARCH

Finally, we compare GlobalSearch using the Breadth-First Search (BFS) and the heuristic search. In addition, we compare the algorithms with and without $d_{\max}$. As described in Sec. III-D, the heuristic search employs a heuristic cost, which is defined as the sum of the minimum magnitudes of histogram at the current node and its parent node such that it may search less number of nodes compared with using BFS. This may cause reduction in searching time.



(a) The average number of removed obstacles (the average K) of BFS and the heuristic search



(b) The algorithm runtime of BFS and the heuristic search

FIGURE 11. Simulation results for BFS and the heuristic search.

FIGURE 12. An instance of a real environment for experiments and a vision view from the vision sensor mounted on the wrist.

The simulation results in 30 random configurations for each N are summarized in Figs. 11 ($N = 7, 10, 13$, and 16). For both of BFS and the heuristic search, $d_{\max}$ limits the area around a target and the objects only within that area are involved to compute the histogram. Due to this, using $d_{\max}$ could reduce the average K in Fig. 11a and the algorithm runtime in Fig. 11b for the both searches, compared to the results without using $d_{\max}$. BFS with $d_{\max}$ removed the fewest number of objects on average as in Fig. 11a. For the case of 16 objects, the heuristic search with $d_{\max}$ reduces 70% in the algorithm runtime compared to BFS with $d_{\max}$.

When $d_{\max}$ is not applied, the average K's of BFS and the heuristic search are same in Fig. 11a. The heuristic cost (H^{t}_{G}) without $d_{\max}$ decreases by 1 per the depth of tree because a node is generated for removing each object from outside to inside in the workspace. This may cause the same average K's for both searches. When using $d_{\max}$, some

(a) The task with LocalSearch. The number of removed obstacles (K) is 2.

(b) The task with GlobalSearch. The number of removed obstacles (K) is 1.

FIGURE 13. **The task execution with $N = 8$. A target object is a blue box (diget box).**

nodes, where H^t_G decreases by more than one per the depth of tree, can be generated. This can reduce the number of objects and the histogram. In some cases, BFS can reach a node, where the histogram magnitude converges to 0 (i.e., the target is graspable) earlier than the heuristic search, even any node at the same depth in the sequence does not have a minimum H^t_G. Therefore, BFS may have the fewer average K than the heuristic search. Since the number of generated nodes in heuristic search is smaller than in BFS, the heuristic search took less time as in Fig. 11b.

B. PRACTICAL EXPERIMENTS

Experiments are conducted to validate the applicability of our two algorithms in a real environment. We use a Panda manipulator which has 7-DOF and a Robotiq 2F-85 gripper which has 2-fingers with 85*mm* stroke. The experiment instances consist of objects placed randomly on a shelf and a tub to place the objects as seen in Fig. 12. The objects are snack boxes which are different in shape. To obtain the poses and shapes of objects, we use an Intel realsense D435i as a vision sensor and Complex-YOLO [30] for object recognition. Complex-YOLO creates a 3D bounding box that represents the shape and pose of an object with a point cloud. The reachable direction of the object is determined by the stroke of the gripper and the bounding box. The vision sensor is attached to the wrist of the manipulator.

We test our algorithms with the number of objects $N = 5$, 8 and with 30 random instances for each N. Figs. 13 show the task process of each algorithm with $N = 8$. We measure the number of removed obstacles (K) and the success rate. The results of the experiments are summarized in TABLE. 4. GlobalSearch has a lower K compared to LocalSearch. As N increases, the difference in K between the two algorithms increases like the simulation results (Fig. 10a). However, The success rate is higher in LocalSearch than in GlobalSearch. The main cause of the task failure is the uncertainties of the objects' postures. The values of an object

TABLE 4. **The success rate of each algorithm.**

(a) The number of removed obstacles of each algorithm (standard deviation)

	N (the number of objects)	5	8
# of removed obstacles (K)	LocalSearch	1.43 (0.50)	2.27 (0.64)
	GlobalSearch	1.23 (0.43)	1.80 (0.55)

(b) The success rate of each algorithm

	N (the number of objects)	5	8
Success rate (%)	LocalSearch	93	83
	GlobalSearch	90	77

recognized by the vision sensor have errors in size or pose. In particular, the error of an object in the back, which is partly hidden from view, is higher than that of an object in the front. LocalSearch selects an object by using the newly updated poses of objects before grasping. On the other hand, GlobalSearch finds the order of obstacles based on the initially obtained values of objects. Therefore, GlobalSearch has a higher probability of failure to grasp objects during operation compared to LocalSearch.

