國立臺灣大學電機資訊學院電機工程學系

碩士論文

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

# 基於搜尋演算法之規劃移動式機器手臂尋找與夾取遮蔽目標物

# Search Algorithm based Planning on Finding and Grasping of Occluded Target Object with a Mobile Robot Manipulator

林友騏

Yu-Chi Lin

指導教授：傅立成 博士

Advisor: Li-Chen Fu, Ph.D.

中華民國 103 年 7 月

July, 2014

# 國立臺灣大學碩士學位論文
# 口試委員會審定書

基於搜尋演算法之規劃移動式機器手臂尋找與夾取遮蔽目標物

Search Algorithm based Planning on Finding and Grasping of Occluded Target Object with a Mobile Robot Manipulator

本論文係林友騏君（學號 R01921003）在國立臺灣大學電機工程學系完成之碩士學位論文，於民國 103 年 7 月 22 日承下列考試委員審查通過及口試及格，特此證明。

口試委員：

[illegible] （簽名）
（指導教授）

[illegible]　　[illegible]

[illegible]　　[illegible]

系主任 [illegible] （簽名）

# 誌謝

首先，我要感謝我的父母，感謝他們對於我從小的栽培與付出。兩年碩士生涯即將結束，從碩一進入機器人組以來，經歷了無數次的拍片、DEMO，這之中雖然失敗的多而成功的少，然而，經由多次的經驗與對機器人功能更深入的掌握與有計畫性的建構基礎功能，現在的系統穩定度比起剛進入實驗室時已有著顯著的提升。感謝佩文學姊、元翰學長、建科學長在我碩一時給予的指導及共同合作的無數次DEMO。博士班的士桓學長、家明學長與同屆的庭升、亦修、仲達更是伴隨著我直到碩二，碩一的韶廷、致昂、東諺、靖螢讓實驗室變得更豐富、熱鬧與多元，我們一起準備 DEMO、作實驗、在實驗室聊天、吐苦水、講垃圾話、看世足賽，在碩二漫長的努力中一起互相扶持走過每次 DEMO 與最後的口試。

這篇論文的完成主要感謝傅老師的指導與督促，自進入實驗室以來，老師對研究做學問的方式深深影響著我，並提供實驗所需的軟硬體設備與完善的研究環境，此外，老師以現場 DEMO 對此篇論文驗證的方式使這篇論文除了理論的發展外實作部分也更加紮實，最後更是付出了寶貴的時間校正此篇論文。感謝口試委員宋開泰教授、胡竹生教授、羅仁權教授、范欽雄教授對論文的寶貴意見使這篇論文更臻完善。特別感謝士桓學長在此篇論文的初期階段時提供大量珍貴的討論界定論文的基礎架構，也感謝韶廷不遺餘力地協助完善艾薇兒機器手臂的運動模組以及時安在口試時的詳實紀錄。

感謝電機所同屆的佳勳、伯豪與奕廷以及其他電機同學平時的討論與無數次愉快的大咪後晚餐時間，感謝明理、仁德、聖化、定國、安陞等學長對我簡報技巧的精進提出各種寶貴的意見，能跟你們一起走過這兩年是我的榮幸。

碩士生涯中，我有幸擔當三門課的助教，首先感謝安陞學長讓我有機會在線性代數課程中擔任讀書會助教，在教學相長的風氣下教了一學期線性代數，也感謝傅老師的賞識，讓我擔任機器人學的助教，在東諺的協助下，完成了多次作業的批改與購置新型的機器手臂，最後感謝 C 類助教偉新在我擔任電子學三讀書會助教時的聯絡與代班協助，讓我很有成就感地上完一學期的電子學。

友騏

2014/7/29

# 中文摘要

隨著機器人與相關研究的發展由工廠漸漸走向辦公室及家庭環境，機器人在人類生活環境中服務與互動顯得十分重要，機器人應該要能做為人類生活中的幫手並提供各式服務，而在各項服務中，搜尋並遞送物品對使用者來說是很實用的一項技能。為了達成此功能，前人的研究多假設物品放在開放空間中，機器人在室內環境中以影像搜尋目標物並抓取之，過程中機器人規劃出最佳的路徑移動並且徹底觀察環境中的每個角落。然而，在真實世界中，物品很可能被其他物品所阻擋以致單純的影像搜尋不可能找到物品，因此，近期的文獻提及了以機器手臂移除障礙物以達成目標物搜尋之規劃，這些研究成果規劃機器人移動障礙物的順序並探索後方空間。

在本篇論文中，我們結合了以機器手臂移除障礙物與視覺主動搜尋之規劃，提出了一整合兩種方式之物品搜尋系統，機器人可以變換位置觀察環境，亦可以使用手臂移動障礙物來進行搜尋。本篇論文提出之規劃概念是以最小化預期搜尋時間以及在發現目標物後最小化目標物抓取時間為目標，使用 A*演算法並在搜尋空間內進行取樣以達成在有限時間內完成規劃。此外，本論文加入了視覺回授以確保機器人準確地執行計畫。最後，我們透過在書架內物品阻擋的環境中搜尋目標物的實驗來驗證本論文提出方法之優越性。

**關鍵字**：機器手臂操作, 搜尋物體規劃, 辦公室機器人

# ABSTRACT

As robots and robotic researches marching from factory to office and home, the ability of robot to interact with complex human-living environment becomes pivotal. To show its value, the robot should be able to do various tasks as an assistant in human-living environment. Searching and delivering object in indoor environment is one of the tasks practical to user. Previous study mainly focused on visual search of objects in indoor environment. The search is performed by a mobile robot which plans a best route to observe the environment and discover the target object. However, in real world, objects may be occluded by other objects or structures, which means pure visual search is impossible to find these targets. As a result, some recent works discussed the object search method by removing objects that block and hide the target object.

In this thesis, we propose an object search planning system that combines visual and arm manipulation search. The robot can either reposition one of the accessible object with its arm or move its platform to view the environment from a different position to discover the target object. The concept of planning is A* Planning which minimizes the expected time to discover the target and then the time to grasp the target in clutter after its discovery. Visual sensor feedback is included to assure the accuracy of each action performed by the robot. We evaluate the proposed approach with experiment in the scenario of object search in a shelf environment where objects may occlude or block access to one another.

**Keywords**: robot arm manipulation, object search planning, office robot

# CONTENTS

# LIST OF FIGURES

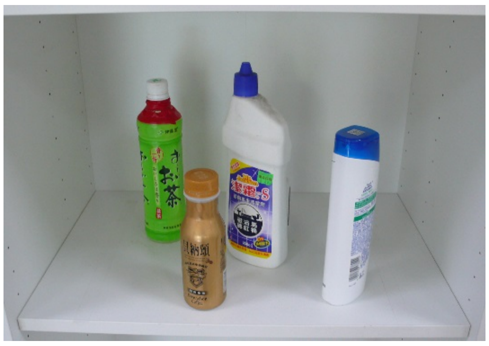
# LIST OF TABLES

# Chapter 1

# Introduction

## 1.1 Motivation

Researches on service robots boomed in this century, and the research society continues growing. Although most of service robots are not commercialized yet, researchers in labs from all over the world have made robot do various tasks. One of them is the object grasp and delivery with gripper. However, we demand more than that. In most of the robot demonstration, target object is assumed to be placed in an open space where the robot is able to see and grasp the target directly. Many researches have been done to analyze how the robot grasps objects in such scenario. In reality, however, objects are usually placed in clutter, which means the robot may not be able to grasp the target directly. Furthermore, in many cases, objects are placed in a fridge or cupboard and occluded by other objects, as shown in Figure 1.1. In both cases, the robot needs to plan the actions in order to find and grasp the target. In this thesis, we aim to solve the robotic object search planning problem by allowing the robot to manipulate objects and change camera viewpoint in order to gain more information about the space and to discover the target.


(a) Front view of the objects

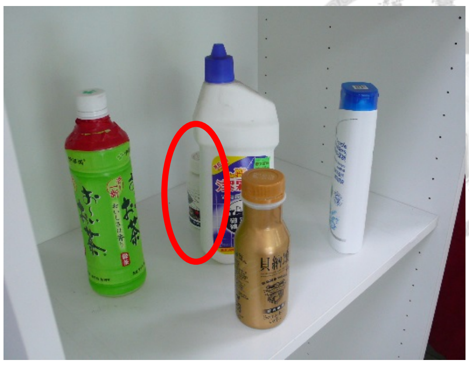
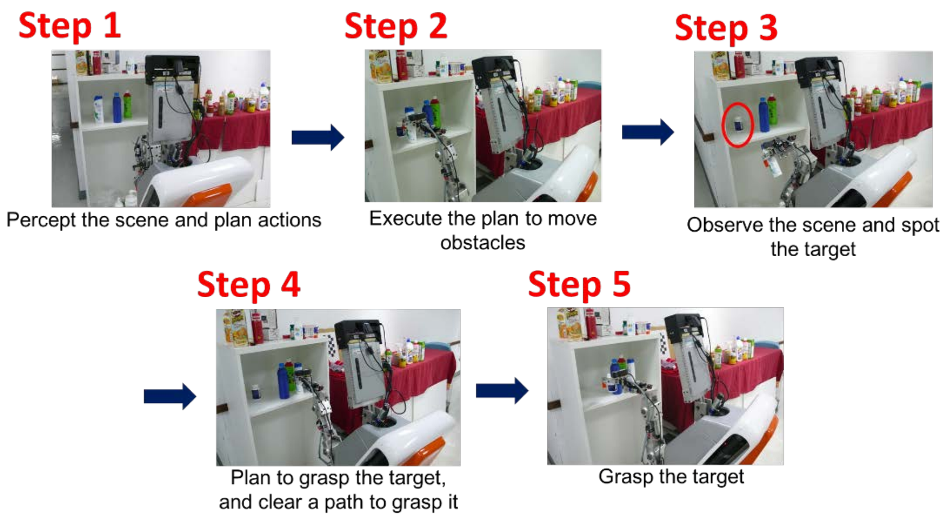

(b) The target object is occluded

Figure 1.1 Typical scene in object search by manipulation

## 1.2 Problem Formulation

We formulate the problem as the robotic object search and grasp planning in limited workspace like fridge or cupboard. In such cases, pure visual search is not enough to find the target, and the robot should also manipulate objects in the scene to reveal the target and clear a path to grasp it. In the first place, the target object is hidden behind several objects in the workspace. The robot percept the scene and register each object in view with 3D sensors, such as RGB-D camera, stereo camera or LiDAR. A series of actions are generated according to the perception result to discover and then grasp the target object. To achieve this, the robot can either move objects to new positions or change the robot pose to better approach certain object or view the workspace in each action in series.



Step 1
Percept the scene and plan actions
Step 2
Execute the plan to move obstacles
Step 3
Observe the scene and spot the target
Step 4
Plan to grasp the target, and clear a path to grasp it
Step 5
Grasp the target

Figure 1.2 General process of searching and grasping the target

When making a decision on which action to perform, the robot should consider both the "information gain" and the "accessibility effect" caused by the action. "Information gain" means how possible the target object will be revealed after this action. Intuitively, the robot tends to take actions with maximum information gain to quickly find the target. Nevertheless, this greedy strategy will not always work. The objects are placed in clutter, so the robot may not be able to access every object, including the one with maximum information gain, in the workspace. Furthermore, the robot should also consider where to place the object to benefit later actions. In other words, during the planning, the robot should also considers if the action makes other objects accessible or inaccessible. This effect of the action is summarized as "accessibility effect." The robot should be able to balance both considerations and plan a solution to find the target as soon as possible.

To better analyze such complex object search problem, several prior conditions need to be assumed in order to make the problem tractable. First, we assume there is only one hidden object in the workspace, which is the target and the geometry and object cloud of the target are known *a priori*. This assumption allows us to plan the search process based on object knowledge that can be visually acquired online, which are geometry and pose of every object except for the target in the workspace. Though it is possible to have more than one objects that are hidden, we may re-plan after spotting hidden objects other than the target. Thus, without loss of generality, we still assume the target is the only hidden object in order to make our solution clearer. Second, we let the robot arm always approach objects in horizontal direction. Third, all objects are standing on a horizontal plane and convex in outer contour of each slice parallel to the XY plane. Finally, the object's center of mass is assumed to be the geometric center of the object. These four conditions are set mainly for our explanation about how to select the grasp points throughout the planning so as to reduce the problem complexity.

## 1.3 Challenges

In this thesis, the workspace is assumed limited, which implies that the objects are impossible to be removed from the scene. Instead, they are moved to new poses inside the workspace and may affect subsequent actions and plan. The planning thus becomes a “reconfiguration planning” problem with great complexity. Furthermore, since an action influences following actions, a sensor feedback system is required for the robot to make sure that each action is performed accurately.

## 1.4 Related Works

In the field of object search by a mobile robot, the former researches focused on active visual search of objects. Such category of researches [1-5] assumes that the target object is placed in open place in the map where the robot can see directly. The planning determines where the robot should go to gain more information about the environment to find the target more quickly. Among those works, [2-4] further utilized the semantic meaning of objects and the spatial relations among objects to search the target object more quickly.

In recent years, numbers of results on “Search by Manipulation” have emerged. In real life, the objects are often placed in fridge or cupboard. The objects are occluded and blocked by other objects, which forces the robot to search the target by arranging the objects with the manipulator. It is worthwhile to mention that the works, [6] and [7], are pioneers in this field of study, but they addressed this problem in a quite different way. For example, [6] formulated the problem as a search of an object in multiple containers. They analyzed the probability of each container containing the target object with space constraint in each container and object co-occurrence relation. The robot then searches the container with high probability in containing the target by manipulation.

Furthermore, [7] and its final form [8] addressed this problem as an optimization in expected time to search the target in single container. Among all related works, this work correlates with our work most. In fact, despite that our problem setting shares some similarity to theirs, such as the target being the only hidden object and the expected search time being optimized, our problem scope becomes more general in the sense that several of their conditions are relaxed here. Specifically, our robot grasps the object in multiple direction, and observes the container more freely at number of poses. When facing large objects which is ungraspable, the robot may push it aside to discover the target object. Moreover, the robot is not allowed to permanently remove the objects from the scene, which turns out to be more challenging.

In former works [7, 8], under fixed camera pose assumptions, the occluded volumes may be divided into separated parts, and the search by manipulation planning can be done independently in these separated occluded volumes. By doing this, the robot can search the occluded target rather efficiently. In our scenario, the robot is allowed to observe the objects at different poses, which makes the robot more easily find the target, but there are inevitably many intersections of the occluded volumes observed from different camera positions. Therefore, the algorithm proposed by [7] fails to be applicable to our scenario.

Besides, [9] also discussed this problem by dividing the workspace into large grids and each object can only be contained in one grid. They also assumed a fixed camera position in planning. However, different from [7], they rearrange objects to a new grid position inside the container to explore the grids behind the grid which the object originally situated in. They model the object visibility and accessibility with single grid world, which is unrealistic. In their framework, the objects can be manipulated with push action to handle objects with large size.

## 1.5 Objectives

In this thesis, we propose a planner based on A* planning to optimize the expected time to search the target object, which is occluded by several visible objects, in a limited workspace like cupboard, shelf and fridge. We combine the active visual search and manipulation search in single framework. The robot can either move to different poses to observe the workspace or manipulate one of the objects. After the object's discovery, another planner to optimize the time required to grasp target is performed. In execution, the robot fixes its motion with sensor feedback to ensure the plan is accurately executed.

## 1.6 Thesis Organization

This thesis is organized as follows: In Chapter 2, we give an overview of our system architecture and hardware, and briefly introduce the tools and algorithms used in this thesis. In Chapter 3, the process to model workspace and object is discussed. The workspace and object models are input of the planner. In Chapter 4, the main contribution of this thesis, the search and grasp planner, is introduced in detail. After the planner, the sensor feedback in execution is also included in Chapter 4. Chapter 5 shows the experiment setting and result to evaluate the performance of the proposed approach to search and grasp target object. Finally, we conclude this thesis in Chapter 6.

# Chapter 2

# Preliminaries

## 2.1 System Overview

### 2.1.1 Hardware System

The hardware system is a mobile robot called ARIO, as shown in Figure 2.1. The robot is equipped with a differential drive and a 5-DOF gripper. Since each motor on ARIO has been paired with an industrial quality motor drive responsible for handling precise dynamic control, in our research we only need to implement high level position control to control various robot tasks. The sensors used in this thesis include an Xtion Pro RGB-D camera mounted on the head of the robot and a DS-325 near range RGB-D camera installed on the wrist of the robot arm. Xtion Pro on the head is used as the main sensor to percept the whole workspace, while DS-325 is served as a sensor feedback tool to increase accuracy of the grasp actions.

