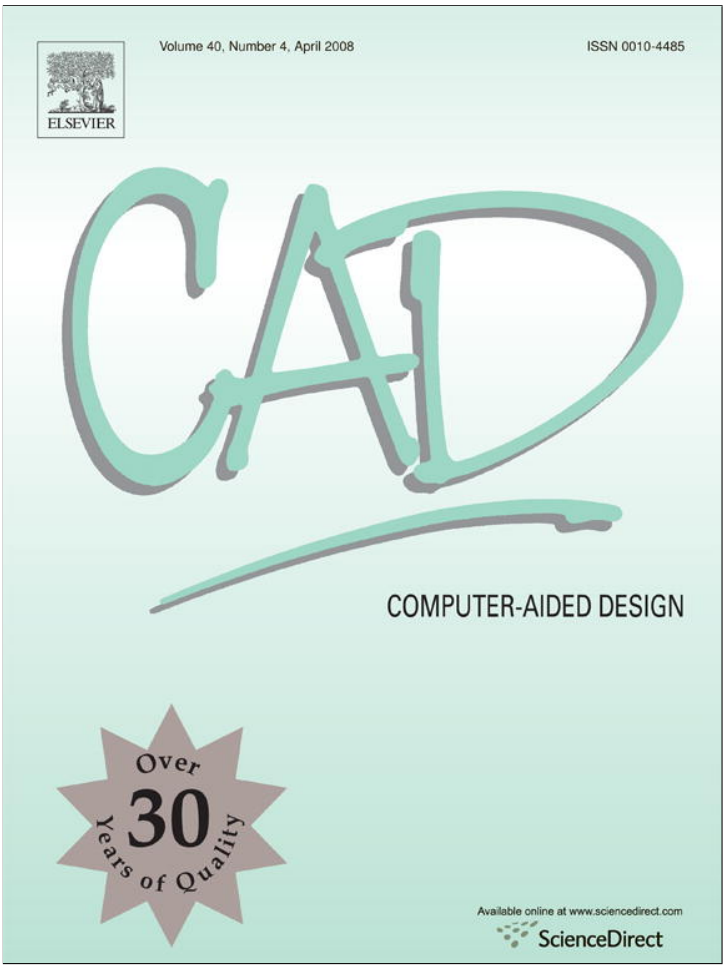


Available online at www.sciencedirect.com
ScienceDirect
Computer-Aided Design 40 (2008) 422–438
COMPUTER-AIDED DESIGN
www.elsevier.com/locate/cad

Visibility-based spatial reasoning for object manipulation in cluttered environments

Han-Young Jang[a], Hadi Moradi[b], Phuoc Le Minh[c], Sukhan Lee[c], JungHyun Han[a,*]

[a] College of Information and Communications, Korea University, Seoul, 136-701, Republic of Korea
[b] Department of Computer Science, University of Southern California, LA, CA, USA
[c] Intelligent Systems Research Center, Sungkyunkwan University, Suwon, Republic of Korea

Received 12 December 2006; accepted 10 December 2007

Abstract

In this paper, we present visibility-based spatial reasoning techniques for real-time object manipulation in cluttered environments. When a robot is requested to manipulate an object, a collision-free path should be determined to access, grasp, and move the target object. This often requires processing of time-consuming motion planning routines, making real-time object manipulation difficult or infeasible, especially in a robot with a high DOF and/or in a highly cluttered environment. This paper places special emphasis on developing real-time motion planning, in particular, for accessing and removing an object in a cluttered workspace, as a local planner that can be integrated with a general motion planner for improved overall efficiency. In the proposed approach, the access direction of the object to grasp is determined through visibility query, and the removal direction to retrieve the object grasped by the gripper is computed using an environment map. The experimental results demonstrate that the proposed approach, when implemented by graphics hardware, is fast and robust enough to manipulate 3D objects in real-time applications.

Keywords: Object manipulation; Visibility; Accessibility; Spatial reasoning; Graphics hardware

1. Introduction

Recently, a wide spectrum of effort has been expended in order to extend the robotics technology beyond industrial applications. Examples include mobile robots in the realm of service and personal assistance, especially for aiding the elderly or the disabled. A typical task of such a service or assistant robot is to manipulate objects of daily necessities on the user's request. Note that *object manipulation* comprises several subtasks, such as approaching, grasping, removing and delivering, each of which appears to be highly involved. In general, it is extremely difficult for a robot to achieve full autonomy of object manipulation in a home or service environment. This is because, unlike an industrial environment, a home or service environment is often unstructured, may not be kept under control, and thus, may not remain static. Furthermore, object manipulation in the context of service to humans needs to be performed in real time.

Traditionally, object manipulation has been part of *motion planning*. Unfortunately, the complex nature of motion planning for high DOF (degrees of freedom) robots often makes it impossible to achieve real-time performance [1]. For instance, the generation of a mobile manipulation path for a humanoid robot to grasp an object may require a few minutes [2]. Even for a single arm (typically of 6 or 7 DOF) with no mobile platform, real-time motion planning has been a challenging task [3]. It becomes much harder, if not intractable, for motion planning to be done in highly cluttered environments. The direct application of conventional motion planning to a grasp point selected on a trial-and-error basis would be too costly and prone to failure if the selected grasp point of the requested object does not turn out to be collision-free. If the grasp point is not reachable due to the obstacles in the neighborhood of the object, the robot requires further time-consuming motion planning, with a new grasp point until it succeeds. It would be much better to have a local but fast motion planner that can simultaneously determine a grasp point and local paths to access the grasp point and remove the grasped object, such that the local motion planner can be well integrated with a general motion planner for improved overall efficiency.

This paper proposes real-time techniques for providing such a local motion plan for the cluttered neighborhood of

* Corresponding author. Tel.: +82 2 3290 3207; fax: +82 2 927 3207.
E-mail address: jhan@korea.ac.kr (J. Han).

0010-4485/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2007.12.004

a requested object, and leaving the rest to a general motion planner. Specifically, this paper focuses on the visibility-based spatial reasoning techniques, in order to determine the *gripper access directions* and *object removal directions*, i.e. the directions along which the robot gripper can access and remove the requested object. The real-time performance is achieved using visibility query and environment map, which are supported by commodity graphics hardware.

The organization of this paper is as follows. Section 2 introduces related work. Section 3 presents object recognition and 3D workspace modeling. Section 4 overviews the proposed approach in the context of motion planning. The major contributions of this paper are presented in Section 5, which discusses the gripper accessibility analysis, and Section 6, which discusses the technique for determining the object removal directions. Section 7 discusses the integration of the proposed approach with general motion planning. Section 8 presents the experimental results and evaluates the performance of the proposed approaches. Finally, Section 9 concludes this paper.

2. Related work

The visibility-based spatial reasoning techniques presented in this paper complement general motion planning, and eventually enable the motion planner to achieve real-time performance in cluttered environments. In the motion planning field, probabilistic/randomized methods have been targeted to generate paths with high probability for high-dimensional configuration spaces, but may not be able to generate paths in real time. For instance, Kuffner et al. [2] used RRT-connect (Rapidly-exploring Randomized Trees) to generate paths for a humanoid robot. In their experiments, a typical grasping operation took between a few seconds to a few minutes to generate a path, on a 900 MHz Pentium 3 running Linux. Even for a single arm (typically of 6 or 7 DOF), motion planning presents a computational challenge due to the dimensionality of the search space. For example, Hsu et al. [3] analyzed the performance of motion planning for robots with different DOF, and showed that it may take a few seconds for motion planning for a 6- or 7-DOF robot. In addition, the probabilistic approaches have a difficulty in generating a path in cluttered environments due to the narrow passage problem [4] even though they usually work better than many alternatives such as the elastic strip method, which is discussed later. There have been many efforts to solve or reduce this problem [4–6] but still there is no guarantee to do motion planning in real time in cluttered environments.

Brook proposed the elastic strip method, which is another motion planning approach, promising to provide smooth motion and real-time performance [7]. In this approach, the potential field generated by the obstacles in the environments is applied for the robot to naturally move the robot away from the obstacles. At the same time, an elastic strip, stretched from the current pose to the goal pose, provides the potential necessary to move the robot to the goal. Finally, the task potential is applied to achieve the task of grasping and removing an object. Unfortunately, this method inherits the shortcomings of the potential field approach, i.e. being trapped in local minima. This is a serious problem in cluttered environments in which the chance of being trapped in local minima or oscillation is very high. In such cases, escaping from local minima would take a long time or may not be successful [8].

There is another issue in the elastic strip method, where motion planning may be performed given a grasp point on an object, without knowing whether it is actually reachable. In case the given grasp point is not reachable, due to the obstacles in the neighborhood of the object, the robot should attempt other grasp points until finding a reachable one. This trial-and-error approach would take a long time if each individual trial is time-consuming. In the probabilistic method, multiple grasp points can be used in the query phase, and connected to an already-built roadmap. In other words, after constructing the roadmap in the learning phase, multiple candidate grasp points are added to the configuration space in the query phase such that the algorithm continues to connect these points to the roadmap. The immediate advantage of this approach, over the elastic strip method, is that there is no need to recalculate/reconstruct the initial roadmap for different grasp points. In cluttered environments, however, generation of the initial roadmap may be time-consuming, and the process of connecting all the new points to the roadmap often fails, due to the narrow passage problem. On the other hand, it would be possible to avoid the costly process of connecting all grasp points to the roadmap by simply relying on the solutions based on the connectivity of a few grasp points to the roadmap. However, this simplification may result in overseeing better grasp points.