V. CONCLUSION AND FUTURE WORKS

In this paper, we propose a global search and a local search algorithms to plan the rearrangement of objects in clutter to grasp a target. The two algorithms aim to determine as few obstacles to be removed as possible. Also, they consider the variable reachable directions of objects. The algorithms are evaluated with the varied number of objects and compared in realistic simulation environments.

The global search algorithm finds a global solution through a tree search method. The search strategy used runs in exponential time but its algorithm runtime is not prohibitive, as pruning and ordering are implemented to improve searching speed. Even with the moderate runtime (up to 10 seconds with 16 objects), the number of removed obstacles can be reduced so that the total time (the sum of algorithm runtime and robot execution time) can decrease. On the other hand, the local search algorithm performs faster than the

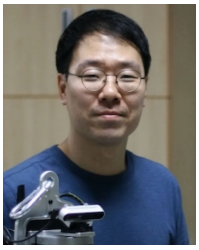
global search algorithm and can work online, which means it searches an object to be remove one by one. If an environment changes dynamically (i.e., a hidden object appears or object recognition is noisy so object poses change frequently as in the practical experiments), the local search algorithm could be more appropriate as it doesn't compute the entire sequence again but remove the next obstacle at each step. Also, the local search algorithm has a polynomial time complexity so it can be adaptive to the size of instances. It is shown that the both algorithms are complete and can guarantee to obtain a solution. For the global search, a heuristic cost was applied to reduce the number of searching nodes such that it could provide a solution close to the global optimum but in less time than using BFS.

To consider the uncertainties of sensing data in the real world, one of the future works may employ an error measurement experiment to estimate the size and posture of an object. This may help our planners to be more reliable in real applications. Another future work is to deal with partially observable environments. If a target or some objects are invisible due to the poses of a camera and other objects, it is necessary to find such objects effectively.

REFERENCES

[1] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2007, pp. 3327–3332.
[2] M. R. Dogar and S. S. Srinivasa, "A planning framework for non-prehensile manipulation under clutter and uncertainty," *Auto. Robots*, vol. 33, no. 3, pp. 217–236, Oct. 2012.
[3] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 639–646.
[4] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox, "6-DOF grasping for target-driven object manipulation in clutter," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 6232–6238.
[5] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox, "Contact-GraspNet: Efficient 6-DoF grasp generation in cluttered scenes," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 13438–13444.
[6] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, "Object rearrangement using learned implicit collision functions," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 6010–6017.
[7] M. Moll, L. Kavraki, and J. Rosell, "Randomized physics-based motion planning for grasping in cluttered and uncertain environments," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 712–719, Apr. 2018.
[8] R. Papallas and M. R. Dogar, "Non-prehensile manipulation in clutter with human-in-the-loop," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 6723–6729.
[9] K. Ren, L. E. Kavraki, and K. Hang, "Rearrangement-based manipulation via kinodynamic planning and dynamic planning horizons," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Aug. 2022.
[10] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *Robotics Research*. Cham, Switzerland: Springer, 2020, pp. 405–419.
[11] L. Pinto, A. Mandalika, B. Hou, and S. Srinivasa, "Sample-efficient learning of nonprehensile manipulation policies via physics-based informed state distributions," 2018, *arXiv:1810.10654*.
[12] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang, "Rearrangement with nonprehensile manipulation using deep reinforcement learning," 2018, *arXiv:1803.05752*.
[13] S. D. Han, N. M. Stiffler, A. Krontiris, K. E. Bekris, and J. Yu, "Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps," *Int. J. Robot. Res.*, vol. 37, nos. 13–14, pp. 1775–1795, Dec. 2018.
[14] A. Krontiris and K. E. Bekris, "Dealing with difficult instances of object rearrangement," in *Robotics: Science and Systems*, 2015.
[15] S. D. Han, N. M. Stiffler, K. E. Bekris, and J. Yu, "Efficient, high-quality stack rearrangement," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1608–1615, Jul. 2018.
[16] P. M. U. Eljuri, L. El Hafi, G. A. G. Ricardez, A. Taniguchi, and T. Taniguchi, "Neural network-based motion feasibility checker to validate instructions in rearrangement tasks before execution by robots," in *Proc. IEEE/SICE Int. Symp. Syst. Integr. (SII)*, Jan. 2022, pp. 1058–1063.
[17] R. Wang, K. Gao, D. Nakhimovich, J. Yu, and K. E. Bekris, "Uniform object rearrangement: From complete monotone primitives to efficient non-monotone informed search," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 6621–6627.
[18] W. Vega-Brown and N. Roy, "Asymptotically optimal planning under piecewise-analytic constraints," in *Proc. Workshop Algorithmic Found. Robot.*, 2016.
[19] I. Ulrich and J. Borenstein, "VFH+: Reliable obstacle avoidance for fast mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 1998, pp. 1572–1577.
[20] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, "Efficient obstacle rearrangement for object manipulation tasks in cluttered environments," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 183–189.
[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2009.
[22] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2016.
[23] J. Lee, C. Nam, J. Park, and C. Kim, "Tree search-based task and motion planning with prehensile and non-prehensile manipulation for obstacle rearrangement in clutter," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 8516–8522.
[24] M. Freese, S. Singh, F. Ozaki, and N. Matsuhira, "Virtual robot experimentation platform V-REP: A versatile 3D robot simulator," in *Proc. Int. Conf. Simulation, Modeling, Program. Auto. Robots*, 2010, pp. 51–62.
[25] S. Chitta, I. Sucan, and S. Cousins, "MoveIt! [ROS topics]," *IEEE Robot. Autom. Mag.*, vol. 19, no. 1, pp. 18–19, Mar. 2012.
[26] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. Millennium Conf. IEEE Int. Conf. Robot. Automation Symposia (ICRA)*, Apr. 2000, pp. 995–1001.
[27] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.
[28] F. Zacharias, C. Borst, and G. Hirzinger, "Bridging the gap between task planning and path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 4490–4495.
[29] C. Nam, S. H. Cheong, J. Lee, D. H. Kim, and C. Kim, "Fast and resilient manipulation planning for object retrieval in cluttered and confined environments," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1539–1552, Oct. 2021.
[30] M. Simony, S. Milzy, K. Amendey, and H.-M. Gross, "Complex-YOLO: An euler-region-proposal for real-time 3D object detection on point clouds," in *Proc. Eur. Conf. Comput. Vis. (ECCV) Workshops*, Sep. 2018, pp. 197–209.