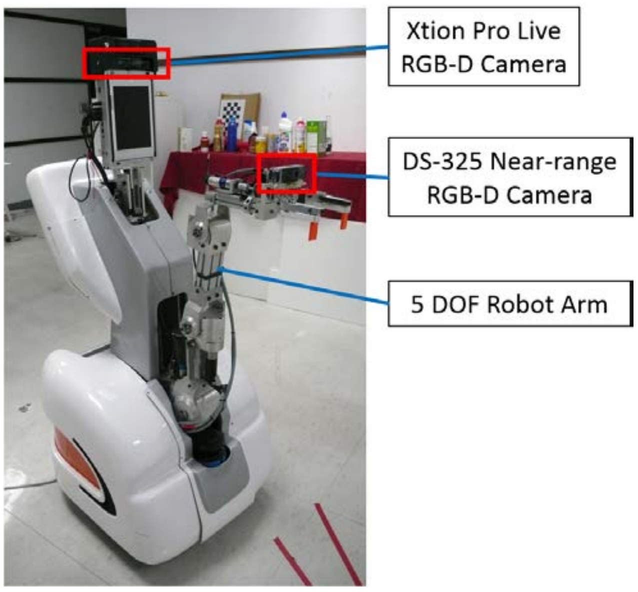


Figure 2.1 ARIO

### 2.1.2 Software System

The software system is composed of 4 parts, as shown in Figure 2.2. At first, perception module models the workspace and objects based on raw RGB-D data. The 3D perception in this thesis is implemented with Point Cloud Library (PCL)[10]. The search planning module then plans a series of search actions with the model derived from perception module. Execution module carries out the plan and fixes every action with sensor feedback to minimize execution error. After the target is spotted, the robot plans to grasp the target in clutter and executes the action with the execution module again.

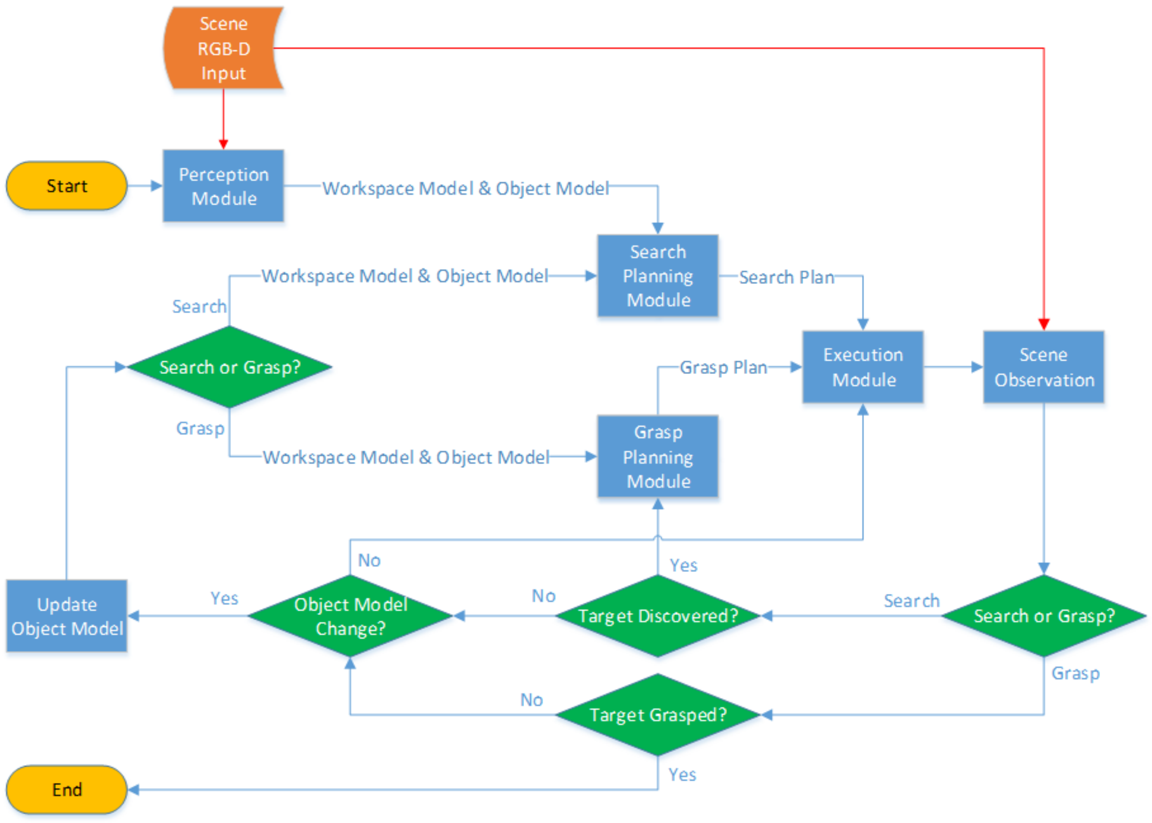


Figure 2.2 Flowchart of the software system

## 2.2 A* Algorithm

A* Algorithm [11] is a best-first search algorithm to plan a least-cost path from start node through multiple nodes connected by edges to a goal node. It is an improvement from breadth-first search algorithm to always check the most promising child node first to reduce the search space. A* algorithm keeps track of every node's cost from start node, which is the "past cost" of the node. Furthermore, A* algorithm estimates each node's cost to the goal node, also known as "future cost", and always finds the child nodes with the minimum past and future cost sum. If the future cost meets admissible heuristics, the A* algorithm can always find an optimal path from the start node to the goal node. A* algorithm has several advantages in planning. One of them is that A* algorithm can be implemented only with the connection relation between nodes and the heuristics for future cost estimation for each node. The algorithm does not have to know the neighborhood of goal node in the beginning of the search, which is very convenient in the cases where the position of the goal node in the graph is unknown or there are multiple goal nodes. Another advantage of A* algorithm lies in the memory usage and computational complexity. Since A* algorithm always expands the child nodes with the minimum past and future cost sum, it visits fewer nodes than breadth-first search and does not have to keep the full graph with all nodes. Instead, the robot only memorize nodes being traversed and their child nodes. Such characteristics is especially advantageous when the growing graph costs much.

## 2.3 Random Sample Consensus (RANSAC)

Random Sample Consensus [12] is a method to estimate parameters of a model from a set of data which contain outliers. The method first randomly samples some data from the whole dataset and then fit them with the model, find parameters of the model, and then check the number of data that can be described by the model. If the number is high, it is a good fit and hence the model parameters are recorded. The model can be refined by taking more iteration of sampling. This method has advantages in finding the model parameters when there are significant amount of outliers.

## 2.4 Object Feature Detector & Descriptor

In execution, the robot may move to several locations and view the workspace at different poses. To register the same object seen at different viewpoints, we match features extracted from the objects. To find feature correspondence in point cloud data, the most intuitive method is feature detector and descriptor based on the local 3D geometry of the object. However, in real world, many objects have similar shape, as shown in Figure 2.3. Therefore, we also include the color or gray scale intensity cue to find feature correspondence. After careful evaluation, the 5D Harris Corner feature detector is adopted in this thesis. After feature extraction, each feature point is described in ColorSHOT descriptor for matching. Though feature detectors and descriptors considering both geometry and color are more computationally expensive, they are more robust under uniformity in either color or geometry.

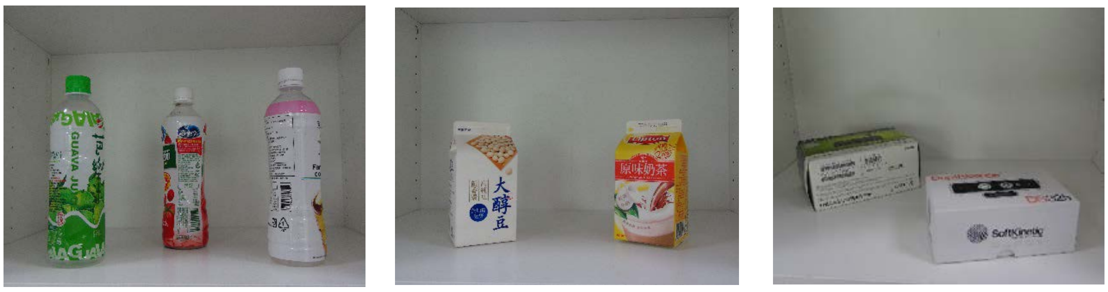

(a) (b) (c)

Figure 2.3 Examples of objects similar in shape

### 2.4.1 5D Harris Corner Detector

5D Harris corner feature detector is an combination of 2D Harris which is proposed by Harris and Stephens [13] to detect corners in grayscale image and 3D Harris that detects geometric corner in 3D surface [14]. Since it merges 2 Harris corner detector, the input data of 5D Harris must be dense structured point cloud, whose every point $p$ corresponds to a pixel $(u,v)$ in the image plane. The result of 5D Harris considers the corner both in geometry and grayscale intensity.

2D Harris feature detector detects pixels in the grayscale image that have large variation in intensity in both directions. By checking the difference between the target pixel and its neighboring pixels, we know how large the intensity variation is in the target pixel's neighborhood. The 2D Harris corner is computed by finding the eigenvalues of the sum of intensity gradient covariance matrix, $S$, over a neighborhood window $W$.

$$S = \sum_{(u,v)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \sum_{(u,v)\in W} I_g I_g^T \tag{2.1}$$

$$I_g = \begin{bmatrix} I_x & I_y \end{bmatrix}^T \tag{2.2}$$

where $(u,v)$ is a pixel inside the window, and $\begin{bmatrix} I_x & I_y \end{bmatrix}^T$ is the intensity gradient at each image coordinate $(u,v)$ along $x$ and $y$ directions, respectively.

In 3D Harris, the grayscale intensity is replaced by surface normal at the point on the surface. The neighborhood $(NB)$ of a point is defined as a sphere centered at the point with user-defined radius. The 3D Harris covariance matrix is calculated as:

$$S = \sum_{p\in NB} N_p N_p^T \tag{2.3}$$

$$N_p = \begin{bmatrix} N_x & N_y & N_z \end{bmatrix}^T \tag{2.4}$$

where $N_p$ is the normal vector at point $p$. In 5D Harris, we concatenate $N_p$ and $I_g$ to form the vector containing derivatives in both XYZ space and grayscale image plane, namely,

$$D_p = \begin{bmatrix} N_p^T & I_g^T \end{bmatrix}^T \tag{2.5}$$

The sum of covariance matrix over the neighborhood is defined as:

$$S = \sum_{p\in NB} D_p D_p^T \tag{2.6}$$

with the covariance matrix sum, each point can be scored with the Harris response function:

$$H(p) = \det(S) - trace^2(S) \tag{2.7}$$

### 2.4.2 ColorSHOT Descriptor

Generally speaking, we need a descriptor to show if an extracted feature point is similar to other feature points, and ColorSHOT [15, 16] is one. ColorSHOT is an improvement of the original SHOT (Signature of Histograms of OrienTations) descriptor, which only focuses on shape feature. ColorSHOT considers both shape and texture (color) of the neighborhood of the feature point. In ColorSHOT descriptor, all neighboring points are mapped to a reference frame based on the Eigenvalue Decomposition of the scatter matrix of the points which are neighbors of the feature point. Note that these points are divided by grids in spherical coordinates.

The descriptor describes the local shape feature by defining a histogram in each grid with bins representing different values of inner product between the normal vector of the point and the feature point vector. For color part of the descriptor, the histogram is composed with of bins of the distances of points to the feature point in RGB space expressed as:

$$l_C(C_f, C_p) = \sum_{i=1}^{3} \left| C_f(i) - C_p(i) \right| \tag{2.8}$$

where $C_f$ and $C_p$ are three dimensional RGB color vectors at the feature point and an arbitrary point, respectively. The distance is actually a 1-norm in RGB space. The color space can be replaced with CIELab or HSV. Finally, the descriptor vector is constructed by concatenating the shape and color descriptor vectors, as shown in Figure 2.4.

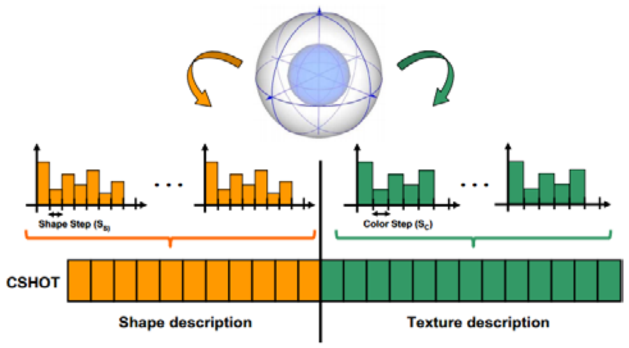


Figure 2.4    ColorSHOT Feature Descriptor[16]

# Chapter 3

# Workspace Model and Object Models

Before planning, the robot should identify the workspace boundary and segment the scene to find object models in the workspace, as shown in Figure 3.1. An object database is also generated online to keep the object's geometry model, pose, and available action to be applied to the object. The workspace and object model provide fundamental information for later planning and runtime feedback in execution.

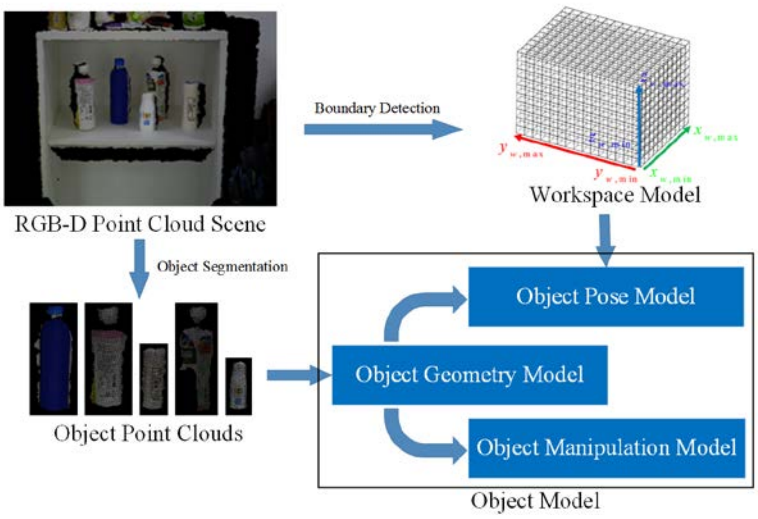


Figure 3.1 Workspace Model and Object Models

## 3.1 Workspace Model

The workspace is where the robot interacts with objects and discovers the target. Since the scenario in this thesis is searching target in limited workspace like cupboard, shelf and fridge. We model the workspace as a cuboid divided into voxel grids, as shown in Figure 3.2. The size of the voxel is adjustable to tradeoff between the precision and the computational cost. In this thesis, the voxel’s dimensions are 1cm x 1cm x 1cm, and it is the unit of the workspace. Each voxel in the workspace is either unknown, empty or occupied by an object.

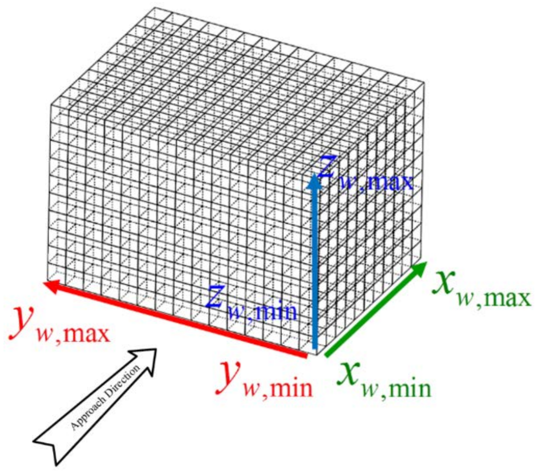


Figure 3.2 Workspace and its dimension

To identify the dimensions of the workspace, there are a few approaches, and one of them is to record the information in the global semantic map so that the robot knows the exact location and dimension of the workspace. In this thesis, however, we assume that initially the robot only knows the rough location of the workspace in the map and the workspace lies within the robot view, so that local perception is required to have more accurate dimension and location of the workspace for the sake of planning. We achieve this by first identifying the upper, lower, left and right boundaries of the workspace with depth gradient feature. Then, referring to Figure 3.2, we first move the robot to the front of the workspace, being aligned with the midline of the workspace, and face the workspace directly. This pose is the starting pose of the robot in our planner. Within the side boundaries, we take the shallowest and deepest points as the shallower and deeper boundaries of the workspace. As a result, the whole workspace can be regarded as a cuboid whose three sides are aligned with the three axes in the global coordinate frame. In the sequel, we denote the left and right bounds as $\left(y_{w,\max}, y_{w,\min}\right)$, the upper and lower bounds as $\left(z_{w,\max}, z_{w,\min}\right)$, and deeper and shallower bounds as $\left(x_{w,\max}, x_{w,\min}\right)$.

## 3.2 Object Geometry Model

After removing all the boundary points of the workspace, the rest point cloud are objects inside the workspace. We segment each object with Euclidean clustering approach to obtain each object point cloud $P = \{p_1, p_2, ..., p_n\}$, where $p_1, p_2, ..., p_n \in R^3$ are points of the point cloud. Since the object is assumed to be convex, we calculate the convex hull of the point cloud $P$ and compose the object's geometry model with voxels whose center is inside the hull. The geometry model is thus a voxel cloud $V$. The occluded part of the object may not be modeled correctly. However, during planning, the robot moves to different positions and view the object from different perspectives. At that time, the robot is able to update the object model and re-plan if necessary.

### 3.2.1 Geometric Primitive Model-based Object Modeling

For objects which can be fitted well with a primitive geometry model, we can apply Random Sample Consensus (RANSAC) on the point cloud to get a precise object geometry model with model-based approach. Though we don't know which model suits the object beforehand, we can still fit the object with predefined model and check its similarity to the model. Only fittings with high similarity are applied.