This paper presents an accessibility analysis technique for computing gripper access directions. In general, accessibility analysis refers to spatial reasoning activity that seeks to determine the directions along which a tool can access an object. The major work in accessibility analysis has been done in the inspection field, with particular significance to the coordinate measuring machines (CMMs) [9]. However, the application fields also include tool path planning for assembly [10], sensor placement for computer vision [11], numerically controlled machining [12], etc. In these fields, the proposed algorithms run mostly off-line to determine the accessibility.

Spyridi and Requicha [9,13] were the first to incorporate a systematic accessibility analysis for inspecting features. They use a computationally intensive method to determine first if a point is locally accessible, and then consider the entire workpiece for verification. Lim and Menq proposed an accessibility analysis approach, for an infinite length probe based on a ray tracing algorithm [14]. They determine a discrete 3D accessibility cone which is transformed into a 2D map in which only the orientation of the probe is expressed by two angles in a spherical coordinate system. A heuristic is used to determine the optimal probe direction for a set of inspection points. Limeiam and ElMaraghy [15] addressed accessibility analysis of a point in 3D space using elementary solid modeling operations: intersection, translation and scaling. The accessibility analysis research report most relevant to our

efforts can be found in [16], where the accessible directions for CMM tactile probes are computed using a cube map [17]. In this approach, however, it is assumed that the probe's path to the workpiece is completely clear, i.e. the probe may collide only with the workpiece.

The issue of collision-free motion planning in straight line motion has been addressed in assembly planning [18–20]. However, in these studies, only the assembly parts are considered for collision, ignoring the robot assembling the parts, its gripper, and the environment surrounding it. Furthermore, the assembly planning methods only addressed off-line planning in which the time constraint is not critical. In contrast, this paper presents an approach that considers the whole robot and the surrounding environment for manipulation. In addition, a major achievement in this approach is its real-time performance which makes it suitable for real-life dynamic environments.

The accessibility analysis proposed in this paper relies on a visibility test. The visibility test has been a fundamental problem since the very beginning of the computer graphics field. Among the visibility issues, the focus was dominantly on hidden surface removal. The problem has been mostly solved, and the *depth test* technique using *z-buffer* [21] dominates for interactive applications. The z-buffer stores the z-coordinate, i.e. depth value, for every pixel that has been rendered so far. When the next object is processed, the depth test discards its pixel if the pixel's depth is greater than the current value for that location in the z-buffer. This has the effect that a pixel occluded by another pixel is not rendered. Such an occluded pixel is said to be invisible. The depth test allows only visible pixels to be rendered. In addition to the z-buffer, current commodity graphics hardware supports the image-space *visibility query* [22,23], which computes the number of visible pixels belonging to a given primitive.

3. Object recognition and workspace modeling

The experimental robotic platform used in this study is shown in Fig. 1(a). It is equipped with a robotic manipulator (Fig. 1(b)), which is similar to a human arm with 7 DOF. The arm contains a *parallel jaw gripper* (Fig. 1(c)), which provides the basic capability for grasping and manipulating objects. A *stereo camera* is mounted on the parallel jaw gripper in an eye-on-hand configuration, and is used for modeling the workspace.

Fig. 2 shows the overall software system built for object manipulation. We designed and implemented a 3D workspace modeling module, and a visibility-based spatial reasoning module which works as a local motion planner. The workspace modeling techniques have been presented in [24], and this paper focuses on the visibility-based spatial reasoning techniques. For the sake of completeness, however, this section summarizes the workspace modeling techniques presented in [24].

Fig. 3 shows the experimental workspace named *Robot Café*. The stereo camera, the external parameters of which are calibrated a priori using a calibration block, captures the workspace on the fly and produces 3D *point clouds* as range data. The 3D point clouds are associated with photometric data such as colors of the corresponding pixels in the reference 2D images. Fig. 4(a) and (b) show the cluttered workspace and its point clouds, respectively. In Fig. 4(b), the 3D points are displayed with their colors, not only for the purpose of visualization but also for showing that both geometric and photometric features are used for object recognition and workspace modeling. The workspace at hand are assumed to be rich with textures enough for the stereo camera to capture a sufficient number of 3D points to process. Should a poorly textured environment be considered, an active 3D camera, e.g. with structured light [25], can replace the stereo camera.

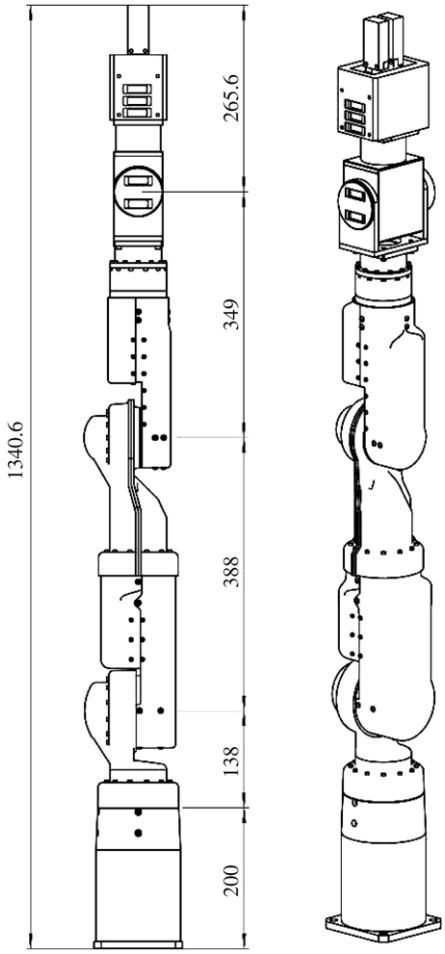
Suppose that the pizza sauce bottle located at the center of the cluttered environment in Fig. 4(a) is the object requested to be manipulated, henceforth called the *target object*. Recognition and pose estimation are done for the target objects that have geometric and photometric feature representations, such as 3D solid/geometric models and SIFT (scale-invariant feature transform) [26,27] models respectively, in the object database.

Recognition and pose estimation of the target object start with extraction of 3D features such as 3D SIFTs (i.e., SIFTs with their 3D position data), 3D lines, and/or other 3D shape features from 3D point clouds and 2D images. The extracted 3D features, which may be incomplete and noisy, are then matched with the target object model in the database to generate all the possible interpretations (with their probabilities) on the pose of the target object in the workspace. A series of stereo images captured successively on the fly are then used to filter multiple interpretations and converge them into the correct pose estimate. The positions of stereo camera to be used for filtering are obtained either from the visual odometer using 3D SIFTs or from the manipulator encoder readings. Note that filtering of multiple interpretations from a sequence of images is a process of accumulating evidences for robust decision making, thus making the target object recognition and pose estimation insensitive to environment variations such as lighting conditions and occlusions.

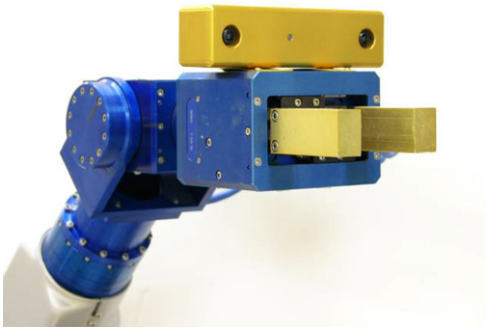
The target object model extracted from the database is placed into the pose estimated in the recognition phase, and then the 3D point clouds belonging to the target object are removed, as shown in Fig. 4(c). The remaining point clouds, i.e. the points in the neighborhood of the target object are taken as *obstacles*, which should be avoided during object manipulation. We often have a huge number of obstacle points, and therefore the points are dynamically sorted in regularly-spaced uniform 3D cells. If a cuboid cell contains at least one point, it is taken as an obstacle. Fig. 4(d) shows the workspace model composed of the target object and obstacle cells. The 3D obstacle cells can be updated as more 3D point clouds are accumulated from multiple viewpoints and distances.

Note that the cuboid cell representation is a conservative over-estimation of the extents of the obstacles. It is said to be conservative because whenever an obstacle cell is collision-free with the robot arm and target object, so are the actual obstacle points in it. Therefore, correct spatial reasoning with the obstacle cells does not produce an invalid path, e.g., in collision with the environment.

(a) Robot platform.



(b) Arm specification (http://www.amtec-robotics.com/robotersysteme_en.html).



(c) Gripper with an eye-on-hand configuration.

Fig. 1. A service robot.

4. Overview: Spatial reasoning for motion planning

The approach proposed in this paper delimits the cluttered environment neighboring the target object, as shown in Fig. 5. The cluttered neighborhood is represented by a sphere, named the *locality sphere*. Using the locality sphere, the overall path of the robot gripper to access, grasp, and deliver the target object is broken down into 4 sub-paths, as illustrated in Fig. 5: (i) from the *start pose* to the *access pose*, located at the boundary of the locality sphere, (ii) from the access pose to the *grasp pose* in which the object is grasped by closing the gripper's jaw, (iii) from the grasp pose to the *delivery pose*, also located at the locality sphere's boundary, and finally (iv) from the delivery pose to the *goal pose*. Moving the base of the robot has not been part of our current research.