JINHWI LEE (Student Member, IEEE) received the B.S. degree in mechanical engineering from Hanyang University, where he is currently pursuing the Ph.D. degree in mechanical engineering. He is also a Student Researcher with the Artificial Intelligent Robotics Center, Korea Institute of Science and Technology (KIST), Seoul, South Korea. His research interests include task and motion planning for a mobile robot.



CHANGJOO NAM (Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Korea University, Seoul, South Korea, in 2009 and 2011, respectively, and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 2016. He has been an Assistant Professor with the Department of Electronic Engineering, Sogang University, since 2021. From 2018 to 2021, he was a Senior Research Scientist with the Robotics and Media Institute, Korea Institute of Science and Technology (KIST), Seoul. He worked with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, as a Postdoctoral Fellow, from 2016 to 2018. His research interests include task and motion planning for multirobot coordination and robotic manipulation.

JONG HYEON PARK (Member, IEEE) received the B.S. degree in mechanical engineering from Seoul National University, Seoul, South Korea, in 1981, and the S.M. and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 1983 and 1991, respectively. Since 1992, he has been working at the School of Mechanical Engineering, Hanyang University, Seoul, where he is currently a Professor. He was a Visiting Researcher at Waseda University, Tokyo, Japan, in 1999, as a part of the Korean Science and Engineering Foundation-Japan Society for the Promotion of Science Program, a KOSEF-CNR Visiting Researcher at the Scuola Superiore Sant'Anna, Pisa, Italy, in 2000, a Visiting Scholar at MIT, from 2002 to 2003, a Visiting Scholar at Purdue University, West Lafayette, IN, USA, from 2008 to 2010, and a Visiting Scholar at the University of Stuttgart, Germany, in 2019. He was also associated with Brooks Automation Inc., Chelmsford, MA, USA, from 1991 to 1992, and from 2001 to 2002. From 2010 to 2017, he was served as the Senior Editor for the *Journal of Mechanical Science and Technology*. His research interests include biped robots, robot dynamics and control, haptics, and biorobots. He is also a member of the Korean Society of Mechanical Engineers, the Korean Society of Automotive Engineers, Korean Society of Precision Engineering, and Institute of Control, Robotics, and Systems.

CHANGHWAN KIM (Member, IEEE) received the B.S. degree in mechanical engineering and the M.S. degree in machine design engineering from Hanyang University, Seoul, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree in mechanical engineering from The University of Iowa, Iowa City, IA, USA, in 2002. From 2002 to 2004, he was a Research Associate with the Robotics and Automation Laboratory, University of Notre Dame, Notre Dame, IN, USA. Since 2004, he has been working at the Korea Institute of Science and Technology (KIST), Seoul. His research interests include task and motion planning for robot manipulation and social robots.