One geometry model is the cylinder model. Here, we focus on cylinder vertical to the horizontal plane. Since cylindrical objects like bottles, cans, and cups are common in daily life, it is the top choice of geometry model. The cylinder model is expressed as:

$$Cylinder = \left\{P_C \in R^3 \middle\| d\left(P_C, L\right) - r \right| < \varepsilon \right\} \tag{3.1}$$

where $P_C$ are points that fit the cylinder model, $L \in R^3$ is the central axis of the cylinder, $r$ is the radius, and $\varepsilon$ is the error tolerance of the model. To test if the model well describes the object's geometry, we find the root-mean-square error of radius $r$ to the distance of every point in the object point cloud $P$ to the central axis $L$:

$$E_{rms} = \sqrt{\frac{\sum_{i=1}^{n}\left(d\left(P,L\right)-r\right)^2}{n}} \tag{3.2}$$

if $E_{rms}$ is lower than certain threshold, which is determined by the magnitude of the sensor's random error, the object is modeled as cylinder. Voxels that are inside the cylinder form the object model voxel cloud $V$. Modeling a cylindrical object as cylinder has more advantages than getting a precise model, and one of them is that the object has only one pose, which reduces the computational cost in deriving its manipulation model.

Besides cylinder model, we also fit object clouds with cuboid model. We first extract the plane which is parallel to $z$ axis and contains the most points from the object cloud, called major side plane. Second, we iteratively extract planes from the object cloud. Finally, we calculate the proportion of the points that belong to the planes vertical or parallel to the major side plane to score the similarity of the object cloud to a cuboid. The cuboid model is the object cloud bounding cuboid in the orientation determined by the normal vector of the major side plane.

According to the object's size, we divide the objects into 2 categories: small and large. If the robot gripper can completely surround the object, the object is small; otherwise, it is considered large. By checking the radius of the minimum bounding circle of the object's projection to the XY plane, we can classify the objects as small or large ones. The reasons to classify objects by its size lie in the difference in manipulation models of the two kinds, which will be discussed it in section 3.5.

## 3.3 Object Pose Model

In this section, we give a brief introduction about how an object's pose is described in this thesis. As stated in Chapter 1, we assume that all objects are standing on a horizontal plane. Therefore, the object's pose is special Euclidean group $SE(2)$, which contains the object's position $(x_o, y_o)$ on the table, and the orientation $\theta_o$ rotating against the z axis. Since the workspace is discretized to be voxel grids, the position $(x_o, y_o)$ is mapped to the voxel in the bottom layer of the workspace. For object without geometric primitive model, $(x_o, y_o)$ are the $x$ and $y$ component of the Euclidean mean of the voxels in the geometry model. For cylindrical objects, $(x_o, y_o)$ are the $x$ and $y$ component of the centroid of the cylinder model.

Since there is little difference for planning and manipulation in slight orientation change, the object orientation unit is set to be 5 degrees in this thesis. The initial orientation of the object first observed by the robot is set as 0. For cylindrical objects, there is no or little difference in different orientations, so we let the orientation be always 0.

## 3.4 Object Manipulation Model

In this section, we discuss how objects are manipulated by the robot and how objects are influenced by the manipulation. The object manipulation model models the effect of manipulation based on the manipulation type and the object pose relative to the gripper with absence of obstacles. There are three types of manipulation defined in this thesis, which are "grasp," "place" and "push-aside."

### 3.4.1 Grasp Manipulation

First manipulation is "grasp". We limit the grasp direction to horizontal direction, as shown in Figure 3.3, so we can define a grasp as:

$$GRASP = \{\theta_{init}, p_g\} \tag{3.3}$$

where $\theta_{init}\left(0^{\circ} \leq \theta_{init} < 360^{\circ}\right)$ means the initial object orientation relative to the gripper, which can also be regarded as the arm's approach direction to the object, and $p_g = \left(x_g, y_g, z_g\right) \in R^3$ is the grasp point expressed in object coordinate. For non-cylindrical objects, the object's orientation may vary after manipulation. This effect will be discussed in detail in section 3.5.1. We assume the grasp point does not change and it serves as rotation center when grasping. The final object orientation is determined by $GRASP$ and is denoted as $\theta_{fin}\left(GRASP\right)$. If the object's width in final pose $\theta_{fin}$ is below the minimum or above the maximum spread of the gripper, the object cannot be grasped from $\theta_{init}$. As a result, we can filter all $GRASP$ with the above criterion and get a set of feasible $GRASP$, denoted as $GRASP_f$. We define object grasp model ($GM$) to describe the effect of each feasible $GRASP$ to the object:

$$GM\left(GRASP_f\right) = \left\{\theta_{fin}\right\} \tag{3.4}$$

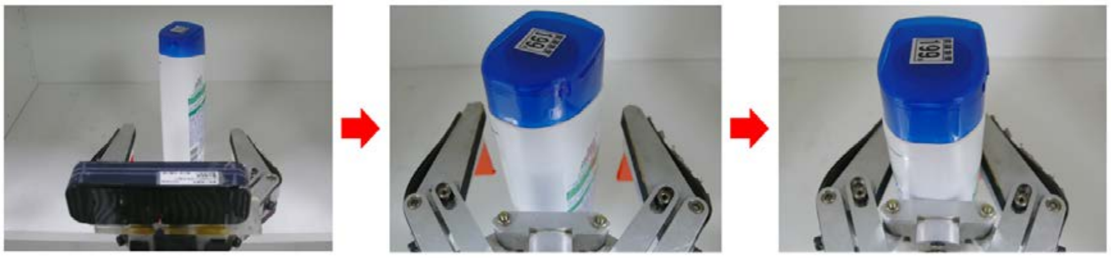


Figure 3.3 Grasp manipulation

### 3.4.2 Place Manipulation

The object grasped will be released and placed somewhere in the workspace. A place manipulation is defined as:

$$PLACE = \left\{\theta, p_g\right\} \tag{3.5}$$

where $\theta$ is the relative pose of the object to the gripper, which equals the final relative pose $\theta_{fin}$ in $GM$, and $p_p$ is the place position with respect to the object centroid, which is the same as the grasp position $p_g$. The object place model ($PM$) is therefore the object pose relative to the gripper:

$$PM\left(PLACE\right)=\left\{\left(x_{fin},y_{fin},\theta\right)\right\} \tag{3.6}$$

### 3.4.3 Push-aside Manipulation

For a large object, it may be placed at a position where it is not prehensile from all available directions. Therefore, we design push-aside manipulations for the robot to move large objects by pushing it aside, as shown in Figure 3.4. A push-aside manipulation can be modeled as a three degrees-of-freedom (DOF) movement of a pushing plane. In ARIO robot, the arm pushing trajectory is fixed to an arc, so the DOF is reduced to 1. Therefore, in this thesis, we define a push-aside manipulation as:

$$PUSH\_ASIDE=\left\{\left(x_{init},y_{init},\theta_{init}\right),DIR\right\} \tag{3.7}$$

where $\left(x_{init},y_{init},\theta_{init}\right)$ is the initial pose of the object relative to the gripper, and $DIR$ is the push direction, either clockwise or counterclockwise. Unlike grasp manipulation, push-aside manipulation can be applied to an object in every orientation, so all push-aside manipulations are feasible. The object push-aside model ($PaM$) is:

$$PaM\left(PUSH\_ASIDE\right)=\left\{\left(x_{fin},y_{fin},\theta_{fin}\right)\right\} \tag{3.8}$$

where $\left(x_{fin},y_{fin},\theta_{fin}\right)$ is the final pose of the object in the gripper coordinate. The pose change effect of the push manipulation is detailed in section 3.5.2.

From the manipulation model, we get a mapping from initial pose to final pose in gripper coordinate. Thus, the object movement in the global coordinate frame can be determined if we know the gripper motion in that coordinate.

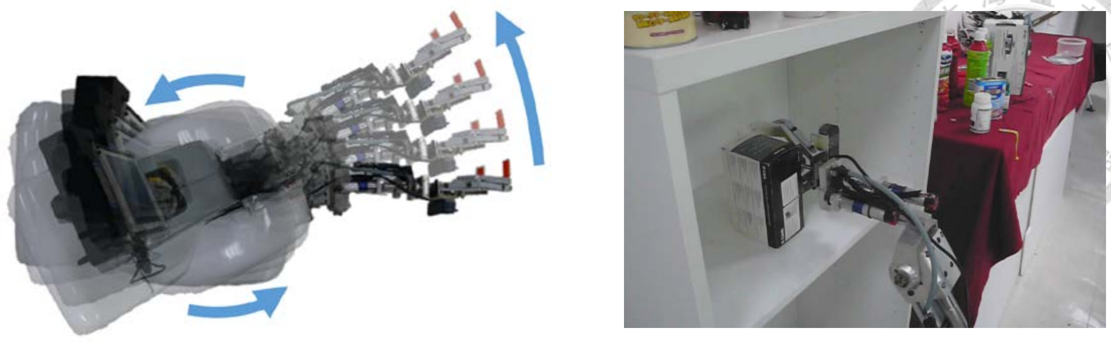


Figure 3.4　　Push-aside manipulation

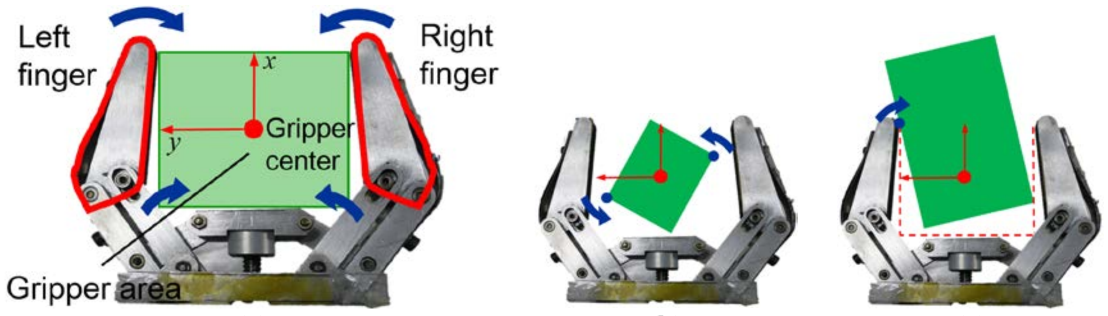
## 3.5　Object Pose Prediction after Manipulation

For non-cylindrical objects, object pose may change after manipulation. To plan actions to search for the target object, the robot should know how objects' pose changes in each manipulation beforehand. In the following section, we discussed how we predict the pose variation effect caused by "grasp" and "push-aside", respectively.

### 3.5.1　Grasp Manipulation

The pose variation in grasp manipulation is a gripper dependent. The shape and the closing motion of the gripper is crucial in predicting the final pose after manipulation. Figure 3.5(a) shows the gripper of ARIO. The gripper area, marked as the green rectangular area in Figure 3.5(a), is the area to contain the object in the gripper. Inside the gripper, we define gripper coordinate frame centered at the center of the gripper area, as shown in Figure 3.5(a). We assume that the object pure rotates against the gripper center during grasping. When grasping small objects, the gripper moves to align its center to the object center at the grasp height, as shown in Figure 3.5(b). Whereas in grasping large objects, the gripper may not be possible to align its center to the object center at the grasp height, so the gripper will try it best to fit the object into it, as shown in Figure 3.5(c). If a large object cannot fit into the gripper to occupy more than half of the gripper area in certain relative pose to the gripper, the approach direction is considered invalid.

In order to predict the object rotation, we first find the contact point first touched by the gripper in current relative object pose to the gripper. If the contact point is in first or third quadrant of the gripper coordinate, the object rotates clockwise, as shown in Figure 3.5(c). If the contact point is in second or fourth quadrant of the gripper coordinate, the object rotates counterclockwise, as shown in Figure 3.5(b). If there are multiple contact points in neighboring quadrants, the object reaches its final pose. In conclusion, by following the above procedures, we can derive the final relative object pose given an initial relative object pose to the gripper and a grasp height.



(a) (b) (c)

Figure 3.5 Grasp model

## 3.5.2 Push-aside Manipulation

Similar to grasp manipulation, the pose variation relative to the gripper in a push-aside manipulation is assumed to be a pure rotation against the object center of mass, which is also the object geometric center as we assumed. In push-aside manipulation, we also divides the object into 4 quadrant, as shown in Figure 3.6. The $x$ axis of push-aside coordinate frame is parallel to the arm, and $y$ axis is vertical to it. If the contact point is in first or third quadrant, the object rotates clockwise. If the contact is at second or forth quadrant, the object rotates counterclockwise. If there are multiple contact points in two quadrants of the same side, the object reaches final relative pose to the arm. To ensure the object converges to final relative pose after pushing, we let the robot push each object for at least 10 degrees.

In order to prevent the object sliding toward $+x$ direction in the push-aside coordinate frame and escaping from the push-aside manipulation, we equip the gripper with a pair of blocking plate to confine the object, as shown in Figure 3.6. As a result, the global motion of the object in push-aside manipulation is thus considered as a combination of pure translation along the arm's circular trajectory and a pure rotation against the object center.

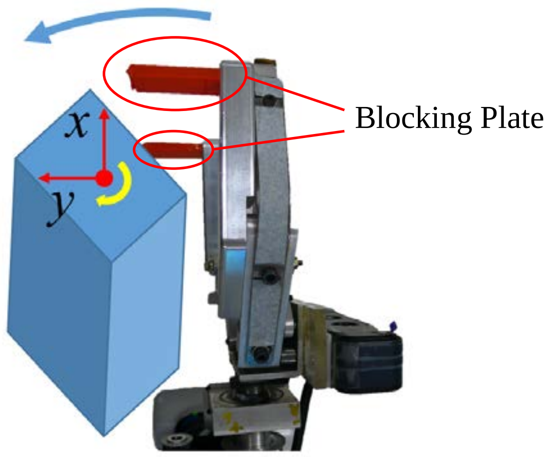


Figure 3.6 Push-aside model

# Chapter 4

# Planning and Sensor Feedback in Execution

## 4.1 Robot Pose Sampling

In the planner, the robot can change its pose to gain better view of the workspace and manipulate objects from different directions. To reduce the computational cost of the planner, we identify a few positions $\left(x_{r_i}, y_{r_i}\right)$ in front of the workspace as the robot pose candidates, referring to Figure 4.1, which are expressed as follows:

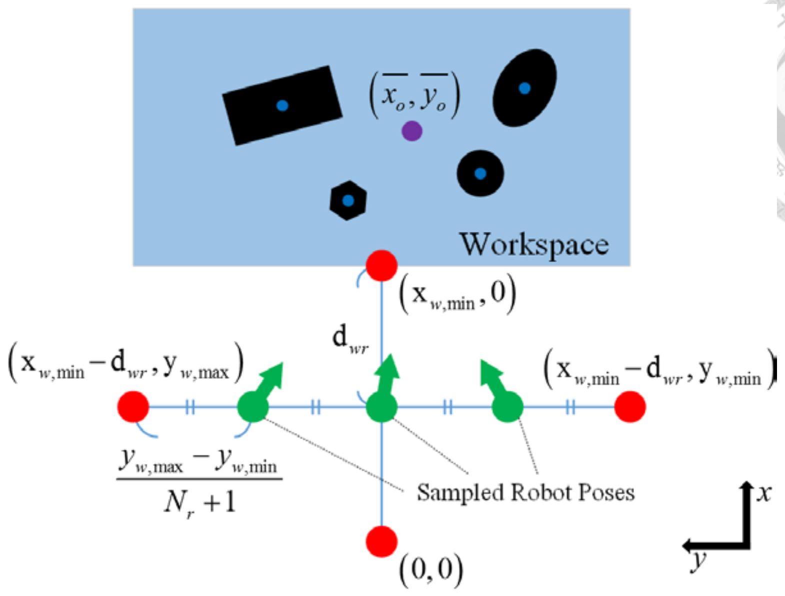
$$\begin{aligned} x_{r_i} &= x_{w,\min} - d_{wr} \\ y_{r_i} &= \frac{i}{N_r+1} y_{w,\min} + \frac{N_r+1-i}{N_r+1} y_{w,\max} \end{aligned}, \; i = 1, \ldots, N_r \tag{4.1}$$

where $N_r$ is the total number of robot pose candidates, and $d_{wr}$ is a fixed distance between the robot and the shallower bound of the workspace, which is determined by the robot arm's workspace and also the view angle of the camera. Note that the robot orientation is determined by the position to make the robot face the objects, i.e.

$$\theta_{r_i} = \tan^{-1}\left(\frac{\overline{y_o} - y_{r_i}}{\overline{x_o} - x_{r_i}}\right) \tag{4.2}$$

where $\left(\overline{x_o}, \overline{y_o}\right)$ is the Euclidean mean of all visible object's positions in the beginning of planning. Finally, all robot pose candidates can be expressed as a set:

$$POSE_r = \left\{ p_{r_i} \middle| p_{r_i} = \left(x_{r_i}, \mathrm{y}_{r_i}, \theta_{r_i}\right),\ i = 1, \ldots, N_r \right\} \tag{4.3}$$



$(\bar{x}_o, \bar{y}_o)$
Workspace
$(x_{w,\min}, 0)$
$d_{wr}$
$(x_{w,\min} - d_{wr}, y_{w,\max})$
$(x_{w,\min} - d_{wr}, y_{w,\min})$
$\frac{y_{w,\max} - y_{w,\min}}{N_r + 1}$
Sampled Robot Poses
$x$
$y$
$(0,0)$

Figure 4.1 Robot pose candidates ( $N_r = 3$ )

## 4.2 Search Planning

At the beginning of planning, the robot first locates possible poses of the target in the occluded part of the workspace. With the aforementioned workspace and object models, the robot knows the workspace boundaries and the way to interact with each object. With such prior knowledge, the robot plans the search actions to find the target object. That is, given the proposed planner, the robot finally generates a plan which contains a series of these search actions.