The first and fourth sub-paths in Fig. 5 lie in a relatively clear area, and do not require a significant amount of time for planning. Thus, a general motion planning can be invoked for generating the sub-paths efficiently. In contrast, the second and

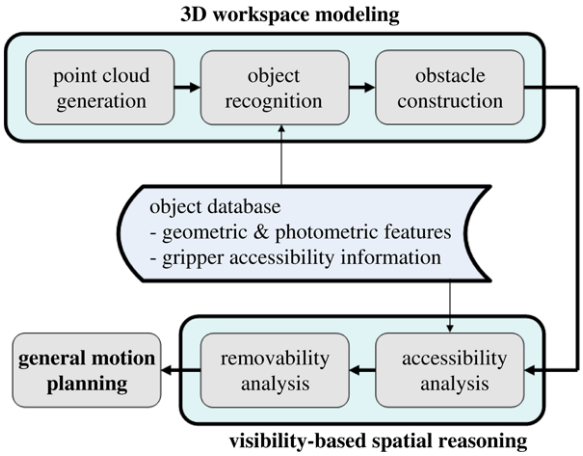


Fig. 2. Flow chart of the proposed system.

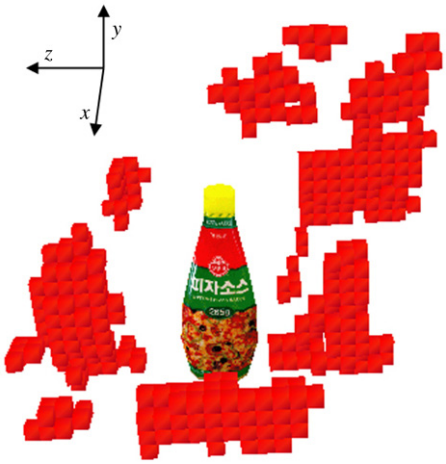
Fig. 3. Experimental workspace: Robot Café.

third sub-paths are located in the cluttered area, and planning for these is usually time-consuming. In the proposed approach, planning the paths within the locality sphere is *accelerated* by the gripper accessibility analysis (discussed in Section 5) and object removability analysis (discussed in Section 6). As illustrated in Fig. 2, the general motion planning is invoked after the accessibility and removability analyses are done.

(a) Objects in the workspace.

(b) Point clouds for the range data.

(c) Target object registered in the scene.



(d) Workspace model.

Fig. 4. Workspace modeling.

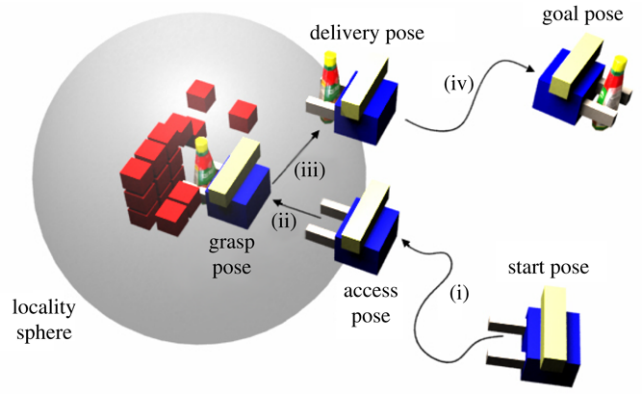


Fig. 5. Locality sphere and the overall path of the robot gripper.

In this study, the gripper's path within the locality sphere is restricted to be linear. Along a so-called *access direction*, the gripper translates, i.e. follows a linear path, from the access pose to the grasp pose. The gripper accessibility analysis discussed in Section 5 computes the access pose and the grasp pose, which are connected through the access direction. Since the accessibility analysis is done by graphics hardware, it is extremely fast. Note that, however, the accessibility analysis is done not for the entire arm, but for the gripper only. Therefore, motion planning is required to guarantee the collision-free path for the rest of the arm while the gripper travels along the access direction. In other words, the collision-free path of the gripper is guaranteed by accessibility analysis, while that of the rest of the arm is guaranteed by the motion planner. Motion planning only for the rest of the arm runs much faster than that for the entire arm.

At the grasp pose, the parallel jaws are closed, in order to grasp the target object. After grasping the object, following a linear direction which is called *removal direction,* the gripper will move it to the delivery pose. The removability analysis, presented in Section 6, guarantees the collision-free path of the gripper and the object, from the grasp pose to the delivery pose. In restricting the movement of the gripper to the computed linear path, motion planning is invoked for the rest of the arm.

In summary, the visibility-based spatial reasoning methods are proposed to complement general motion planning in two senses: (1) The sets of all possible grasp points and access/removal directions are determined extremely quickly, eliminating the time-consuming trial-and-error steps of the general motion planning approaches. (2) The gripper's linear paths within the cluttered environment are efficiently generated, reducing the complexity of the overall motion planning.

5. Gripper accessibility analysis

This section presents the technique for computing the access pose, the grasp pose, and the access direction. As discussed in Section 3, all target objects have object models in the database, and the object model contains gripper accessibility information of each object. For the pizza sauce bottle shown in Fig. 6, it is reasonable to define four *accessible directions*: $\pm x$ and $\pm z$ with respect to its local frame. (Among them, only two, $-x$ and $+z$, are illustrated in the figure). In contrast, access along vector $(-1, 0, -1)$ may not lead to stable gasping. The four directions, $\pm x$ and $\pm z$, are stored as recommended accessible directions in the object database.

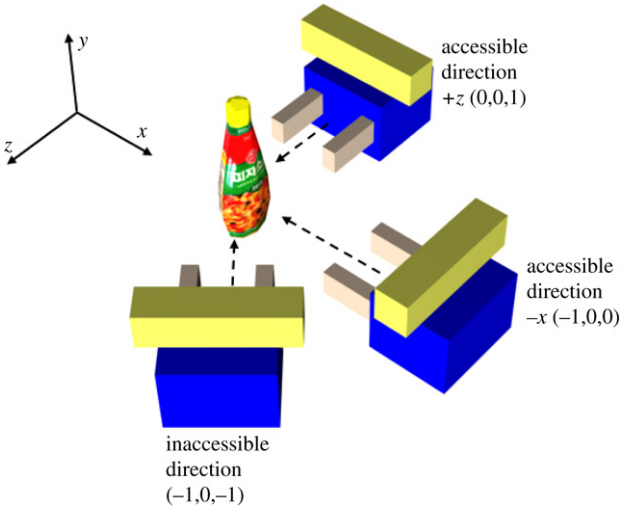
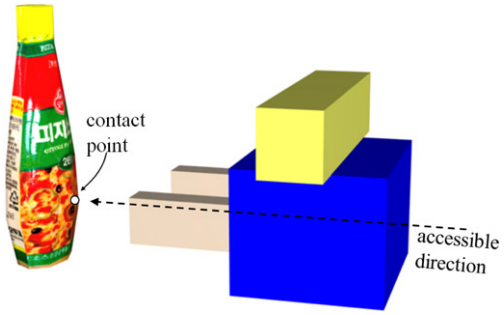


Fig. 6. Accessible directions in the database.



(a) Contact point.

(b) Contact curve associated with accessible direction $-x$.

(c) Contact curve associated with accessible direction $+z$.

Fig. 7. Contact point and curve.

For an accessible direction, the gripper can touch the target object at (infinitely) many *contact points*. As illustrated in Fig. 7(a), a contact point is defined as the intersection between the object surface and the gripper axis when the gripper translates toward the object along the accessible direction. A set of contact points can be represented as a curve, named a *contact curve*. In the current implementation, a contact curve is represented using a cubic Bézier curve.

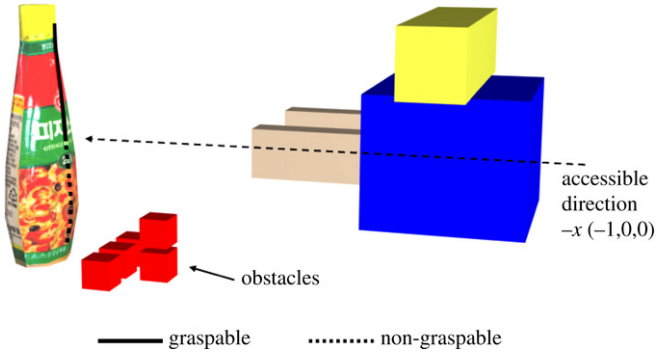
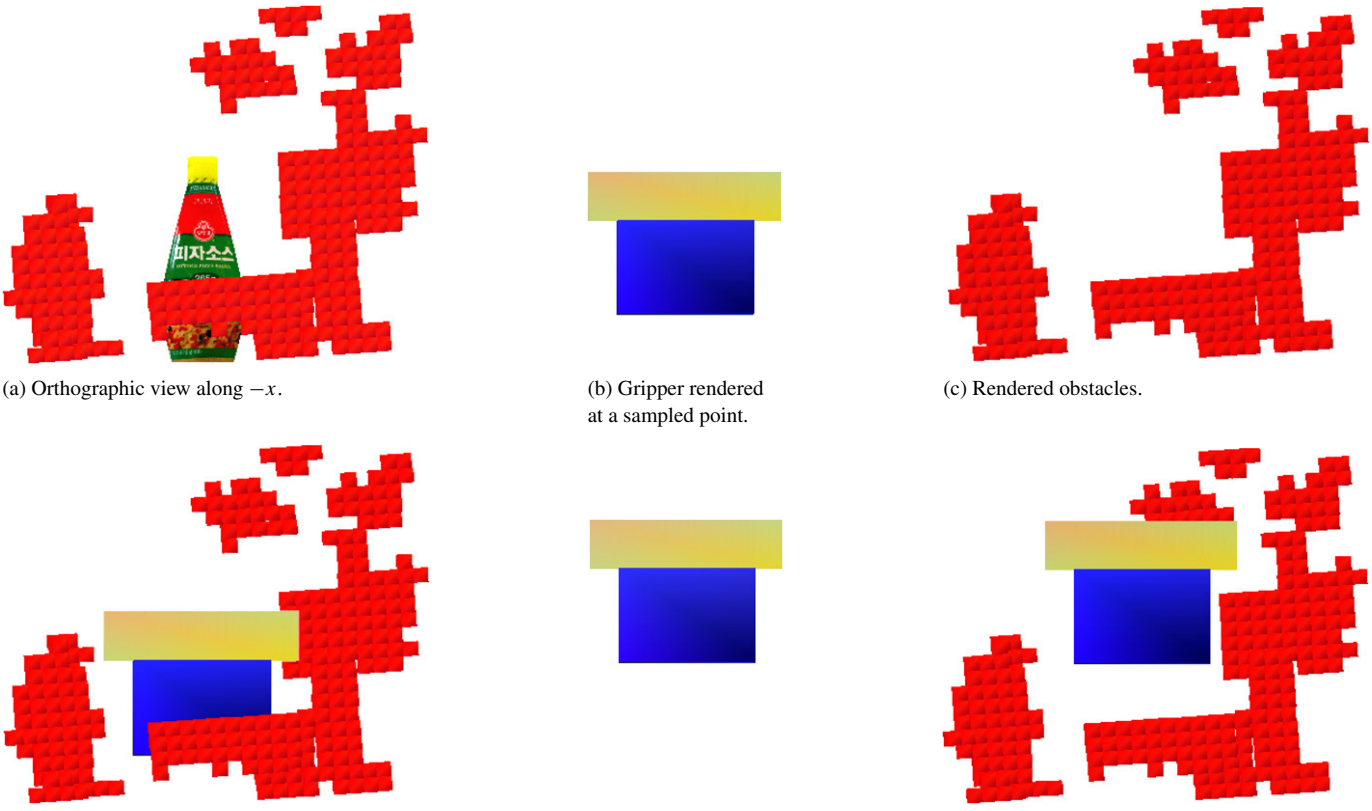