### 4.2.1 Actions in Search Planning

There are three types of action: “Platform Move”, “Grasp and Place” and “Push-aside”. First, “Platform Move” means that the robot moves to some predefined robot pose candidates to observe the workspace in different direction. Second, “Grasp and Place” is that the robot grasps an object, observes the workspace when the manipulated object is temporally removed from workspace, and places it in new object pose. Third, during “Push-aside” action, the robot pushes a large object to a new object pose and then observes the workspace.

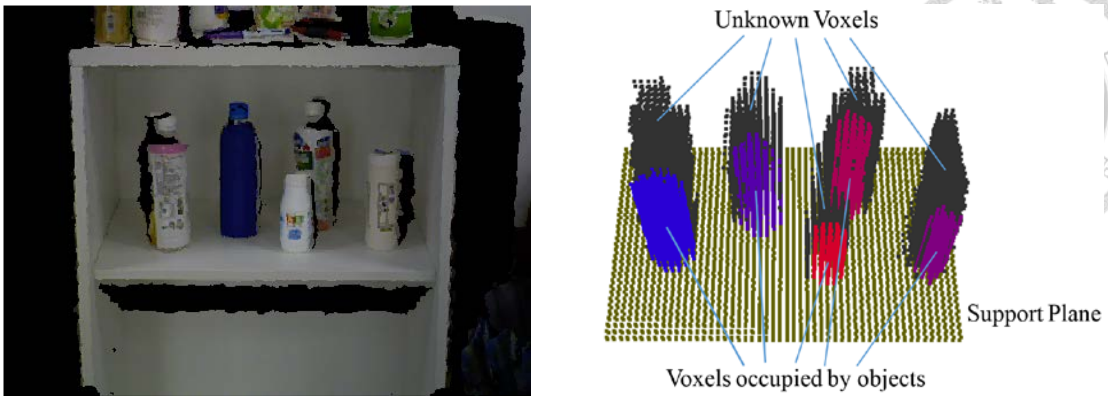
These three types of actions differ not only in the robot motion, but also in the observation of the workspace. The “Grasp and Place” action has advantage over other two actions in observing the workspace with one object being removed. Further, the “Platform Move” allows the robot to change its viewpoint, and the effect is like moving several objects’ positions relative to the robot in one action. Finally, the “Push-aside” action makes large objects easier to be manipulated by the robot, and find the target quicker.

Table 4-1 Comparison of action types

| Action Type | Observation | Advantage | Disadvantage |
|---|---|---|---|
| Grasp and Place | Between grasping and placement | Observe the workspace with one object removed from it | Under the risk of manipulation error |
| Platform Move | After movement | Observe from different position without touching the objects | Hard to find all occluded target poses |
| Push-aside | After pushing | Move ungraspable large object | Need much free space for the robot to sweep over |

## 4.2.2 RGB-D Camera Perception Model

The sensor used in this thesis for planning is an RGB-D camera mounted on the head of the robot. The perception model for RGB-D camera is pinhole camera model. We can project every voxel in the workspace to the image plane, which is a 480 by 640 matrix, with the pinhole model. After object modeling, the voxels occupied by objects are marked as each object category, as shown in Figure 4.2. If a voxel and an occupied voxel are projected to the same pixel and the voxel is farther than the occupied voxel, then the voxel is blocked by an object, so it is marked as “unknown.” The rest of the voxels are empty voxels, as shown in Figure 4.2.



(a) RGB-D point cloud scene (b) Objects and unknown voxels

Figure 4.2 Objects and unknown voxels

### 4.2.3 Hidden Target and Reveal Condition

The geometry of the target is assumed to be known before planning. We can easily get the object model of the hidden target. To find possible pose of the hidden target, we check if the target geometry model occupies only unknown voxels at each pose $(x, y, \theta)$. If so, the pose is a possible hidden pose. We denote the set of possible hidden target pose as $HT$ and the number of possible hidden target pose as $N_{HT}$.

The reveal condition specifies how the object is considered as revealed. It may vary with different positions and altitudes of the camera relative to the workspace. In our cases, the robot arm and workspace are lower than the camera. Therefore, we define the reveal condition as follows: A possible hidden target pose is revealed if one of the voxels of its geometry model that is higher than the half height of the target object is observed by the camera. Note that the revealing of a possible hidden object pose is an irreversible event. In other words, a possible hidden target pose which is revealed is permanently revealed. To track the condition of each possible hidden target pose, we define the "reveal state" ($RVS$) as:

$$RVS = \left\{ rvs(ht_i) \middle| ht_i \in HT,\ i = 1, \dots, N_{HT},\ rvs(ht_i) = \begin{cases} 1 & \text{, if } ht \text{ is revealed} \\ 0 & \text{, otherwise} \end{cases} \right\} \quad (4.4)$$

### 4.2.4 Object Accessibility in Search Planning

Object accessibility describes whether each object can be manipulated by the robot at each robot pose candidate. The accessibility condition of an object for a robot pose is one of the three: out of robot arm's workspace, in workspace but blocked by obstacles, or manipulable. We can encode the accessibility as:

$$a_{ij} = \begin{cases} -1 & \text{, object } j \text{ is out of workspace at robot pose } i \\ 0 & \text{, object } j \text{ is in workspace but blocked by obstacles at robot pose } i \\ 1 & \text{, object } j \text{ is manipulable at robot pose } i \end{cases} \quad (4.5)$$

The obstacles include other objects, possible hidden target, and the workspace boundaries except the frontal frame at the shallower bound. We check collision not only between the robot arm and obstacles, but also the manipulated object and obstacles. By identifying the accessibility of all the objects, the planner knows all the manipulation choice it has at the current state. The "object accessibility state" for search planning can be defined as:

$$ACS_S\left(POSE_o, RVS\right) = \left\{a_{ij} \,\middle|\, i = 1,\ldots,N_r,\ j = 1,\ldots,N_o\right\} \quad (4.6)$$

where $N_r$ and $N_o$ are numbers of robot pose candidates and visible objects, respectively. The accessibility state is determined by the poses of the objects, reveal state of the hidden target, the location of all robot pose candidates, workspace boundaries and the robot arm kinematics. Since the latter three are invariant during planning, the object accessibility state is a function of the set of all object poses, denoted as $POSE_o$, and the reveal state.

### 4.2.5 Graph in Search Planning

The search planning is regarded as path finding problem in a directed graph. The node of the graph is a state in the planner which is the combination of the current robot pose, object poses, and the reveal state of the hidden target, as shown in the following expression:

$$Node = \left\{ p_{r_i}, POSE_o, RVS \middle| p_{r_i} \in POSE_r \right\} \tag{4.7}$$

A node is connected to its parent node with an edge which represents a feasible action. With an arbitrary node, denoted as $Node_k$, and a feasible action in $Node_k$, denoted as $A$, we can define an edge from $Node_k$: $Edge(Node_k, A)$. By changing the definition of the cost function associated with the edge, we can change the behavior of the planner. In this thesis, we adopt the definition from [7] to minimize the expected time to reveal the target. The edge cost function is defined as:

$$f_e\left(Edge\left(Node_k, A\right)\right) = \frac{\left(\Delta n_{HT}\right)_A}{N_{HT}}\left(T_{\left(Node_1, Node_k\right)} + T_A\right) \tag{4.8}$$

where $\left(\Delta n_{HT}\right)_A$ is the number of possible target poses revealed due to action $A$, and $T_A$ is the time required for the action. $Node_1$ is the initial node of the planner, and $T_{\left(Node_1, Node_k\right)}$ is the time required from the initial node to the node $Node_k$.

### 4.2.6 A* Object Search Planning

As stated in section 2.2, A* planning can find the optimal path with minimum edge cost sum to travel from the initial node to the goal node in the graph by estimating the future cost of nodes to the goal node with admissible heuristics. Therefore, before planning, we should define the initial node, the goal node, the past cost of each node, and the estimation of future cost.

As shown in (4.7), a node contains the information of robot pose, object poses, and reveal state. The initial node is simply the state before the robot starts planning. In the initial node, the robot pose is $(0, 0, 0)$ in global coordinates, objects are untouched and all possible hidden target poses are not revealed. The goal node of the planning is the node that all possible hidden target poses are revealed. With a path from the initial node to an arbitrary node $Node_k$, we have the past cost of $Node_k$ and is able to estimate the future cost of $Node_k$. The past cost of each node is the sum of edge costs from initial node to the node:

$$f_p = \sum_{t=1}^{k-1} f_e\left(Edge\left(Node_t, A_t\right)\right), A_1 \sim A_{k-1} : \text{Current Plan} \tag{4.9}$$

As for the future cost, [7] estimated the future cost of an arbitrary node $Node_k$ as:

$$f_{fs} = \left(\frac{n_{HT,Node_k}}{N_{HT}}\right)\left(T_{(Node_1,Node_k)} + T_{A,\min}\right) \tag{4.10}$$

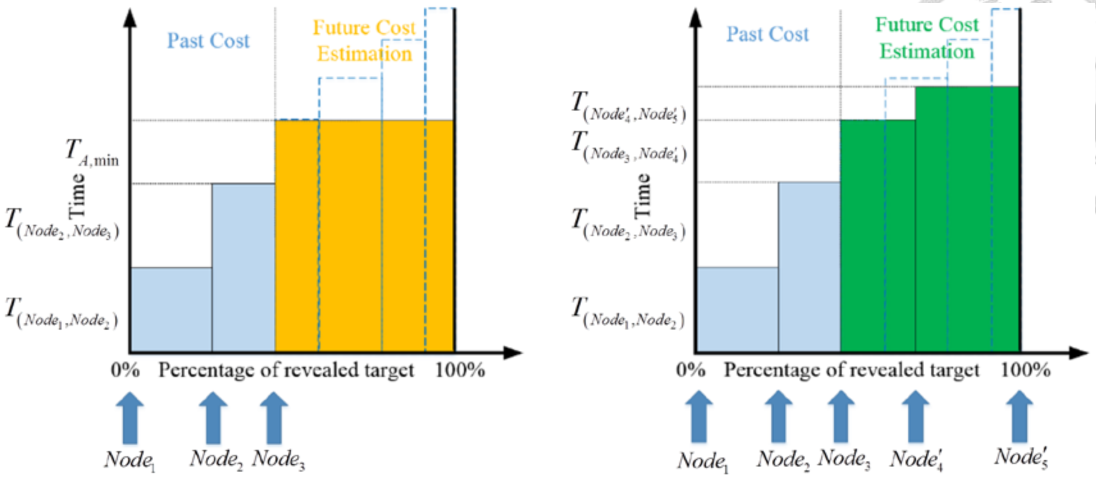
where $n_{HT,Node_k}$ is the number of remaining hidden target poses and $T_{A,\min}$ means the minimum time required among all feasible actions in $Node_k$. The estimation expects all the remaining hidden target poses to be revealed with single action which takes the minimum time in all feasible actions of the current node. This optimistic estimation surely underestimates the future cost and is admissible. We call this future cost estimation as "single-action future cost estimation", denoted as $f_{fs}$. Figure 4.3(a) shows the illustration of the past cost and single-action future cost estimation. The area of the bars represents the expected time to reveal the target. We also draws the true optimal plan with blue dashed line.

The performance of A* algorithm relies on the future cost estimation. If the estimated future cost is closer to the real future cost, the nodes that are considered will be fewer which then results in shorter search time. The exact future cost of each node is the expected time to reveal all hidden target poses from the node. If we assume that the robot can observe the workspace from all robot pose candidates at the same time, the objects can be removed from the workspace without placing them back, and all objects are always accessible, and then the optimal plan is to always remove the object which occludes the most hidden target poses. This greedy plan, which simply ignores all constraints to choose an action, surely underestimates the expected time to reveal all target poses. We call this estimation as “greedy future cost estimation” and is defined as:

$$f_{fg} = \sum_{Node_i = Node'_{k+1}}^{Node'_n} \left( \frac{n_{HT,Node_{i-1}} - n_{HT,Node_i}}{N_{HT}} \right) \left( T_{(Node_1,Node_k)} + T_{(Node_k,Node'_n)} \right) \tag{4.11}$$

where $Node'_n$ is the goal node in the greedy planner, and $Node'_{k+1} \sim Node'_n$ are generated with the greedy planner under the above assumptions. The idea of greedy future cost estimation is shown in Figure 4.3(b). From Figure 4.3, we can conclude that if $T_{(Node'_k,Node'_{k+1})}$ generated from greedy planner is longer than $T_{A,\min}$ in (4.10), then $f_{fg}$ is guaranteed to be higher than $f_{fs}$. However, whether this condition holds depends on the manipulator speed and platform mobility of the robot. If there are still many hidden target poses not revealed yet, $f_{fg}$ is more likely to be higher than $f_{fs}$ because the robot is more likely to manipulate objects multiple times to reveal all the target poses. If it has been long from the initial node to the current node, the future cost is dominant by the $T_{(Node_1,Node_k)}$ term and these two estimation will not differ a lot. Furthermore, the time to calculate $f_{fs}$ is much shorter than the time for $f_{fg}$. Therefore, it is hard to tell which estimation is universally better. In this thesis, we choose the higher one as the final future cost estimation:

$$f_f = \max\left( f_{fs}, f_{fg} \right) \tag{4.12}$$



(a) Single-action future cost estimation (b) Greedy future cost estimation

Figure 4.3 Comparisons of $f_{fs}$ and $f_{fg}$

### 4.2.7 Action Sampling in Search Planning

The workspace is divided into voxel grid with 1 cm resolution. For a normal size workspace, there are about 1000~3000 grid position to place an object; furthermore, if the object is non-cylindrical, the total poses will be over 200 thousands in worst case. Though in the implementation, the possible placement poses are much fewer (around 100~200) due to the object's size, object accessibility constraints and the robot arm workspace, the number is still too great to make the problem solvable for a computer on a mobile robot. With such high branching factor, constructing the graph is a great burden for the computer, not to mention the planning. However, among these possible placement poses, many of them are similar to other object poses and make little difference in later planning. For example, in Figure 4.4(a), the right cuboid is manipulated to reveal the occluded space behind it, and three of the possible new poses of the cuboid are shown in blue in Figure 4.4(b). However, both three placement poses share same current information gain and same object accessibility, which makes them similar in later planning. This makes the planner check many redundant paths in the graph and waste a huge amount of time. Therefore, it is important to reduce the branching factor to make the problem solvable in some extent. There are three types of action available for every node: “Platform Move”, “Grasp and Place” an object and “Push-aside” an object. We reduce the number of last two actions.

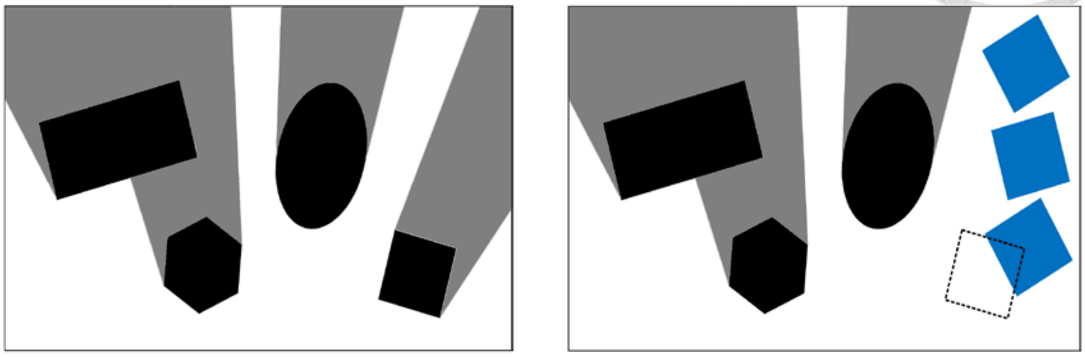

(a) The original object pose (b) Redundant placement poses

Figure 4.4 Redundant actions

In “Grasp and Place” an object, the workspace is observed when the manipulated object is being held in gripper. Therefore, the information gain due to the action is irrelevant to the object placement pose. The placement poses actually affect object accessibility in child nodes and the information gain of later actions, so we classify all child nodes caused by “Grasp and Place” the object by their object accessibility state $ACS_S$. Among each class, we choose one node as the representative. The representative node is the node that the manipulated object occludes minimum number of hidden target poses observed from all robot poses. If there is a draw, choose one with largest $x_o$.

In “Push-aside” action, the information gain is dependent on the pose of the objects being pushed. Therefore, we classify the “Push-aside” action by both the information gain and object accessibility state. As the “Grasp and Place” action, the child node with minimum number of hidden target poses observed from all robot poses is chosen as child states.

## 4.3 Grasp Planning

After the target object is revealed in an observation, as described in section 4.4.1, we can estimate the target's pose with the method described in section 4.4.2. The robot successfully locates the target object's pose and the end of object search is reached. Nevertheless, the search is not considered as the end of a general robot task. In fact, the robot should further hold the target object in hand, so that it can deliver the object to its master. In many cases, the target object is seen by the robot, but still inaccessible. Thus, we need a planner for grasping the target object.

### 4.3.1 Object Accessibility in Grasp Planning

The object accessibility in grasp planning, different from the one in search planning, does not consider the reveal state anymore because the target is already discovered. We express the object accessibility state in grasp planning as:

$$ACS_G\left(POSE_o\right)=\left\{a_{ij}\,\middle|\,i=1,\ldots,N_r,\ j=1,\ldots,N_o\right\} \tag{4.13}$$

where $N_r$ is still the number of robot pose candidates and $N_o$ is the number of visible objects. Note that the target is one of the visible objects now, and hence $N_o$ should be increased by 1.

### 4.3.2 Graph in Grasp Planning

The nodes of the graph in grasp planning are similar to those in search planning. The node of the graph in grasp planning is defined as:

$$Node=\left\{\left(x_{r_i},\mathrm{y}_{r_i},\theta_{r_i}\right),POSE_o\,\middle|\left(x_{r_i},\mathrm{y}_{r_i},\theta_{r_i}\right)\in POSE_r\right\} \tag{4.14}$$

where the nodes are still connected by the edge of the three types of actions. However, the definition of edge cost function is different from that in search planning. In grasp planning, the robot plans for an action sequence that minimizes the time required to grasp the object. Therefore, the edge cost function is simply the time required to perform the action:

$$f_e\left(Edge\left(Node_k, A\right)\right) = T_A \tag{4.15}$$

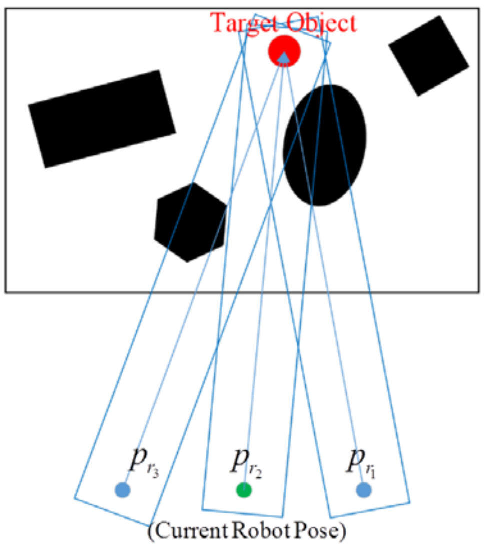
### 4.3.3 A* Object Grasp Planning

In planning, the past cost is the sum of edge costs from the start node, which is the total time spent from the start node. The definition of the past cost function is the same as (4.9) with the edge cost defined in (4.15).

The robot grasps the target object from robot pose candidates. At each robot pose candidate, we select the approach motion of the robot arm to reach the target with minimum number $N_{b_i}$ of objects that block access to the target from the robot pose $p_{r_i}$ $\left(p_{r_i} \in POSE_r\right)$. The approach motion of the arm from each robot pose to the target is fixed in the entire planning. Suppose the robot is at $p_{r_k}$ in arbitrary node $Node_k$. We can estimate the future cost of $Node_k$ as:

$$f_f = \min\left(\left\{\left(N_{b_i} + 1\right) \cdot T_{Mani,\min} + T_{Mov,\left(p_{r_i}, p_{r_k}\right)} \middle| i = 1, \ldots, N_r\right\}\right) \tag{4.16}$$

where $T_{Mani,\min}$ is the minimum time required to manipulate an object, including "Grasp and Place" and "Push-aside" actions, and $T_{Mov,\left(p_{r_i}, p_{r_k}\right)}$ is the time to move from current robot pose $p_{r_k}$ to $p_{r_i}$. Given the current node $Node_k$, $\left(\left(N_{b_i} + 1\right) \cdot T_{Mani,\min} + T_{Mov,\left(p_{r_i}, p_{r_k}\right)}\right)$ means the minimum time required to grasp the target from the robot pose $p_{r_i}$, including the time for the robot to move to $p_{r_i}$ from $p_{r_k}$, to manipulate $N_{b_i}$ blocking objects and to grasp the target. Since the time required for each manipulation is underestimated, the future cost estimation is admissible. Figure 4.5 shows the future cost estimation in one exemplar node. The approach motion of the gripper in Figure 4.5 is a linear motion from the robot to target and is marked with blue arrow and surrounded by a rectangle representing the collision-free area of the gripper to grasp the target.



Minimum time to grasp the target from $p_{r_1}$:

$$T_{p_{r_1},\min} = 2 \cdot T_{Mani,\min} + T_{Mov,\left(p_{r_2},p_{r_1}\right)}$$

Minimum time to grasp the target from $p_{r_2}$:

$$T_{p_{r_2},\min} = 2 \cdot T_{Mani,\min}$$

Minimum time to grasp the target from $p_{r_3}$:

$$T_{p_{r_3},\min} = 3 \cdot T_{Mani,\min} + T_{Mov,\left(p_{r_2},p_{r_3}\right)}$$

Future cost:

$$f_f = \min\left(T_{p_{r_1},\min}, T_{p_{r_2},\min}, T_{p_{r_3},\min}\right)$$

Figure 4.5 Future cost estimation in grasp planning

The start node of grasp planning inherits from the node of search planning when discovering the target. The robot pose is the same, but the object poses require a modification to include the target object pose into $POSE_o$, as mentioned in section 4.3.2. Moreover, if the robot discovers the target with “Grasp and Place” action, the robot enters the grasp planning stage with an object in its gripper. In this case, since the robot already grasped the object, the only feasible set of actions for the start node is “Grasp and Place” the object in gripper. The robot uses the same set of actions as in search planning: “Platform Move”, “Grasp and Place” and “Push-aside” to clear a path to grasp the target. The goal node in grasp planning is that the robot grasp the target.

### 4.3.4 Action Sampling in Grasp Planning

In grasp planning, we also sample object poses to reduce the branching factor of the graph. We classify the manipulation actions of each object in the way the same as in the search planning with the object accessibility state $ACS_G$. The only difference relies on the criterion to choose representative actions from each class. The robot should move obstacles outside arm approaching trajectory to the target. However, the robot does not know which trajectory will be used before the end of the planning and the space outside all trajectories may not be enough to place all the objects. Therefore, sometimes the robot has no choice but to place the object to block one or some of the trajectories.

We sample the placement position by choosing the object pose that blocks minimum number of trajectories. If there are multiple combinations of blocked trajectories with the same number, the robot sample one object pose of each blocking combination. If there is multiple choices of object poses in each blocking combination, select the one whose average distance to those unblocked trajectories is the highest. The distance between an object pose and a trajectory is defined as the minimum distance between the voxels of the object and the trajectory. By doing this, the robot reduces the number of blocked trajectories caused by the manipulation to minimum value, and also considers different approach direction. In Figure 4.6, the robot manipulate the elliptic cylinder to new poses to access the target. The poses shown in yellow blocks one approach trajectories, and the pose shown in blue does not block any trajectory. Therefore, the robot will manipulate the object to the blue pose.

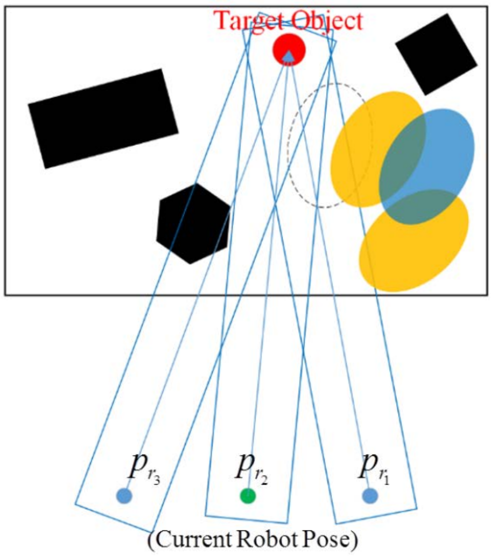


Figure 4.6 Action sampling in grasp planning

## 4.4 Sensor Feedback in Execution

### 4.4.1 Object Registration

We register object point clouds when they are obtained from different camera's viewpoints with the feature matching approach. The feature points are extracted with 5D Harris Corner detector and encoded with ColorSHOT feature vector. Suppose we have a source object point cloud, $CLOUD_s$, from observation, and we would like to find its feature point matching to a target object cloud, $CLOUD_t$, whose object category is known and stored in the database. We map each point in $CLOUD_s$ to a point in $CLOUD_t$ that is nearest to the point in $CLOUD_s$ in ColorSHOT feature space, and vice versa. Only point pair that are mutually mapped to each other is taken as a successful matched feature point. Suppose there are $N$ matched feature points, and we can compute their dissimilarity in feature space, namely, $D_F\left(CLOUD_s, CLOUD_t\right)$, between $\left(CLOUD_s, CLOUD_t\right)$ as:

$$D_F\left(CLOUD_s, CLOUD_t\right) = \left(\frac{\sum d_F\left(p_s, p_t\right)}{N}\right) \tag{4.17}$$

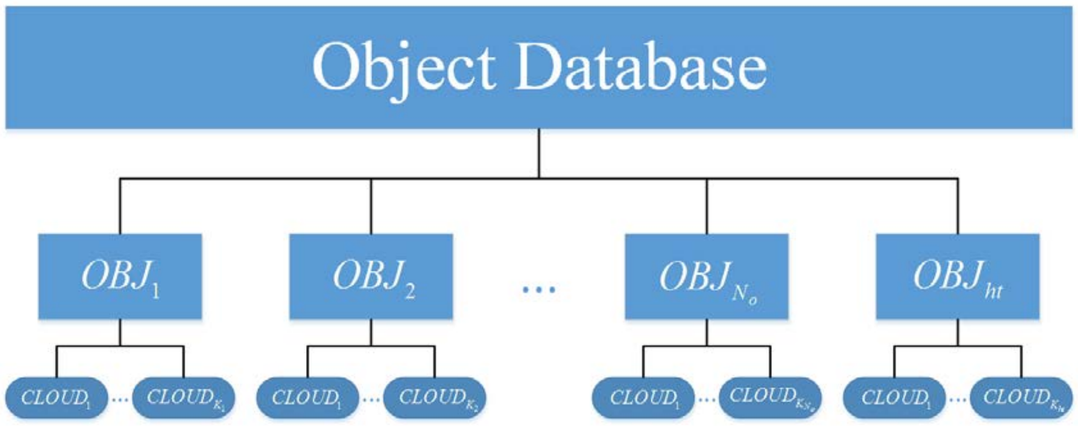
where $\left(\frac{\sum d_F\left(p_s, p_t\right)}{N}\right)$ means the average distance between matched feature point pairs in ColorSHOT feature space.

Registering objects equals to finding correspondence or mapping of objects in current scene to those in database. The database is constructed at the time when the robot first observes the workspace, and it contains all the object clouds of each object categories including the target object, as shown in Figure 4.7. Later observations add more object point clouds in each object category from different view angles based on the registration result. The object point cloud database is divided into $N_O$ visible objects and the hidden target object $OBJ_{ht}$:

$$Database = \{OBJ_1, OBJ_2, ..., OBJ_{N_O}, OBJ_{ht}\} \quad (4.18)$$

Further, each object category ( $OBJ$ ) is composed of $K_i$ object point clouds:

$$OBJ_i = \{CLOUD_1, CLOUD_2, ..., CLOUD_{K_i}\} \quad (4.19)$$


Object Database
$OBJ_1$
$OBJ_2$
...
$OBJ_{N_o}$
$OBJ_{ht}$
$CLOUD_1$ ... $CLOUD_{K_1}$
$CLOUD_1$ ... $CLOUD_{K_2}$
$CLOUD_1$ ... $CLOUD_{K_{N_o}}$
$CLOUD_1$ ... $CLOUD_{K_{ht}}$
Figure 4.7 Object database

Because the object position during each registration is known, position cue is also included to help registration. The dissimilarity between an observation, $CLOUD_s$, and an object, $OBJ_i$, is defined as:

$$D(CLOUD_s, OBJ_i) = \lambda \cdot d_E(CLOUD_s, OBJ_i) + \frac{1}{K_i}\sum_{t=1}^{K_i} D_F(CLOUD_s, CLOUD_t) \quad (4.20)$$

where $d_E(CLOUD_s, CLOUD_t)$ represents the Euclidean distance between the centroids of two object clouds, and $\lambda$ is a weighting factor to balance these two measures. We map new observations to the known object categories in the database. The observation-category mapping $M(CLOUD \to OBJ)$ is a one-to-one mapping and also a hypothesis of the categories of each observed object. To find the best hypothesis, we can define a cost function of each mapping as:

$$f(M)=\sum_{s=1}^{S} D\left(CLOUD_s, OBJ_i = M\left(CLOUD_s\right)\right) \tag{4.21}$$

where $S$ is the object number in this observation. The mapping with minimum cost is the best correspondence and the final registration. If the target object in the database is mapped by an observation after the registration, we conclude that the hidden target has been revealed.

### 4.4.2 Object Pose Estimation

Object pose estimation is achieved by finding linear transformation of matched feature point. Suppose we have a new object point cloud observation $CLOUD_s$ whose pose is unknown, and an object point cloud $CLOUD_t$ of same object category in the database with known pose. The matched feature points in $CLOUD_s$ and $CLOUD_t$ are $\left\{p_{s_1}, p_{s_2}, ..., p_{s_N}\right\}$ and $\left\{p_{t_1}, p_{t_2}, ..., p_{t_N}\right\}$, respectively. The feature points can be expressed as a matrix whose column is the points' coordinate, and we get two matrices $P_s \in R^{3\times N}$ and $P_t \in R^{3\times N}$. The goal is to find the linear transformation ($R,T$) from $P_t$ to $P_s$:

$$\begin{bmatrix} P_s \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_t \\ 1 \end{bmatrix} \tag{4.22}$$

The estimation is done by singular value decomposition (SVD) to obtain the least-square solution of the linear transformation.

### 4.4.3 Grasp Action Sensor Feedback

When grasping the object, there exists some random error in the object's position relative to the gripper and cause misalignment of the object and gripper. The error is up to a centimeter scale and may cause a fail grasp. We solve this problem by mounting an eye-in-hand RGB-D camera on the wrist of the robot arm to adjust the robot arm's final motion. Since the object's approximate position is known, we first filter out the background with depth constraint. For example, the right bottle in Figure 4.8 is too deep to be considered as target, and thus filtered. The object boundaries of an object can be detected with great variation in depth, which is the depth gradient feature. The gripper's approach direction is then fixed to align with the mean position of the boundary, as shown in Figure 4.8. The feedback also ensures the robot arm to extend the right length to prevent knocking down the object or grasping the object too shallow.

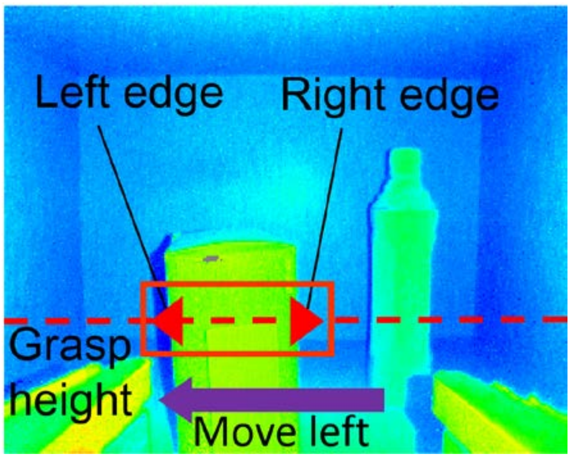


Figure 4.8 View of wrist depth camera to feedback the grasp motion

### 4.4.4 Move Action Sensor Feedback

In executing the plan, the robot moves to different positions and needs to localize itself before manipulating the object. One way is to localize the robot in each frame during the process. However, this approach takes a lot of resource in computation. Since the robot only moves in short distance (<50cm) between each action in the scenario, the wheel odometry is still reliable as an initial guess of the robot's position. Therefore, we only localize the robot before arm manipulation to make sure the manipulation is executed in the robot pose as planned. The robot pose is refined by matching views from last and current pose with Iterative Closest Point (ICP) approach [17]. With the help of the initial guess of the robot pose from the wheel odometry, the alignment is guaranteed to converge to the local minima closest to the initial robot pose.