Fig. 8. Graspable points.

In the database, gripper accessibility is represented as a set of ⟨accessible direction, contact curve⟩ pairs. The pizza sauce bottle has four such pairs, two of which are shown in Fig. 7(b) and (c).

The gripper accessibility retrieved from the database should be *verified* by considering the neighborhood of the target object, i.e. each accessible direction should be verified by testing if the gripper can translate towards the target object along the direction without colliding with the obstacles. Note that the gripper accessibility can be verified for some contact points while it may not be for others due to obstacles. Fig. 8 shows an example, where the verified contact points are named *graspable*. In order to determine the graspable points, the contact curve is sampled, and then collision is tested for each sampled point. The collision test normally requires the swept volume of the gripper, which is generated by linearly connecting the gripper of the access pose to that of the grasp pose. In the proposed approach, the collision test using the swept volume can be simply replaced by *visibility query*, which is supported by virtually all kinds of graphics hardware.

As discussed in Section 2, the visibility query renders an object and returns the number of visible, i.e. un-occluded, pixels that have passed the depth test. For the visibility query, we use *orthographic projection*, the viewing direction of which is made equal to the accessible direction. Fig. 9(a) shows the workspace model of Fig. 4(d) seen along the accessible direction $-x$. The verification process starts by rendering the gripper at a sampled point with the visibility query. No pixel belonging to the gripper is occluded because obstacles have not been rendered. The gripper is fully visible, as shown in Fig. 9(b). The visibility query returns the number of visible pixels, say m. The returned number is recorded, and the depth buffer is cleared. Then, the obstacles within the locality sphere are rendered, as shown in Fig. 9(c). Without clearing the depth buffer, the gripper is then rendered, i.e. the gripper is inserted into the rendered scene of the obstacles in Fig. 9(c). The result is shown in Fig. 9(d). It is observed that the lower part of the gripper is occluded by the obstacles and therefore invisible. If the number of visible pixels returned by the visibility query is n, it is less than m, indicating



(a) Orthographic view along $-x$.

(b) Gripper rendered at a sampled point.

(c) Rendered obstacles.

(d) Ripper rendered into the obstacles (collision).

(e) Gripper rendered at another sampled point.

(f) Gripper rendered into the obstacles (no collision).

Fig. 9. Gripper visibility test.

(a) Pizza sauce bottle. (b) Juice bottle.

Fig. 10. Gripper widths in a 1D texture.

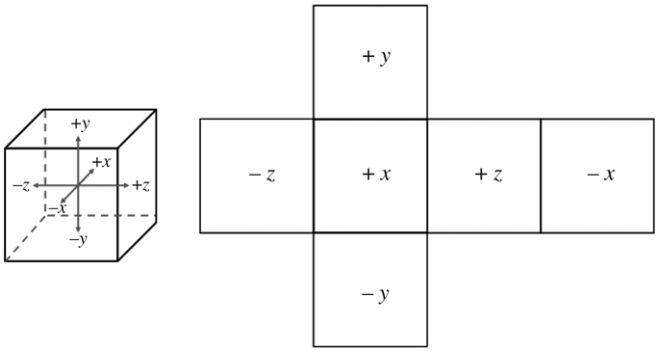


Fig. 11. Cube map.

that the gripper collides with obstacles if it translates towards the sampled point. The sampled point is not graspable.

The visibility analysis is repeated for all sampled points. Fig. 9(e) shows the gripper rendered at a different sampled point. The number of visible pixels m is recorded. Then, the gripper is rendered once again onto the already-rendered environment shown in Fig. 9(c). The result is shown in Fig. 9(f). At this time, no part of the gripper is occluded by the obstacles, i.e. the gripper is fully visible, and the returned value n is equal to m. Therefore, the gripper is determined to be collision-free when translating towards the sampled point, which is consequently taken as the graspable point.

In the current implementation, 10 points are sampled per contact curve, and the set of all graspable points is computed for the target object. (This requires a few milliseconds, as will be discussed in Section 8.) In order to select the best one out of the graspable point set, a cost function has been devised:

$$C = c_1(1/h) + c_2(1/w) \tag{1}$$

where h is the normalized height of a graspable point, and w is the normalized object width at the graspable point. The higher and the thicker, the more stable grasping is achieved and the less cost is needed for grasping. (When w is greater than the maximum distance between the jaws of the gripper, however, w is set to 0 to make the cost C infinity.) When the best graspable point with the least cost is selected, the grasp and access poses are determined. The grasp pose is used as the input for the next stage, which computes the removal direction of both the target object and gripper.

Note that a contact point may require a distinct gripper jaw width, as illustrated in Fig. 10(a). These varying widths are recorded in a *1D texture*, which is normally represented in a 1D array of floating-point values, i.e., each texel is a floating-point value representing a width. The 1D texture is indexed by the parameter $t = [0, 1]$ of the Bézier contact curve. In other words, the Bézier curve's parameter at a sampled point is used to reference the 1D texture and retrieve the width used for evaluating the cost function in Eq. (1). Unless an object is sharply curved, the size of the texture map can usually be kept small. This is especially true for box-shaped objects where the entire contact curve is assigned a uniform gripper jaw width, as shown in Fig. 10(b), where only 2 texels can define the 1D texture.

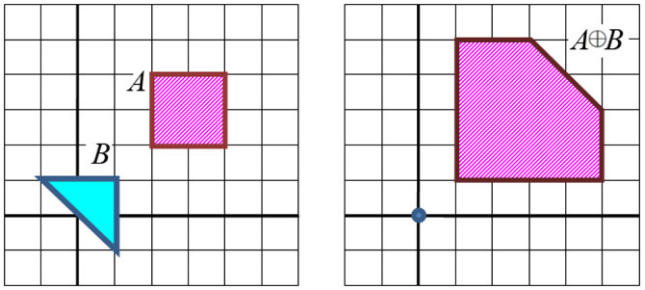
6. Object removability analysis

In computing the collision-free removal directions, *environment map* [28] and *Minkowski sum* [29] prove to be useful. In computer graphics, the environment map is used to describe the scene surrounding an object. The most popular implementation of the environment map is the *cube map* [17], shown in Fig. 11, where each face covers a 90° field of view both horizontally and vertically. There are six faces per cube, and each face is implemented as a square 2D image texture. Notice that a pixel (more precisely, a texel) in the cube map corresponds to a vector from the origin at the cube center.

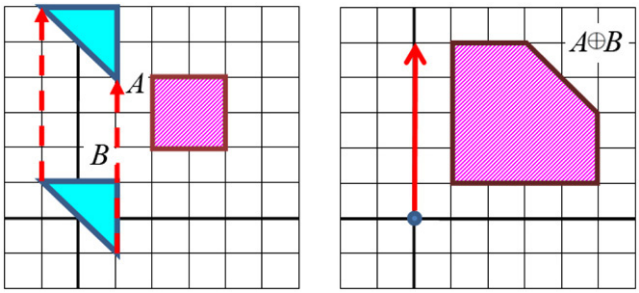
The Minkowski sum of two point sets A and B is defined as the set

$$A \oplus B = \{a - b : a \in A, b \in B\} \tag{2}$$

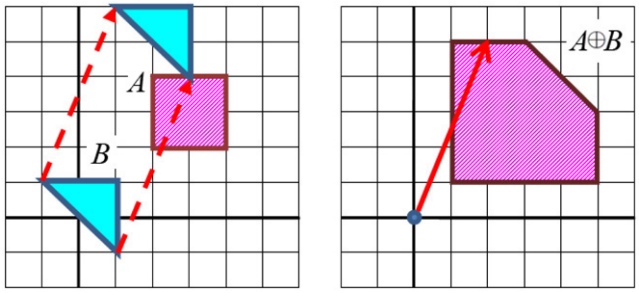
where $a - b$ is the vector sum of the position vectors a and $-b$. Geometrically, the Minkowski sum is obtained by adding A to the reflection of B about the origin, as shown in Fig. 12(a). Suppose that B is moving past the obstacle A. The collision



(a) Minkowski sum.



(b) No collision.



(c) Collision.

Fig. 12. Minkowski sum and collision detection.

test between them is often implemented as follows [30]: A is 'grown' by B to make $A \oplus B$, B is 'shrunk' into a point, and the moving point, i.e. a *ray*, is tested for collision with the grown obstacle $A \oplus B$, as shown in Fig. 12(b) and (c).

Suppose that the gripper is placed at a graspable point to make the grasp pose, as shown in Fig. 13(a). The graspable point is taken as the origin of the coordinate system. Let us denote the combination of the object and gripper by O. The point clouds P is simply a set of points, and therefore the Minkowski sum $P \oplus O$ is the union of $-O$ located at each point p_i in the point clouds. Fig. 13(b) visualizes the Minkowski sum $P \oplus O$.

Placing the viewpoint (eye) at the graspable point, the Minkowski sum $P \oplus O$ is projected to the cube map. Projection is done for each square image of the cube map. Fig. 13(c) shows the result, and Fig. 13(d) is its unfolded version. In the black-and-white images of Fig. 13(c) and (d), the texels which are not occupied by the projected Minkowski sum are colored in white. The white texels represent the rays or directions, along which no part of the target object and the gripper is occluded by the obstacles. In other words, the target object and the gripper do not collide with the obstacles when they linearly translate along the directions represented by the white texels. Such directions are called the *removable directions*.

For selecting the best one out of the removal directions, a simple heuristic has been adopted. Note that the removable directions, shown as white texels in Fig. 13(c), form a set of connected components in the cube map surface. The connected components are easily identified using a labeling algorithm. The largest connected component is selected, and its center texel is selected as the best removal direction. Fig. 13(e) shows the selected removal direction, both in the cube map and in the workspace.

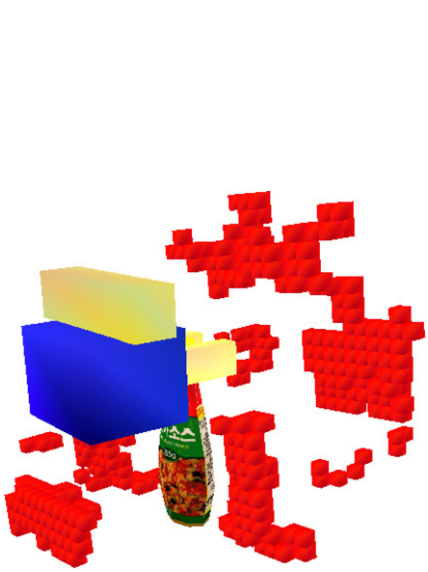
Suppose that no removable direction can be found from the cube map. This happens when the access direction and the graspable point have been verified but the gripper cannot remove the object along any direction. In such a case, the next-best graspable point is selected, according to the cost function discussed in Section 5. With the newly selected graspable point, the stage for computing the removal directions is resumed, i.e. the gripper is placed at the new graspable point to make a distinct grasp pose, and the Minkowski sum of the object and the gripper is projected to the cube map to find the removal directions.

If the number of points in the point clouds is too large, Minkowski sum projection may consume a significant amount of time. In such a case, not the points themselves, but the obstacle cells are used for computing the Minkowski sum, in order to achieve real-time performance. According to the cell size, the combination of the object and gripper is scaled up to make O'. The union of $-O'$ located at each cell's center c_i defines the Minkowski sum, and the removal directions are computed by projecting the Minkowski to the cube map. (The Minkowski sum in Fig. 13 has been constructed in this way.) In this approximate approach, the selected removal direction requires verification to ensure that it is collision-free. The verification is done through the visibility query. The visibility test presented in Section 5 is used, where the gripper is replaced by O (the combination of the object and gripper), and the viewing direction is set to the opposite of the selected removal direction.

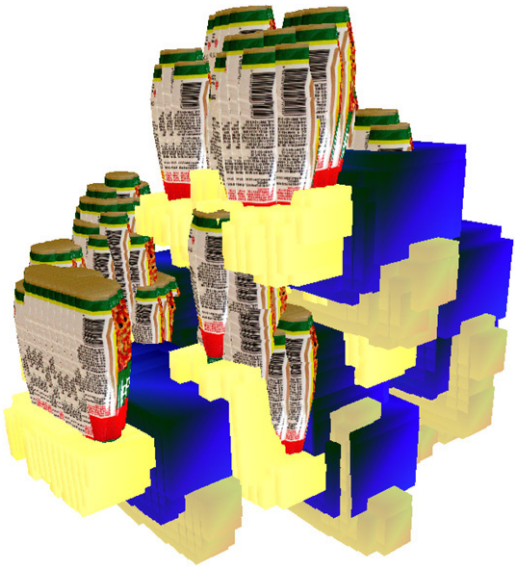
Instead of the cube map, the environment map can be implemented as a *sphere map*, which is a 2D representation of the full 360° view of the scene surrounding an object, as if taken through a fish-eye lens [31]. Considering that only the obstacle cells 'within the locality sphere' are used for Minkowski sum generation, the sphere map might look like a better choice than the cube map. Note that, however, the cube and sphere maps are just different tools for the same purpose, i.e. they describe the scene surrounding an object. Furthermore, the cube map is easier to implement than the sphere map. Therefore, the environment map is implemented as a cube map in the current study.

7. Discussions

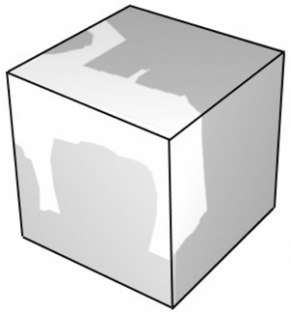
In the current study, the elastic strip method is used for general motion planning, and is accelerated by gripper



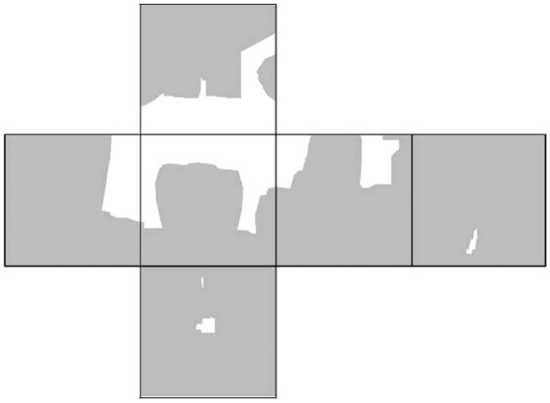
(a) Gripper at the grasp pose.



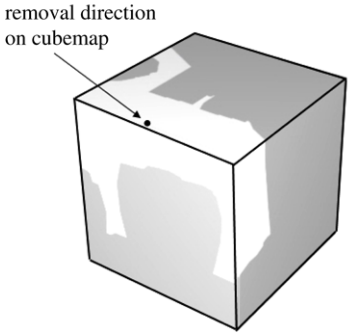
(b) Minkowski sum.

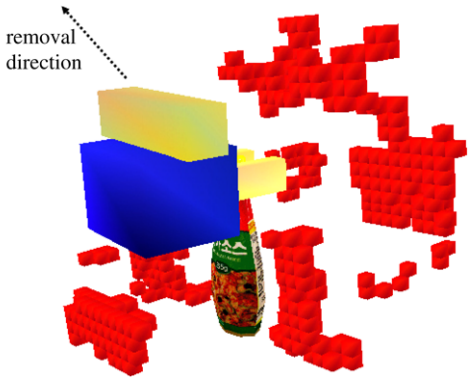


(c) Minkowski sum projected onto cube map.



(d) Unfolded cube map.





(e) Removal direction.

Fig. 13. Projection of Minkowski sum.

accessibility analysis (discussed in Section 5) and object removability analysis (discussed in Section 6) within the locality sphere, as illustrated in Fig. 5. The accessibility analysis is used to determine the set of all *collision-free linear paths of the gripper* from the access poses to the grasp poses. Without accessibility analysis, the elastic strip-based motion planner will run a trial-and-error-based loop to find a collision-free grasp pose. For a sampled contact point, computing a collision-free grasp pose is expensive when implemented in software. Due to obstacles in cluttered environments, a trial may also easily fail. Then, the motion planner should select another contact point and the expensive computation will be

Table 1
Performance measurement in Robot Café scene (ES: Elastic Strip)

	Accessibility analysis	Start pose → access pose	Access pose → grasp pose	Removability analysis	Grasp pose → delivery pose	Delivery pose → goal pose	Total time
ES + spatial reasoning	0.0038	0.375	1.188	0.022	1.188	0.359	3.136
		1.563(0.375 + 1.188)			1.547(1.188 + 0.359)		
ES only	NA	3.015 (best case) 4.852 (worst case)		NA	23.158		26.173 (best) 28.010 (worst)

Number of total obstacle cells = 348; radius = 25 cm; number of obstacle cells inside/outside the locality sphere = 348/0.

repeated until it finds a graspable point. This kind of repeated loop may often hinder real-time planning, as is demonstrated in Section 8. Worse still, even if a collision-free grasp pose is computed, the motion planning from the access pose to the grasp pose may consume a significant amount of time in cluttered environments. Worst of all, a collision-free path to the computed grasp pose may not exist, due to the obstacles on the way to the grasp pose. Then, the whole planning must be repeated by computing another collision-free grasp pose. In the case of probabilistic approaches, the trial-and-error can be eliminated by examining multiple grasp points in the query phase of the algorithm. However, it increases time needed to generate a satisfactory roadmap that can be connected to the multiple grasp points.