# Chapter 5

# Experiment Result

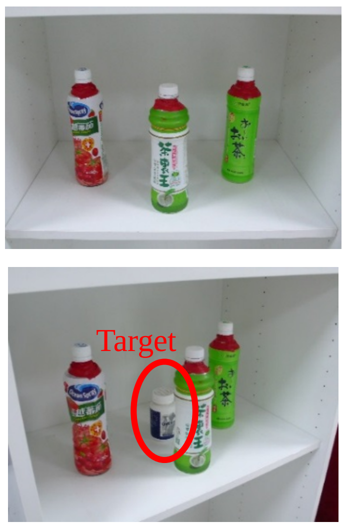
## 5.1 Experiment Setting

In the experiment, the robot is placed in front of a shelf. The dimension of the shelf block is 55 centimeter in width, 35 centimeter in depth and 34 centimeter in height. The bottom plane of the shelf is 93 centimeters above the ground. We set the number of robot pose candidates ( $N_r$ ) as 3, and fixed distance of the robot to the workspace ( $d_{wr}$ ) as 635 mm. The time required for each action is determined by its action type. In the planner, the time for “Platform Move” action is 45 seconds, the time for “Grasp and Place” action is 80 seconds, and the time for “Push-aside” action is 60 seconds.

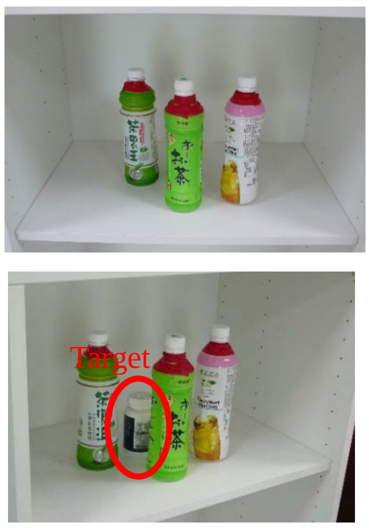
## 5.2 Evaluation on Object Search Planning

We evaluate the performance of the object search planner by testing it on numbers of scenes, as shown in Figure 5.1 and Table 5-1. In each scene, we randomly place the object to test different conditions but do not change the object composition inside the workspace. The objects are placed at least 3 cm apart to allow the robot to manipulate them. Initially, the robot can see all the objects except for the target. Scene (a) and (b) are composed with three PET bottles with different density. Scene (c) and (d) replace one of the PET bottle in (a) and (b) with an object with irregular shape. Scene (e) and (f) both contain four objects, and there are two objects with irregular shape in (f). Scene (g) involves a large object which requires the robot to move the two PET bottles before manipulating it. Scene (h) tests the robot’s ability to planning search and grasp in clutter.

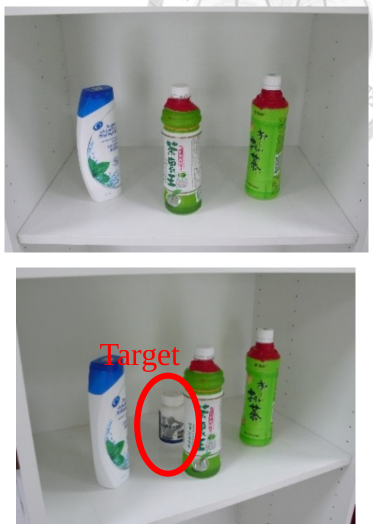
Scene (i) is a general case containing small and large objects with various shapes.



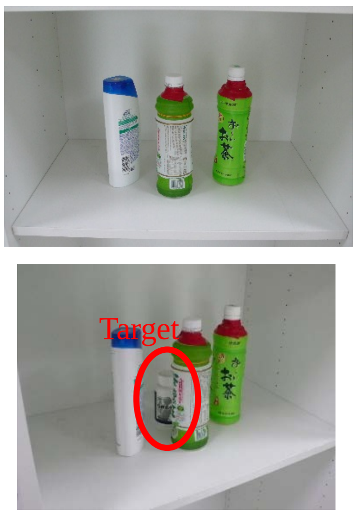
(a)



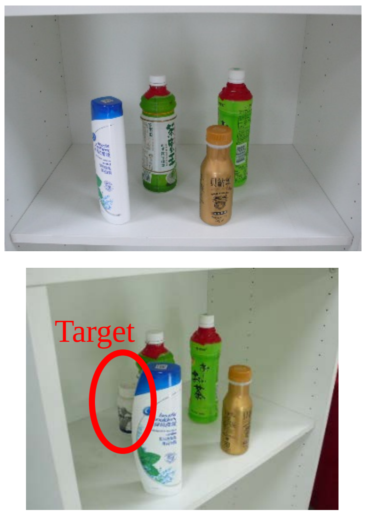
(b)



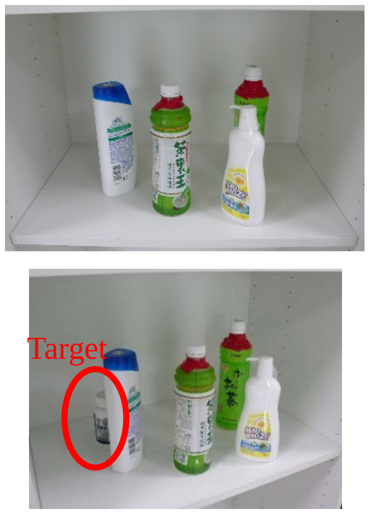
(c)



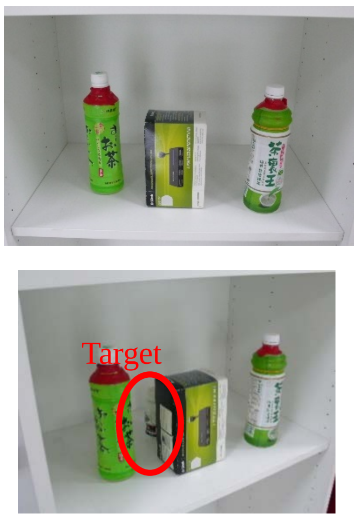
(d)



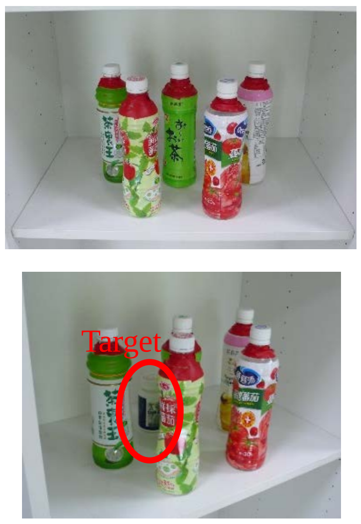
(e)



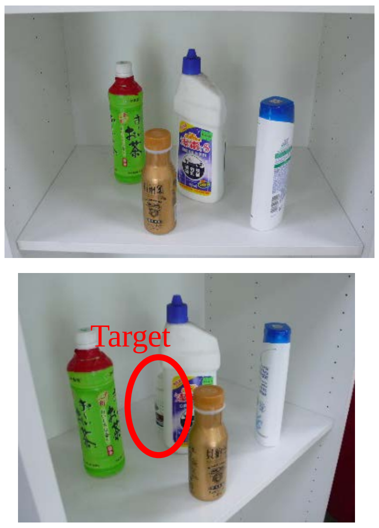
(f)



(g)



(h)



(i)

Figure 5.1 Front view and side view of the test scenes

Table 5-1 Details of the Test Scenes

| Scene | Object Composition | Description |
|---|---|---|
| (a) | 3 PET bottles | The objects are placed loosely. (6cm~7cm apart) |
| (b) | 3 PET bottles | The objects are placed tightly. (3cm~4cm apart) |
| (c) | 1 irregular object<br>2 PET bottles | The objects are placed loosely. (6cm~7cm apart) |
| (d) | 1 irregular object<br>2 PET bottles | The objects are placed tightly. (3cm~4cm apart) |
| (e) | 1 irregular object<br>2 PET bottles<br>1 small bottle | 4 objects which include 1 irregular object |
| (f) | 2 irregular objects<br>2 PET bottles | 4 objects which include 2 irregular objects |
| (g) | 1 large box<br>2 PET bottles | The box is too wide to be grasped in the front, so it needs to be pushed-aside, but is blocked by the bottles. |
| (h) | 5 PET bottles | The objects in the rear row is blocked and also stops the robot from placing object behind them. |
| (i) | 1 large irregular object<br>1 irregular object<br>1 PET bottle<br>1 small bottle | Objects with different shape and size are placed together. |

Since this thesis hold different assumptions and plans a problem with much higher complexity than the state-of-the-art object search planner [7-9], we do not evaluate the proposed object search planner with these approach. Instead, we implemented two other planners to prove the advantage of the proposed planner. The first planner for comparison is the “greedy planner.” The greedy planner ignores long-term optimality and only maximizes the number of revealed target at each step and takes the minimum time. In other words, the greedy planner always maximizes the utility function at each step:

$$U = \frac{\left(\Delta n_{HT}\right)_A}{T_A} \tag{5.1}$$

where $\left(\Delta n_{HT}\right)_A$ is the number of possible target poses which is revealed at the step by taking action $A$, and $T_A$ is the required time. When the robot grasps an object, the greedy planner always places the object to the pose with maximum $x$. The other planner is the A* planner without action sampling. The A* planner will keep all possible child nodes of each node and find an optimal path in the state graph to discover all the target with minimum expected time.

We evaluate the planners by measuring their success rate to generate a plan, the planning time, the expected time to find the target, and the branching factor of the state graph. A planning is failed if the planning time is more than 10 minutes or the resulting action sequence is composed with more than 20 actions. Higher success rate means that the planner is more robust and reliable.

If the planning successes, we also measure its planning time, the branching factor of the state graph and the expected time to find the target. Among these data, the planning time is an important indicator to tell if the planner is practical, and is highly correlated with the branching factor. The expected time to find the target indicates how well the planner achieves its original goal.

The success rate of each planner is shown in Table 5-2. In simple case like (a), all planners are capable of generating a feasible plan. However, when objects are placed tighter, such as (b) and (d), the greedy planner sometimes fail to generate a plan because the objects are blocking one another and may require the robot to move an object without revealing any hidden target poses. In such situation, the greedy planner does not know which action is better and is stuck in the local minima by taking random actions without information gain. On the other hand, the A* planner fails when the optimal solution takes many steps to complete. Therefore, A* planner fails in most complex scenes because the time required to check every similar path in the state graph is too long, which shows the value of action sampling.

Table 5-2 Success Rate of the Planners

| Scene | Greedy Planner | A* Planner | Proposed Planner |
|---|---|---|---|
| (a) | **10 / 10** | **10 / 10** | **10 / 10** |
| (b) | 8 / 10 | **10 / 10** | **10 / 10** |
| (c) | **10 / 10** | 7 / 10 | **10 / 10** |
| (d) | 6 / 10 | 0 / 10 | **10 / 10** |
| (e) | 4 / 10 | 0 / 10 | **10 / 10** |
| (f) | 7 / 10 | 0 / 10 | **9 / 10** |
| (g) | 3 / 10 | 3 / 10 | **8 / 10** |
| (h) | 0 / 10 | 0 / 10 | **8 / 10** |
| (i) | 0 / 10 | 2 / 10 | **8 / 10** |

The proposed planner, also known as A* planner with action sampling, is able to find a plan in most scenes. The proposed planner survives in the scenes that fails the other two planners. However, there are still some situations which the planner cannot handle. Since the objects are randomly placed, sometimes none of the available manipulation action makes certain object accessible. Further, since we follow a greedy criterion when sampling child nodes with same object accessibility state, if an object is blocked by multiple objects, it is possible that none of the action sequences remove all blocking objects. In both case, if the hidden target poses blocked by this object cannot be seen by moving the robot platform, the hidden target pose will never be revealed.

Figure 5.2(a) shows an example of failed scenes. The front row of the PET bottles are graspable by the robot but the two PET bottles in the rear row block the path of the arm to place the bottles to the deep workspace. Moreover, each of the two rear PET bottles are blocked by two bottles, so no single action available to make them accessible. Therefore, after grasping the front bottles and observing the workspace, the robot tends to place them back to their original position to block minimum number of possible hidden target poses, which cause the bottles in rear row never accessible. Figure 5.2(b) shows another example of failed scenes. Though the robot can push large objects aside to move objects which is not graspable, a push-aside action requires very wide free space for the arm to perform it. In Figure 5.2(b), the large box is placed in the deep workspace and all sampled placement poses of PET bottles may block the arm to push the large box.

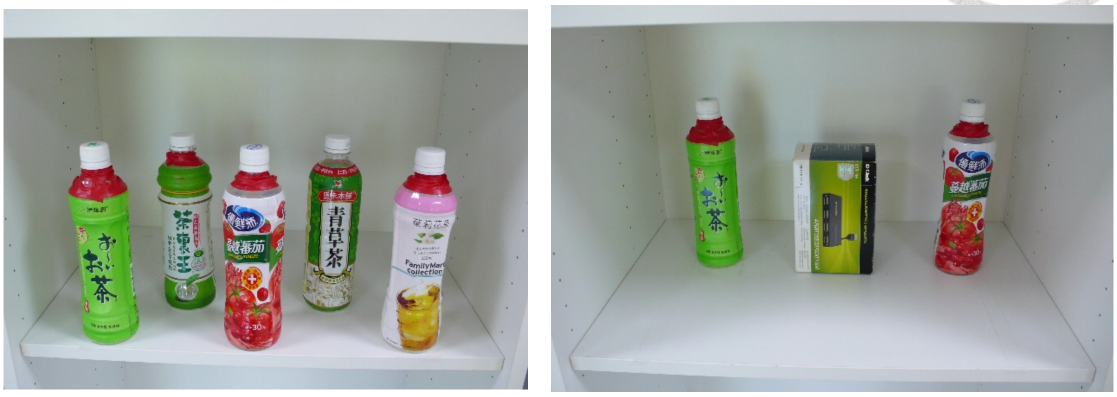

(a) (b)

Figure 5.2 Examples of failed scenes

The planning time and expected time to reveal the target is shown in Figure 5.3. From scene (a), (b), (c) and (d), we can conclude that the expected time and planning time are longer if the objects are place tighter. This result is reasonable since the tight formation of the objects hinders the robot from revealing most hidden target poses by platform movement and forces the robot to manipulate objects. As a result, the planning time is highly correlated with the difficulty in manipulating the objects that occludes hidden target poses. These objects are usually large or placed in deep of the workspace. Therefore, the robot needs to plan many steps of action before manipulating the object. This effect causes the dense and complex scene such as (f) and (h) requires more planning time. However, in these scenes, most of the hidden target poses are revealed before manipulating the final object to reveal the last few target poses occluded by this object. Thus, the total expected time is not extremely high.

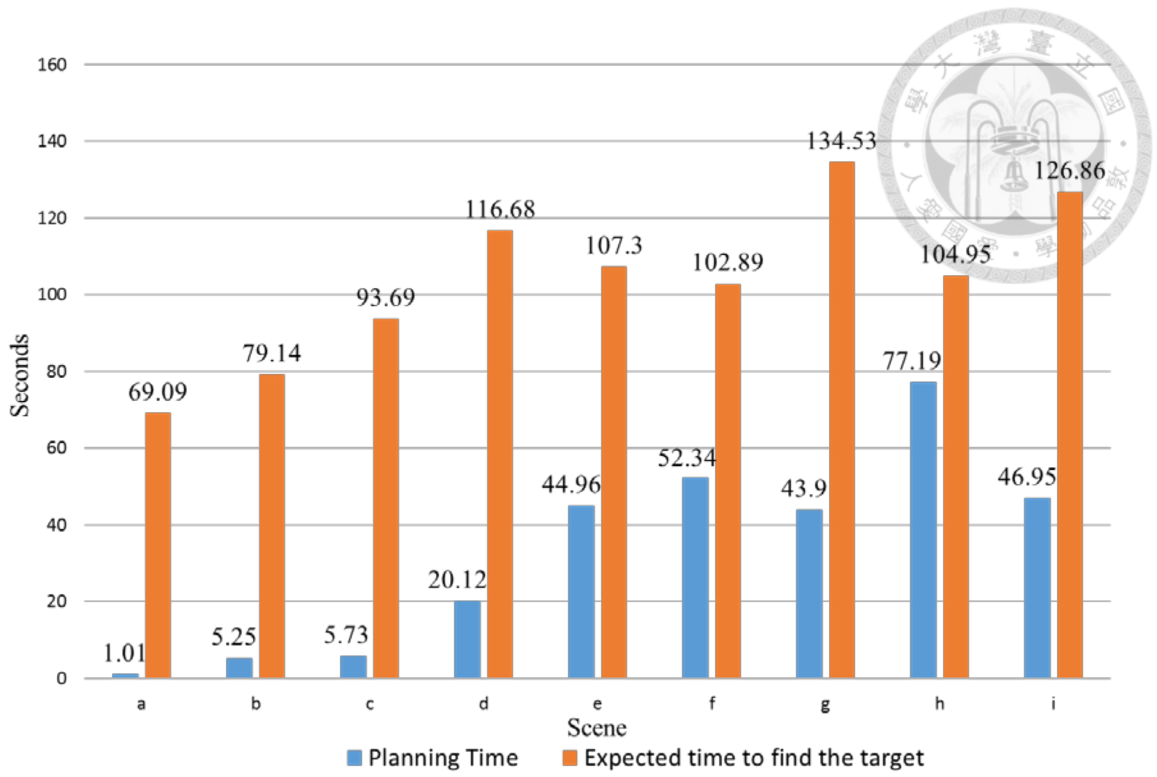
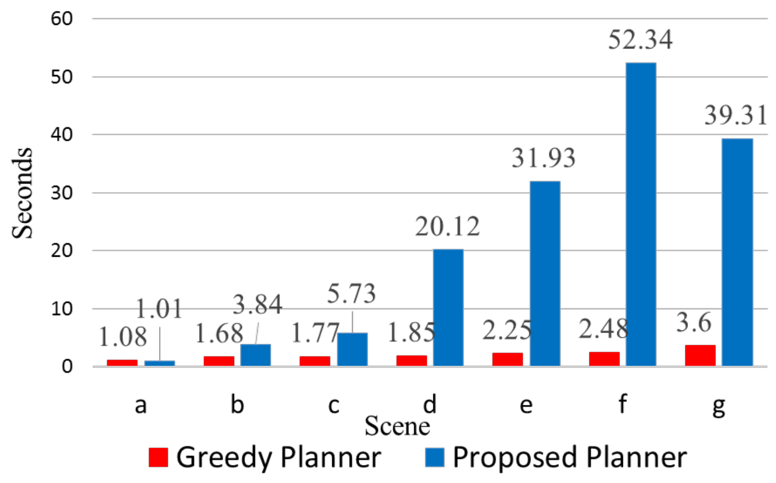