As the accessibility analysis is done not for the entire arm, but for the gripper, motion planning is invoked to generate the collision-free path for the rest of the robot arm while the gripper is restricted to travel along the access direction. In the elastic strip method used for the current implementation, the linear path from the access pose to the grasp pose (computed for the gripper) is taken as an input *candidate path*. The planned path of the rest of the arm may be curved, and more precisely, consists of multiple line segments, while the gripper path is restricted to be linear. Note again that the collision-free path of the gripper is guaranteed by accessibility analysis, while that of the rest of the arm is guaranteed by the motion planner.

The accessibility analysis runs quickly, as shown in Section 8. Skipping of motion planning for the gripper greatly increases the efficiency of the elastic strip motion planning. This is because motion planning for the gripper, which is the closest part of the robot to the cluttered environment, is the most time-consuming component. This complex component has been replaced by a simple and fast accessibility analysis.

Similar discussions can be made for the removal directions. From the grasp pose to the delivery pose, the collision-free path of the gripper is guaranteed by the removability analysis presented in Section 6. Therefore, while the gripper is restricted to travel along the removal direction, motion planning is invoked for the rest of the arm. This greatly increases efficiency.

In many areas of computer graphics field, there have been conflicts between *object-space approach* and *image-space approach*. In the visible surface determination area, for example, the z-buffer algorithm discussed in Section 2 is an image-space approach while depth sorting and bsp (binary space partitioning) algorithms [32] represent the object-space approach. Both of the accessibility analysis based on visibility query and the removability analysis based on the cube map are image-space algorithms. They run quite fast, especially because they are implemented in GPU. On the other hand, a common problem of the image-space approach is that its effectiveness is limited by the image-space resolution and it may be exposed to sampling error. In the proposed approach, various safeguards have been devised, such as the conservative obstacle representation discussed in Section 3, but the image-space algorithms for accessibility/removability analyses may not be completely free from the sampling error. The research work presented in this paper pursues a trade-off between efficiency and accuracy.

8. Experiments

The experimental mobile manipulator consists of a 7-DOF Amtec lightweight arm (Amtec Robotics GmbH) with a parallel jaw gripper installed on top of an Active Media Robotic's Powerbot. The algorithms presented in this paper have been implemented on a modest PC (Pentium 4 2.8 GHz, and NVIDIA GeForce 6600GT graphics card). Fig. 14 shows a series of snapshots for the manipulation task, in which the robot is requested to grasp a pizza sauce bottle in *Robot Café*. Fig. 15 shows the corresponding scenes generated by simulation. Fig. 14(a) shows the start pose, where the images of the workspace are captured and its computer model is constructed. Fig. 14(b) shows the access pose from which a translation motion toward the grasp pose is performed. Fig. 14(c) shows the grasp pose. The obstacles including the banana in front of the pizza sauce bottle do not allow the bottle to be removed horizontally. Similarly, the flower above the bottle does not allow vertical motion. Fig. 14(d) shows the collision-free removal operation, toward the delivery pose, computed using the cube map. Finally, Fig. 14(e) shows the goal pose.

Table 1 compares the performances of motion planning (elastic strip: ES) with and without spatial reasoning for the environment shown in Fig. 14. In this experiment, the spatial reasoning for accessibility analysis requires 0.0038 s, and motion planning from the start pose to the grasp pose takes 1.563 s. Without accessibility analysis, the motion planner takes 3.015 s in the best case and 4.852 s in the worst case. Recall that the bottle has 4 contact curves and each curve is sampled at 10 points, i.e. 40 contact points are tested in total. The best case occurs when the motion planner selects the top of the object for grasping at the first trial, the collision-free grasp pose is obtained, and finally the collision-free path from the start pose to the grasp pose is found. However, it takes more time than

(a) start pose

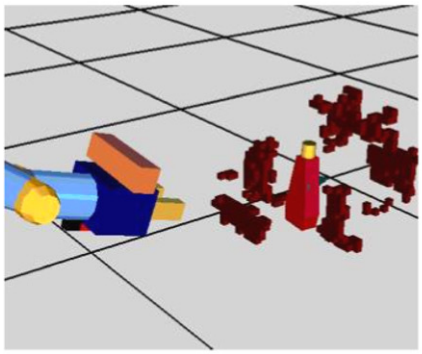
(b) access pose

(c) grasp pose

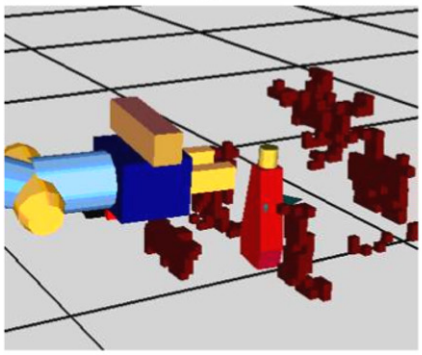
(d) delivery pose

(e) goal pose

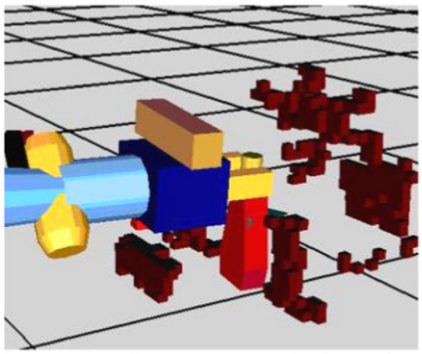
Fig. 14. Snapshots of experiment I.



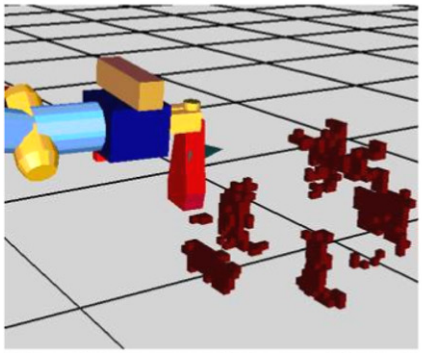
(a) start pose



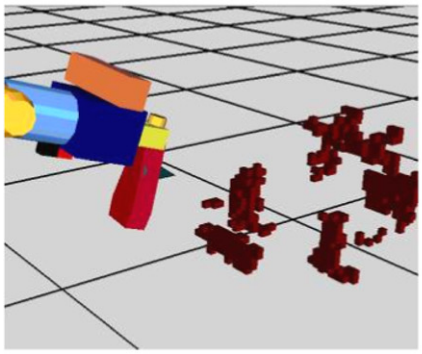
(b) access pose



(c) grasp pose



(d) delivery pose



(e) goal pose

Fig. 15. Simulation scenes of experiment I.

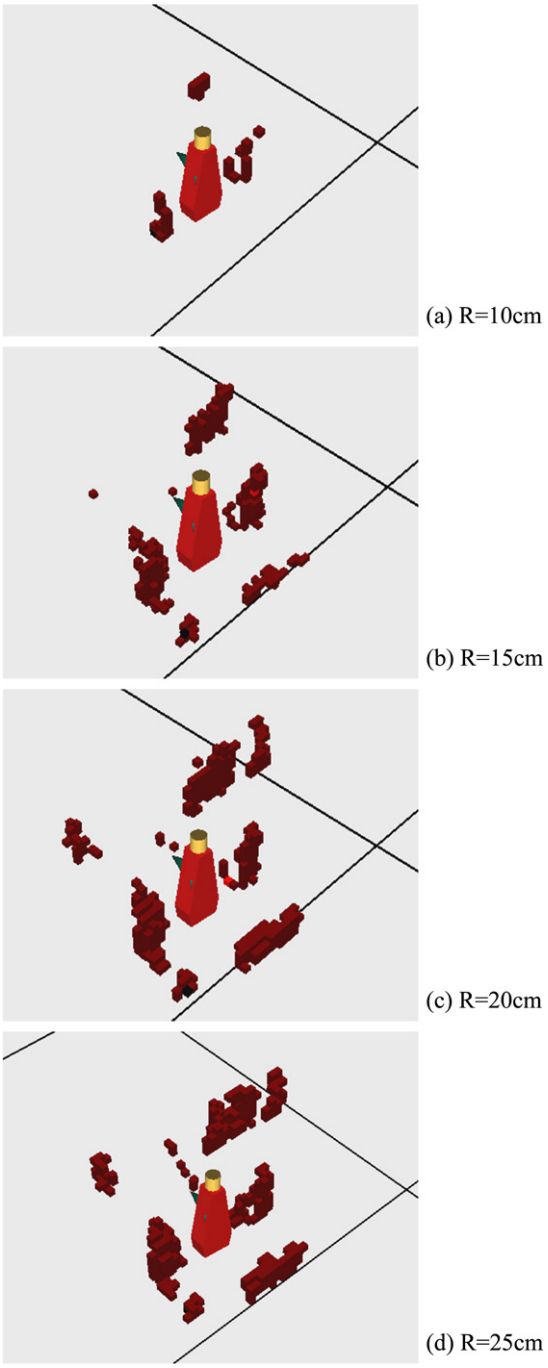


Fig. 16. Obstacle cells for varying radii of the locality sphere.

when the motion planner is accelerated by spatial reasoning, i.e. 3.015 vs. 1.567 (0.0038 + 1.563) s. The worst case occurs when the motion planner obtains a collision-free path after trying many contact points, to which no path from the start pose can be found.[1]

[1] To determine a collision-free grasp pose, the motion planner checks if the given contact point is reachable by the arm. If the contact point can be reached, the arm is tested for collision with the obstacles at the grasp pose. In the experiments, the back face of the bottle is not reachable, and therefore all contact points at the back face are immediately rejected. The sides of the object are reachable, but are rejected by the collision test, which is also fast. No path planning is invoked for the back and side faces. Thus, the time difference between the worst case (4.852 s) and the best case (3.015 s) is not so big.

The spatial reasoning for removability analysis requires 0.022 s, and the whole motion planning from the grasp pose to the goal pose takes 1.547 s. Without removability analysis, the motion planner takes 23.158 s to generate a collision-free path. The obstacles surrounding the target object, such as banana, flower, etc., result in the planner consuming a large amount of time. Note that, in generating the entire path from the start pose to the goal pose, motion planning with spatial reasoning is approximately 9 times faster.

The radius of the locality sphere determines the length of the linear path along which the gripper travels. Depending on the dexterity and manipulability of the arm and characteristics of the gripper, the radius of the locality sphere can vary. For a dexterous arm, the radius can be minimized, e.g. the gripper length plus half the target object's width. The limited dexterity of the arm requires a larger radius of the locality sphere. The robot arm shown in Fig. 1 has limited dexterity, and the sphere radius is made large enough to include as many obstacles immediately neighboring the target object as possible. See below for a more detailed discussion.

Table 2 shows the performance measurement with varying radii of the locality sphere for the environment of *Robot Café*. As the radius increases from 10 to 25 cm, the number of obstacle cells within the locality sphere also increases from 41 to 348, as also illustrated in Fig. 16. Obviously, more obstacle cells lead to less graspable points, and the number of graspable points is reduced from 29 (at 10 cm) to 2 (at 25 cm). The remaining columns of Table 2 show the times required for spatial reasoning and motion planning, and are illustrated in Fig. 17. They can be discussed as follows:

- The accessibility analysis time remains almost constant even though the number of obstacle cells increases as the radius of the locality sphere increases. This is due to the fact that GPU's visibility query is rarely sensitive to the size of the input data.
- From the start pose to the access pose, the motion planning time sharply decreases when the radius of the locality sphere increases from 10 to 15 cm and finally to 20 cm. This is because the number of obstacle cells outside the locality sphere rapidly drops from 307 to 128, and then to 12. In other words, the area in which the general motion planner needs to operate becomes less crowded. In contrast, the planning time does not greatly decrease, from 20 to 25 cm, because just 12 obstacle cells disappear.
- From the access pose to the grasp pose, the time for motion planning accelerated by spatial reasoning remains almost constant. This is because the general motion planning for the 'rest' of the arm, i.e. the part of the robot arm excluding its gripper, considers not only the obstacle cells inside the locality sphere but also the outside cells. In other words, the number of obstacle cells to consider is largely independent of the locality sphere's radius.
- The removability analysis time is not much affected by the number of obstacle cells within the locality sphere.

Table 2
Performance measurement for varying radii of the locality sphere

R	Obstacle cells (in/out)	Graspable points	Accessibility analysis	Start pose → access pose	Access pose → grasp pose	Removability analysis	Grasp pose → delivery pose	Delivery pose → goal pose	Total time
10	41/307	29	0.0037	1.968	1.204 (3.172)	0.021	1.171	1.985 (3.156)	6.353
15	220/128	7	0.0037	1.079	1.171 (2.250)	0.022	1.178	1.072 (2.250)	4.526
20	336/12	3	0.0038	0.407	1.202 (1.609)	0.022	1.196	0.391 (1.587)	3.222
25	348/0	2	0.0038	0.375	1.188 (1.563)	0.022	1.188	0.359 (1.547)	3.136

Table 3
Performance measurement in the table scene (ES: Elastic Strip)

	Accessibility analysis	Start pose → access pose	Access pose → grasp pose	Removability analysis	Grasp pose → delivery pose	Delivery pose → goal pose	Total time
ES + spatial reasoning	0.0033	0.297	0.921 (1.218)	0.022	0.922	0.291 (1.213)	2.456
ES only	NA	2.433 (best case) 3.602 (worst case)		NA	22.184		24.617 (best) 25.786 (worst)

Number of total obstacle cells = 327; radius = 20 cm; number of obstacle cells inside/outside the locality sphere = 327/0.

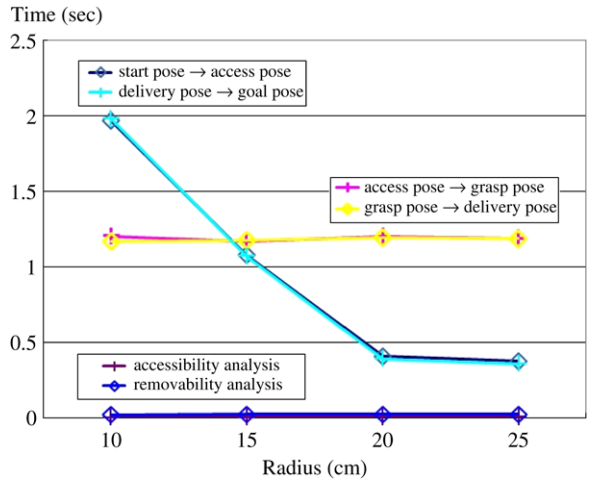


Fig. 17. Execution times for varying radii of the locality sphere.

This is because the GPU operations run quite fast and are rarely sensitive to the size of the input data while the CPU operations dominate the time which is fairly constant.

– From the grasp pose to the delivery pose, the motion planning time remains almost constant for the same reasons explained above for the sub-path from the access pose to the grasp pose.
– From the delivery pose to the goal pose, the planning time sharply decreases and then flattens for the same reasons explained above for the sub-path from the start pose to the access pose.

Table 2 shows that the total planning time decreases as the radius increases. This justifies why the radius of the locality sphere is made large enough to include as many obstacles as possible. Note that, however, we often cannot find a graspable point for an overly large radius. This happens when no clear linear path of the gripper can be found due to the crowded obstacles. Even though a long linear path is found, following the linear path of the gripper puts too many constraints on the robot arm and consequently a motion plan for the entire arm may not be found. Therefore, we need an upper limit of the radius, and it is set to 30 cm in the current implementation. Within the upper limit, the smallest radius that contains the maximum number of the obstacle cells is selected, at intervals of 5 cm. In Table 2, for example, such a radius is 25 cm. When no graspable point is found in the radius, we reduce the radius by 5 cm at a time, and repeat the accessibility analysis.

Figs. 18 and 19 show demonstrations for another target object, a juice bottle, in a different environment. Table 3 compares the performances of motion planning with and without spatial reasoning, and also shows significant performance gain of the proposed spatial reasoning.

Fig. 20 shows a more cluttered environment, where the juice bottle is the target object, and Fig. 21 shows a series of snapshots for the manipulation task. Table 4 compares the performances. In such a cluttered environment, the disadvantage of motion planning without spatial reasoning is obvious due to the higher possibility of falling into local minima. In Table 4, the object removal task (from the grasp pose to the goal pose) consumes a huge amount of time, 45.485 s, and eventually realizes the existence of the local minimum,[2] i.e. no path is found.

[2] The motion planner can use a local minima escape method, such as random walk, to try to generate a path. If the motion planner is lucky, the planner may select a collision-free path in its first trial and succeed. However, there is no upper bound for the time needed to find a collision-free path.

(a) start pose

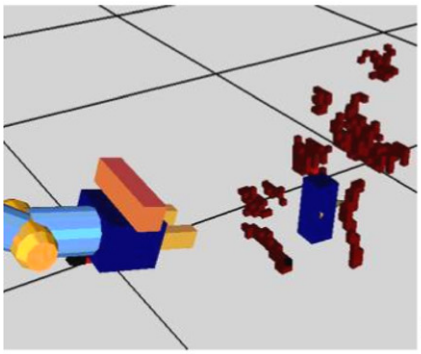
(b) access pose

(c) grasp pose

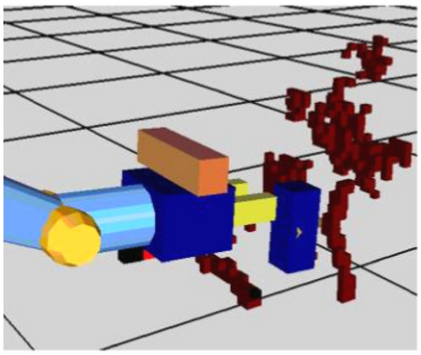
(d) delivery pose

(e) goal pose

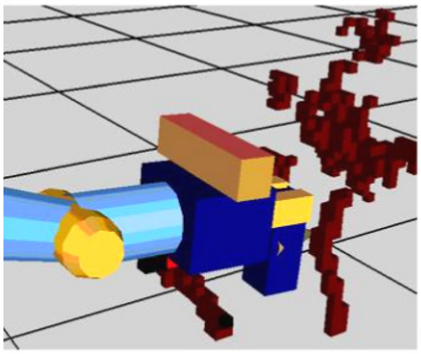
Fig. 18. Snapshots of experiment II.