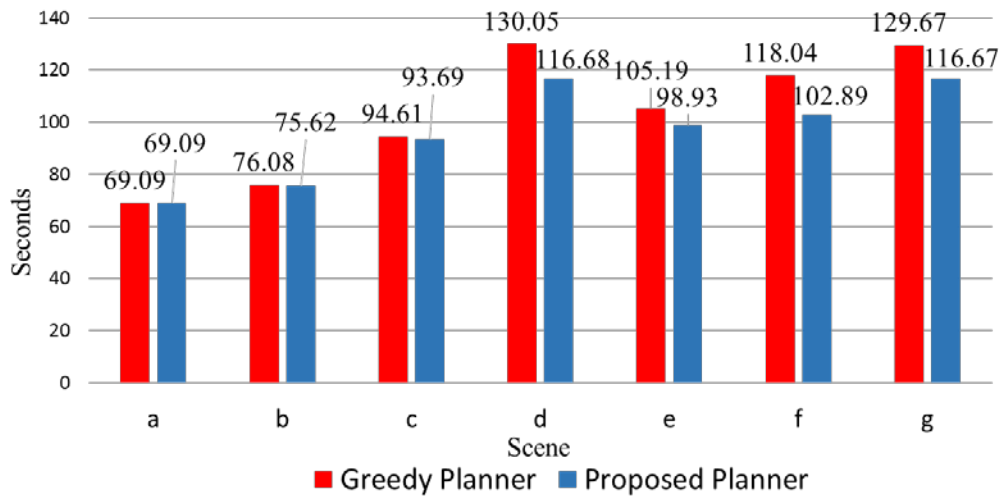
Figure 5.3 Planning time and expected time to find the target of the proposed planner

In scenes which greedy planner and the proposed planner both succeed in generating a plan, the planning time and the expected time to find the target is shown in Figure 5.4. Since the greedy planner only explore one path in the state graph, the planning time is much less than the proposed planner. Nevertheless, the expected time in greedy planner is longer than the expected time of proposed planner. The difference is not significant because the greedy planner can always find actions with high utility in the beginning of planning to reveal many possible hidden target poses, but struggles to reveal the target poses that is occluded by objects in the deep of the workspace. Therefore, in the scene where the greedy and the proposed planner both succeed, the difference in expected time is mainly contributed by the objects in the deep workspace. Therefore, the difference in expected time are longer in complex scenes than in simple scenes.

The A* planner is guaranteed to derive the optimal plan to search the target object. However, it takes a lot of planning time to reveal all the hidden target poses. Figure 5.5 shows the planning time, average branching factor and expected time of the A* planner and the proposed planner on the scene where they both succeed in generating a plan. Although the expected time of A* planner is slightly shorter than the expected time of the proposed algorithm, the A* planner takes much longer planning time. The high branching factor of the A* planner is the main reason for its long planning time. Furthermore, the massively growing states also drains out the runtime memory and in turn drags down the computing speed. Therefore, the action sampling not only saves planning time but also reduces the computing resource required for planning.

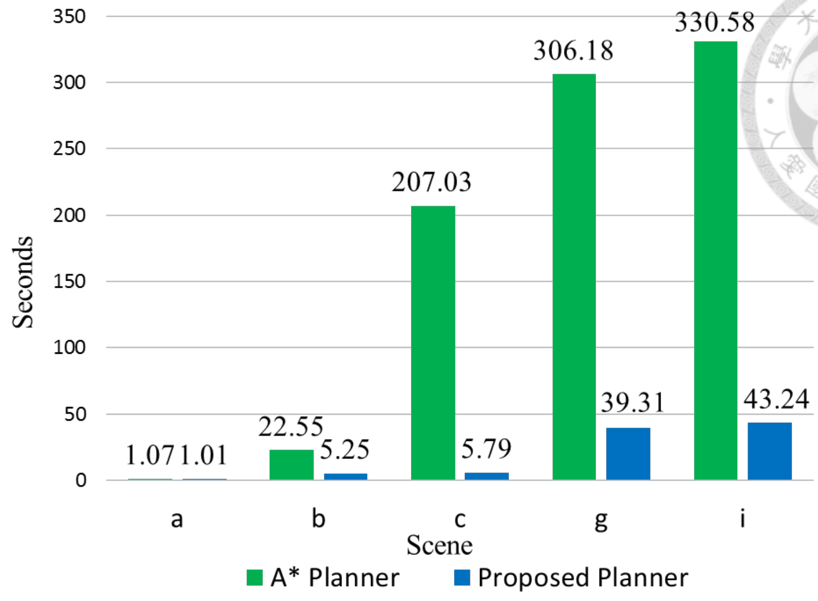


(a) Planning time

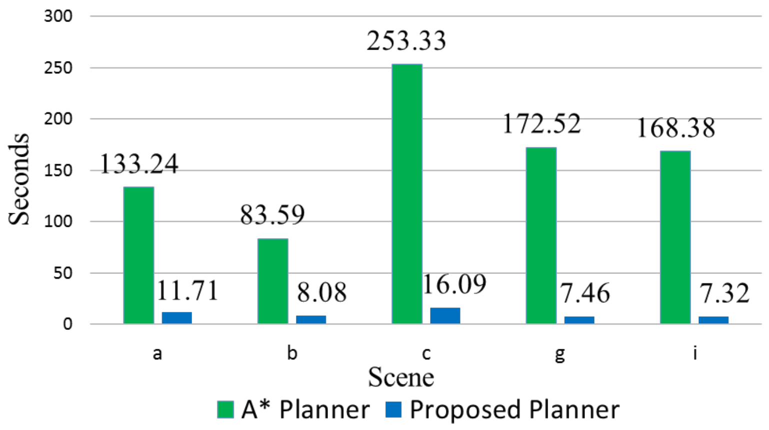


(b) Expected time to find the target

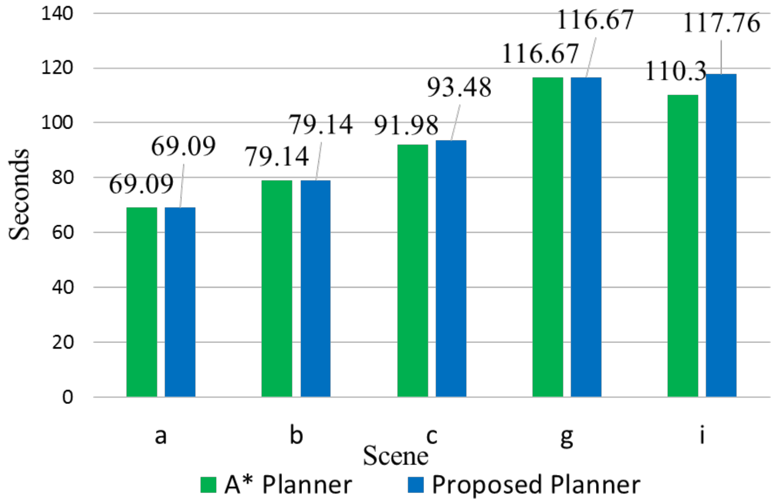
Figure 5.4 Comparison between greedy planner and proposed planner



(a) Planning time



(b) Average branching factor



(c) Expected time to find the target

Figure 5.5 Comparison between the A* planner and the proposed planner

## 5.3 Evaluation on Object Grasp Planning

In this section, we evaluate the performance of the proposed object grasp planner. Since the grasp planner aims to plan an action sequence after discovering the target object, we evaluate the planner by adjusting the scene shown in Figure 5.1 to make the target object visible at the first place and plan grasping. The test scenes is as shown in Figure 5.6, and we do not change the object poses in this evaluation.

(a)

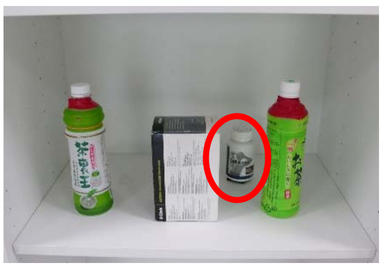
(b)

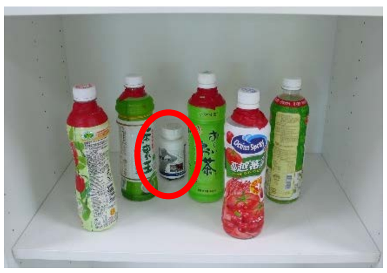
(c)

(d)

(e)

(f)


(g)


(h)

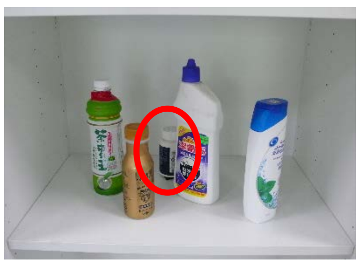

(i)

Figure 5.6 Test scenes in evaluation on object grasp planning

We evaluate the proposed object grasp planner with the A* planner without action sampling to test its planning time, time required to grasp the object, and its branching factor. As in object search planning, a planning is failed if the planning time is more than 10 minutes or the resulting action sequence is composed with more than 20 actions.

The proposed planner successfully plans action in all test scenes, while the A* planner failed to generate a plan in 10 minutes in scene (f), (h) and (i) because many action steps are required to grasp the target. Figure 5.7 shows the performance of the proposed planner under all scenes.

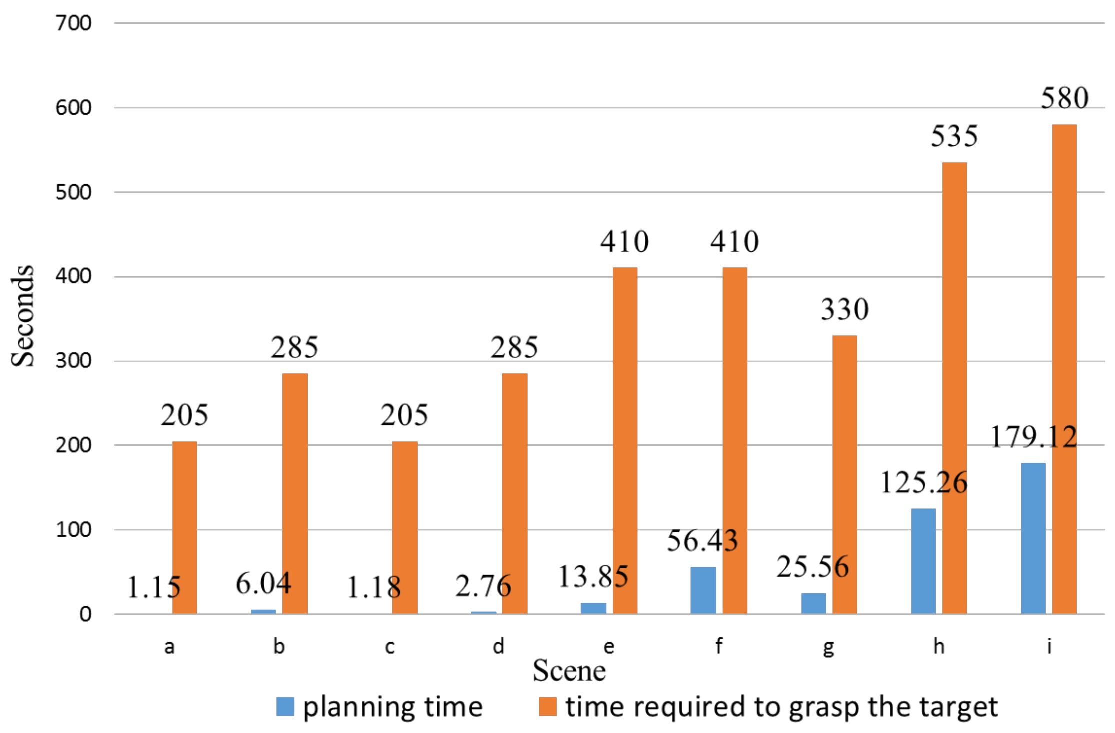


Figure 5.7 Planning time and time required to grasp the target of the proposed planner

Since the time required to grasp the target is highly correlated with the number of obstacles, and the number of obstacles determines the number of actions taken, the trend in planning time is consistent with the trend in time required to grasp the target. In (a), (b), (c) and (d), the scenes that the objects are placed denser, (b) and (d), require the robot to remove more obstacles to grasp the target than in looser scenes, (a) and (c). In (e) and (f), there are more objects, including the objects with irregular shape, which is placed densely and lengthens the planning time. The planner is still able to grasp the target in complex scenes, such as (h) and (i), as long as there exists free space for the robot to grasp objects and place it.

Figure 5.8 and Figure 5.9 shows the planning time and time required to grasp the target of each scene which is successfully planned by both planners. The proposed planner uses much less planning time to get a nearly-optimal solution compared to the optimal solution of A* planner. In summary, the proposed planner performs better than A* planner in simple cases, such as (a), (b), (c) and (d), for shorter planning time, and is more robust and much faster in planning under the complex scenes, such as (e), (f), (g), (h) and (i).

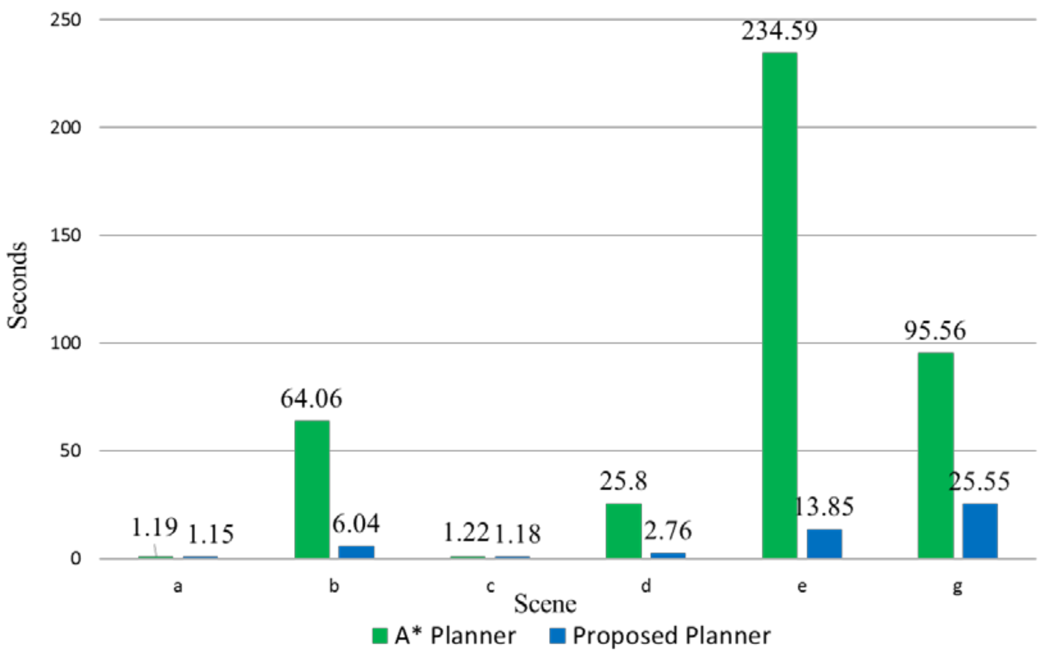


Figure 5.8 Planning time

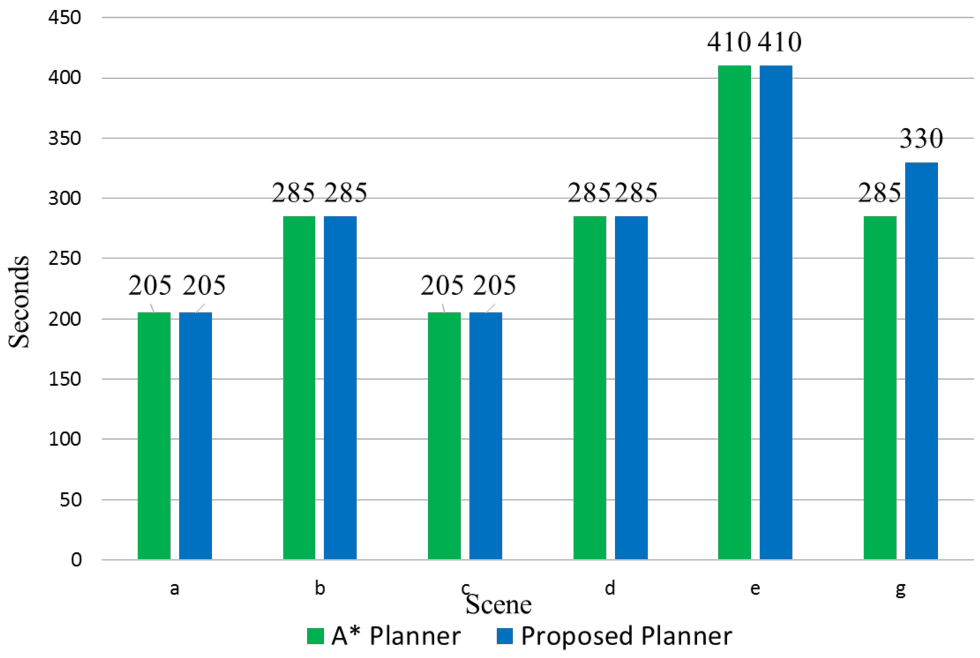
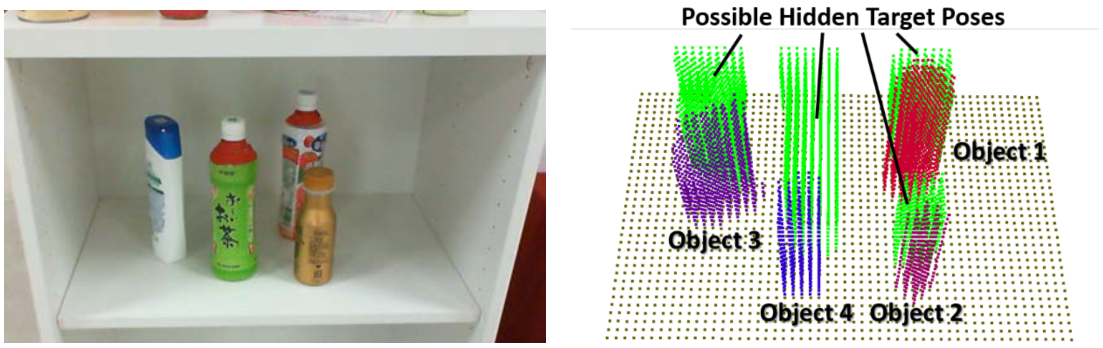