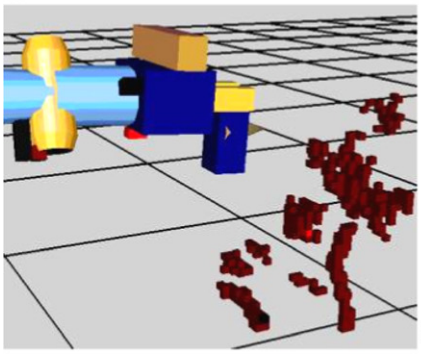
(a) start pose



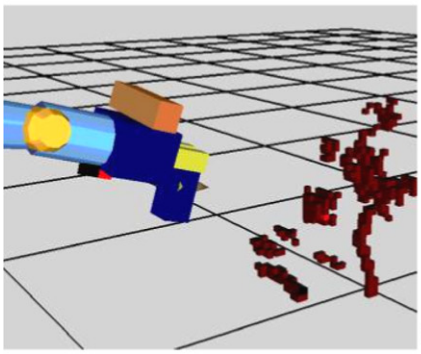
(b) access pose



(c) grasp pose



(d) delivery pose



(e) goal pose

Fig. 19. Simulation scenes of experiment II.

Fig. 20. Workspace for experiment III.

9. Conclusion

This paper presents a novel approach to accessibility and removability analyses for object manipulation tasks: visibility-based spatial reasoning. The accessibility and removability analyses utilize the visibility query and cube map, which are accelerated by graphics hardware. The performance and robustness of the proposed approach are evaluated in cluttered indoor environments. The experimental results demonstrated that the proposed methods are sufficiently fast and robust to manipulate 3D objects for real-time robotic applications.

As shown in the experimental results for the varying radii of the locality sphere, the spatial reasoning for accessibility/removability analyses is largely independent of the locality sphere's radius and runs very fast. Consequently, multiple access and grasp poses associated with different radii can be generated very fast. Similarly, we can obtain multiple delivery poses for different radii of the locality sphere. In the context of real-time motion planning, our next step is to develop a probabilistic motion planner that can process such multiple initial and goal configurations at once. This approach may increase the chance of finding a better path in real time.

In the context of the home service robot, the objects to be manipulated are limited to daily necessities. Fortunately, their shapes are mostly symmetric, and simple objects such as boxes and bottles constitute the majority. It is simple to construct a database for such simple-shaped and symmetric daily necessities. The proposed approach will be extended to enable one to deal with a wider range of objects with arbitrary silhouettes. For such an extension, the proposed framework can be used without alteration. However, the authoring stage for object database construction is becoming increasingly complicated, and a more general representation for contact points is being investigated.

Fig. 21. Snapshots of experiment III.

Table 4
Performance measurement in the refrigerator scene

	Accessibility analysis	Start pose → access pose	Access pose → grasp pose	Removability analysis	Grasp pose → delivery pose	Delivery pose → goal pose	Total time
ES + spatial reasoning	0.0043	0.430	2.210	0.027	2.218	0.438	5.327
		2.640			2.656		
ES only	NA	3.767 (best case) 9.860 (worst case)		NA	45.485		NA

Number of total obstacle cells = 522; radius = 30 cm; number of obstacle cells inside/outside the locality sphere = 521/1.

Acknowledgements

The preliminary versions of the sections in this paper have appeared in [33–35]. The work presented in this paper was performed for the 21st Century Frontier R&D Programs funded by the Ministry of Science and Technology of Korea. This work was also supported by grant No. R01-2006-000-11297-0 from the Basic Research Program of the Korea Science & Engineering Foundation and by MIC, Korea under ITRC IITA-2006-(C1090-0603-0046).

References

[1] Reif H. Complexity of the Mover's problem and generalizations. In: Proc. of the 20th IEEE symposium on foundations of computer science. 1979. p. 421–7.
[2] Kuffner J, Nishiwaki K, Kagami S, Inaba M, Inoue H. Motion planning for humanoid robots. In: Proc. of the 11th international symposium of robotics research. 2003.
[3] Hsu D, Latombe J, Kurniawati H. On the probabilistic foundations of probabilistic roadmap planning. In: Proc. of the 12th international symposium on robotics research. 2005.
[4] Hsu D, Kavraki L, Latombe J, Motwani R, Sorkin S. On finding narrow passages with probabilistic roadmap planners. In: Agrawal P, et al., editors. Robotics: The algorithm perspectives. Wellesley (MA, USA): A.K. Peters; 1998. p. 141–54.
[5] Hsu D, Jiang T, Reif J, Sun Z. The bridge test for sampling narrow passages with probabilistic roadmap planners. In: Proc. of the IEEE international conference on robotics & automation. 2003.
[6] Boor V, Overmars M, Stappen A. The gaussian sampling strategy for probabilistic roadmap planners. In: Proc. of the IEEE international conference on robotics & automation. 1999.
[7] Brock O. Generating robot motion: the integration of planning and execution. Ph.D. thesis. Stanford University; 1999.
[8] Hong S, Moradi H, Lee S, Lee S. A comparative study on local minima escape in potential field method. In: Proc. of the 3rd International conference on autonomous robots and agents. 2006.
[9] Spyridi A. Automatic generation of high level inspection plans for coordinate measuring machines. Ph.D. thesis. University of Southern California; 1994.
[10] Wilson R. Geometric reasoning about assembly tools. Artificial Intelligence 1998;98(1–2):237–79.
[11] Trucco E, Umasuthan M, Wallace A, Roberto V. Model-based planning of optimal sensor placements for inspection. IEEE Transactions on Robotics and Automation 1997;13(2):182–93.
[12] Gupta P, Janardan R, Majhi J, Woo T. Efficient geometric algorithms for workpiece orientation in 4- and 5-axis NC machining. Computer-Aided Design 1996;28(8):577–87.
[13] Spyridi A, Requicha A. Accessibility analysis for polyhedral objects. In: Tzafestas SG, editor. Engineering Systems with Intelligence: Concepts, Tools and Applications. Dordrecht (Holland): Kluwer Academic Publishers, Inc.; 1991. p. 317–24.
[14] Lim C, Menq C. CMM feature accessibility and path generation. International Journal of Production Research 1994;32:597–618.
[15] Limaiem A, ElMaraghy H. A general method for accessibility analysis. In: Proc. of the IEEE international conference on robotics & automation. 1997.
[16] Spitz S, Requicha A. Accessibility analysis using computer graphics hardware. IEEE Transactions on Visualization and Computer Graphics 2000;6(3):208–19.
[17] Green N. Environment mapping and other techniques of world projections. IEEE Computer Graphics and Applications 1986;6(11):21–9.
[18] Ambler A., Barrow H, Brown C, Burstall R, Popplestone R. A versatile computer-controlled assembly system. In: Proc. of the 3rd international joint conference on artificial intelligence. 1973. p. 298–307.
[19] Wilson R. On geometric assembly planning. Ph.D. thesis. Department of Computer Science, Stanford University; March 1992.
[20] Goldberg K., Moradi H.. Compiling assembly plans into hard automation. In: Proc. of the international conference on robotics and automation. 1996.
[21] Catmull E. A subdivision algorithm for computer display of curved surfaces. Ph.D. thesis. University of Utah; 1974.
[22] GL_HP_occlusion_test, http://oss.sgi.com/projects/ogl-sample/registry/HP/occlusion_test.txt.
[23] GL_NV_occlusion_query, http://oss.sgi.com/projects/ogl-sample/registry/NV/occlusion_query.txt.
[24] Lee S, Jang D, Kim E, Hong S, Han J. A real-time 3D workspace modeling with stereo camera. In: Proc. of the IEEE/RSJ international conference on intelligent robots and systems. 2005.
[25] Lee S, Choi J, Baek S, Jung B, Choi C, Kim H et al. A 3D IR camera with variable structured light for home service robots. In: Proc. of the IEEE international conference on robotics and automation. 2005. p. 1859–64.
[26] Se S, Lowe D, Little J. Vision-based mapping with backward correction. In: Proc. of IEEE/RSJ international conference on intelligent robots and systems (IROS02). 2002.
[27] Garcia M, Solana A. 3D simultaneous localization and modeling from stereo vision. In: Proc of the international conference on robotics & automation. 2004.
[28] Blinn J, Newell M. Texture and Reflection in Computer generated Images. Communications of the ACM 1976;19(10):542–7.
[29] Minkowski H. Volumen and Oberflache. Mathematische Annalen 1903; 57:447–95.
[30] Ericson C. Real-time collision detection. Morgan Kaufmann; 2004.
[31] Miller G, Hoffman C. Illumination and reflection maps: Simulated objects in simulated and real environments. In: SIGGRAPH advanced computer graphics animation course notes. 1984.
[32] Hearn D, Baker M. Computer graphics, C Version. 2nd ed. Prentice Hall; 1996.
[33] Jang H, Lee S, Jang D, Kim E, Moradi H, Han J. A graphics hardware-based accessibility analysis for real-time robotic manipulation. In: Proc. of the international conference on intelligent computing. 2005.
[34] Jang H, Moradi H, Lee S, Han J. A visibility-based accessibility analysis of the grasp points for real-time manipulation. In: Proc. of the IEEE/RSJ international conference on intelligent robots and systems. 2005.
[35] Jang H, Moradi H, Hong S, Lee S, Han J. Spatial reasoning for real-time robotic manipulation. In: Proc. of the IEEE/RSJ international conference on intelligent robots and systems. 2006.