Figure 5.9 Time required to grasp the target

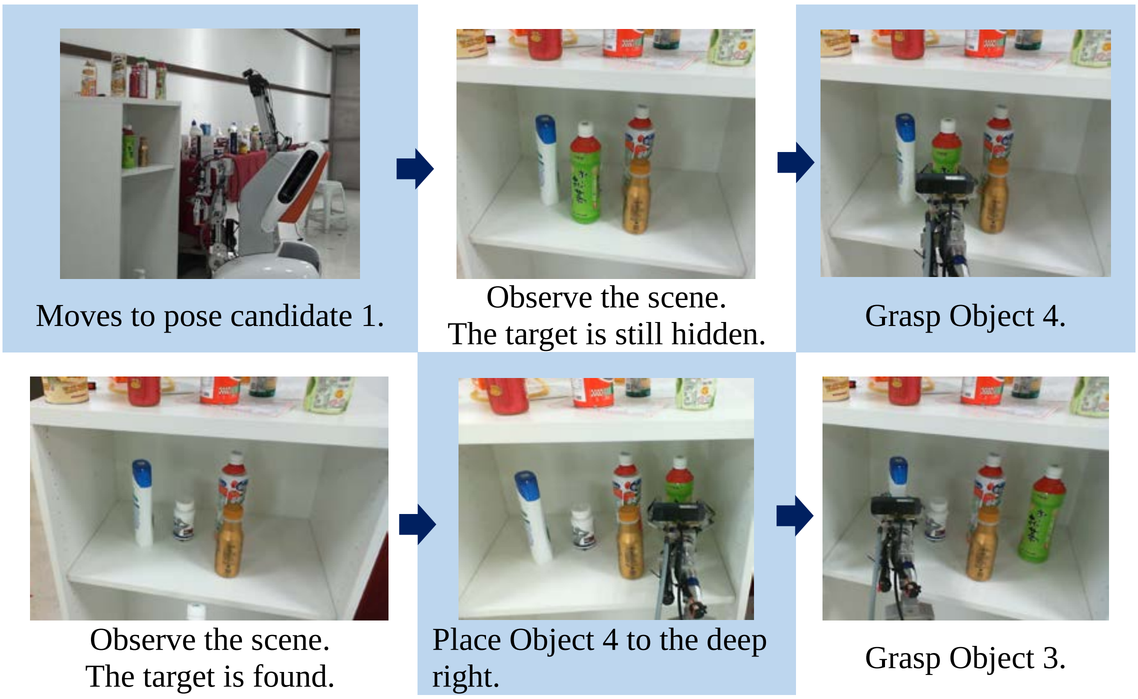
## 5.4 Overall Test

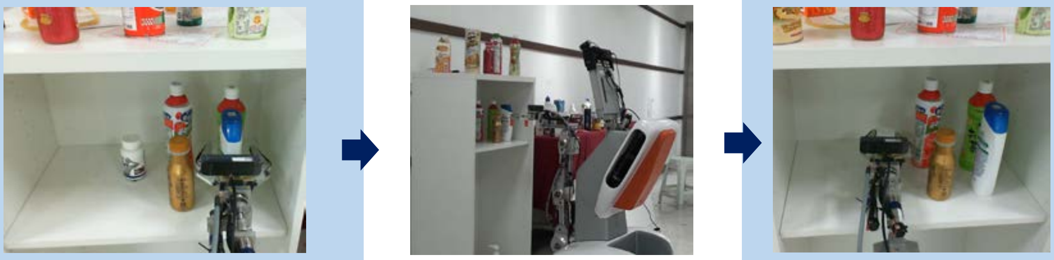
The overall action of object searching and grasping is performed to show the practical use of the proposed planner and system. Figure 5.10 and Figure 5.11 show one of the case involving 3 cylindrical objects and an object with irregular shape. The robot searches the target by arranging the position of objects and plans grasping after the target is discovered. Figure 5.12 and Figure 5.13 show a scene which includes a large object. As proposed, the robot is able to push large object aside and discover the target.



(a) The test scene (b) Workspace voxel grid of the scene

Figure 5.10 Overall test scene 1


Moves to pose candidate 1.
Observe the scene.
The target is still hidden.
Grasp Object 4.
Observe the scene.
The target is found.
Place Object 4 to the deep right.
Grasp Object 3.

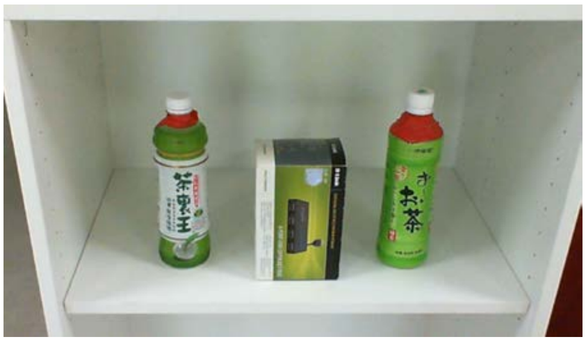


Place Object 3 to the shallow right.

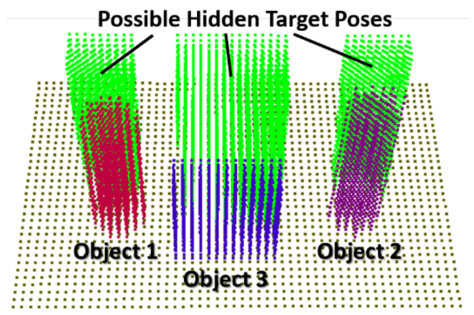
Move to pose candidate 3.

Grasp the target.

Figure 5.11 Action sequence in overall test scene 1

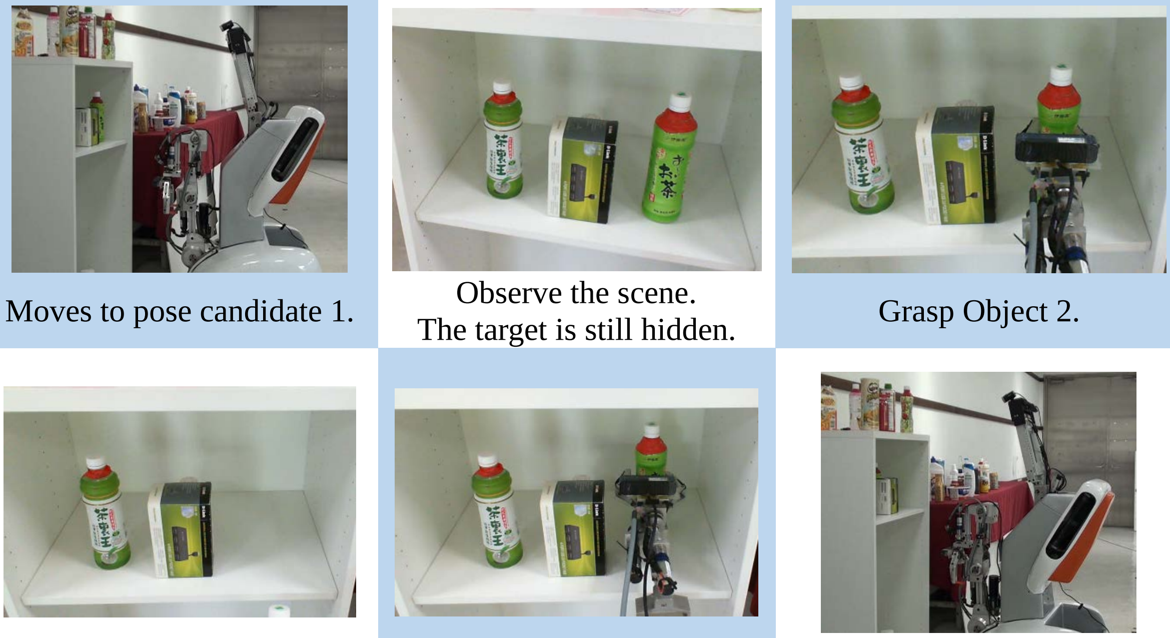


(a) The test scene



(b) Workspace voxel grid of the scene

Figure 5.12 Overall test scene 2



Moves to pose candidate 1.

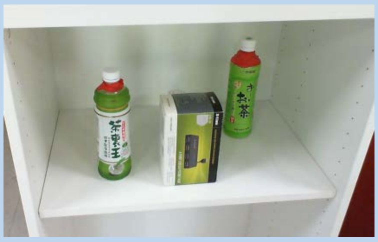
Observe the scene.
The target is still hidden.

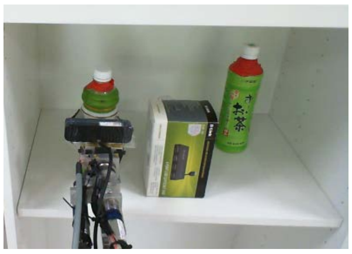
Grasp Object 2.

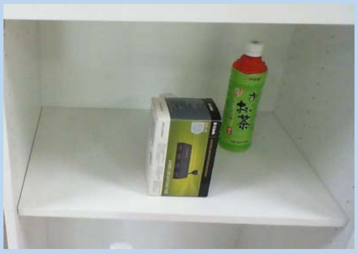
Observe the scene.
The target is still hidden.

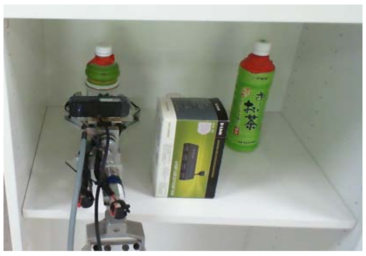
Place Object 2 to the deep right.

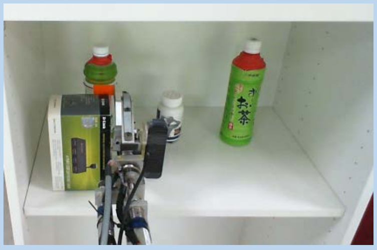
Move to pose candidate 3.

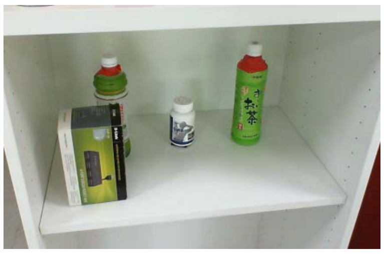

Observe the scene.
The target is still hidden.


Grasp Object 1.


Observe the scene.
The target is still hidden.


Place Object 1 to the deep left.


Push Object 3 to left side.


Observe the scene.
The target is found.

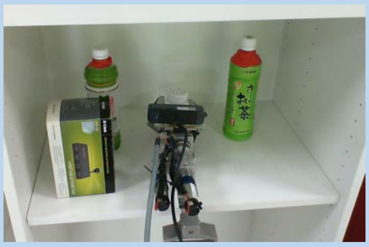

Direct grasp the target.

Figure 5.13 Action sequence in overall test scene 2

# Chapter 6

# Conclusion

In this thesis, we proposed the world's first object search system based on both robot manipulation and active visual search. By modeling the workspace and objects, the robot plans an action sequence to retrieve the target object. We samples the possible child nodes in the planner to simultaneously reduce the planning time and keep valuable choices for next action. In the experiment, we compare our approach with a greedy planner and an A* planner. The results show that the proposed approach is able to generate a plan in reasonable planning time and is more robust than two other planners.

## 6.1 Future Works

The complexity in planning in limited workspace is extremely high and even a feasible plan may not exist due to lack of space inside the workspace. In reality, the space is usually limited, but there may be some space outside the shelf block for the robot to put objects. Therefore, in future works, we would like to expand the limited workspace to spaces outside the block. The robot may divide the space into separated slots to place the objects. In this manner, we can incorporate the planner which allows the robot to permanently remove objects from the workspace to reduce the planning complexity and make the planner more practical and more flexible.

In our assumption, the objects are placed separately to let the robot manipulate them. However, in real world, objects are often stacked together and difficult to be grasped with the parallel plate gripper. Although [18] proposed a physical-based scheme to analyze the effect of collision between the gripper and multiple obstacles to complete grasping in clutter, object searching in clutter requires much more accuracy in object motion. Therefore, we believe that the solution for object searching in clutter lies in minimizing the collision in manipulation. A new gripper design for robot to manipulate object in clutter is thus also a possible direction for future works.

# REFERENCE

[1] Y. Ye and J. K. Tsotsos, "Sensor Planning for 3D Object Search," *CVIU,* vol. 73, pp. 145-168, 1999.

[2] K. Sjo, D. Lopez, C. Paul, P. Jensfelt, and D. Kragic, "Object Search and Localization for an Indoor Mobile Robot," *Journal of Computing and Information Technology,* pp. 67–80, 2009.

[3] A. Aydemir, M. Gobelbecker, A. Pronobis, K. Sjoo, and P. Jensfelt, "Plan-based Object Search and Exploration Using Semantic Spatial Knowledge in the Real World," in *Proceedings of the 5th European Conference on Mobile Robots (ECMR'11)*, ed, 2011.

[4] A. Aydemir, K. Sjoo, J. Folkesson, A. Pronobis, and P. Jensfelt, "Search in the real world: Active visual object search based on spatial relations," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2818-2824.

[5] J. Ma, T. H. Chung, and J. Burdick, "A probabilistic framework for object search with 6-DOF pose estimation," *Int. J. Rob. Res.*, vol. 30, pp. 1209-1228, 2011.

[6] L. L. S. Wong, L. P. Kaelbling, and T. Lozano-Perez, "Manipulation-based active search for occluded objects," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 2814-2819.

[7] M. Dogar, M. Koval, A. Tallavajhula, and S. Srinivasa, "Object search by manipulation," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 4973-4980.

[8] M. Dogar, M. Koval, A. Tallavajhula, and S. Srinivasa, "Object search by manipulation," *Autonomous Robots,* vol. 36, pp. 153-167, 2014/01/01 2014.

[9] M. Gupta, T. Ruhr, M. Beetz, and G. S. Sukhatme, "Interactive environment exploration in clutter," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 5265-5272.

[10] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 1-4.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, pp. 100-107, 1968.

[12] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM,* vol. 24, pp. 381-395, 1981.

[13] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, ed, 1988, pp. 147-151.

[14] I. Sipiran and B. Bustos, "Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes," *The Visual Computer,* vol. 27, pp. 963-976, 2011.

[15] F. Tombari, S. Salti, and L. Di Stefano, "A combined texture-shape descriptor for enhanced 3D feature matching," in *2011 18th IEEE International Conference on Image Processing (ICIP)*, 2011, pp. 809-812.

[16] S. Salti, F. Tombari, and L. Di Stefano, "SHOT: Unique signatures of histograms for surface and texture description," *Computer Vision and Image Understanding,*

vol. 125, pp. 251-264, 2014.

[17] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 14, pp. 239-256, 1992.

[18] M. Dogar, K. Hsiao, M. Ciocarlie, and S. Srinivasa, "Physics-Based Grasp Planning Through Clutter," in *Proceedings of Robotics: Science and Systems*, ed. Sydney, Australia, 2012.