

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

三維透明物體辨識系統於服務型機器人之應用  
3D Transparent Object Recognition for Service Robotics

賴柏任

Po-Jen Lai

指導教授：羅仁權 博士

Advisor: Ren C. Luo, Ph.D.

中華民國 104 年 7 月

July 2015

## 誌謝



就讀碩士班的這兩年是我人生美好的一段回憶。大三時，懵懵懂懂的我加入現在的實驗室做專題，接觸到世界頂級的工具-ROS，開啟我對機器人的研究。如今時光飛逝，四年的時間一轉眼就過了。感謝父母親的栽培教誨、不論原因地支持鼓勵，讓我能夠專注於我喜愛的領域。非常感謝我敬愛的指導老師羅仁權教授，提供我們豐富的資源以及指教，教導我們大至國際觀，小至待人處事的道理，無論哪個環節都是一生難得的體驗。兩年的師生互動中，老師充分展現出國際級學者的視野與風範，不僅帶領我們探索學術知識的廣度與深度，更是給予我們充分的機會去體驗本科系以外的經驗，希望我們成為均衡發展的人才。最重要的是，老師以身作則地積極推動台灣產業轉型，他付出的努力令我非常感動，也期許自己，畢業只是一個過程，我對於推動台灣機器人產業的熱情與努力將不會停息。是老師教導我，國家的未來，是由我們親手打造出來的。另外，要特別感謝獻章學長盡心盡力的教學，在 ROS 團隊的時期我學到許多一生受用的智慧。

在國立臺灣大學智慧機器人及自動化國際研究中心 (NTU- iCeiRA) 兩年的專題生活及兩年研究生活中，我要感謝鏡文、俊吉、瑋隆、獻章、陞祐、繼棠、金成、詩頌、善成、博宇、玲盈、昕映、東榕、旭佳等博班學長姊，還有博瀚、蕭明、昀軒、璿文、耕成、凱杰、彭熙、盛俊、宗緯、政勳、世哲、哲毅、哲瑋、懷遠、少騁、謝浩、冠軒等碩班學長姊，以及同屆共同奮鬥的夥伴宜庭、王蕊、立偉、智賢、相匠、照文、岳軒、昭霖、佳文、辰嘉，最後還有積極認真的學弟妹煒森、建偉、金博、邦甫、冠志、文謙、榮育、柏宏、士紘、耕銘、建安、銘駿及俊豪，以及助理逸偉、安甫、之琳 (Katherine)、婉儀 (Beryl)、中莉 (Allie)、青芳 (Fine)、雯雅 (Tracy)、琬怡 (Wanyi)。謝謝大家支持與幫助，不分晝夜地討論研究，同甘共苦地參加比賽，不厭其煩地幫忙雜務，真的很開心這段時間有幸能跟各位一起經歷奮鬥，相信每個生活點滴都是一生難忘回憶。

能完成這篇論文，我要感謝生命中的每一個人，真的非常謝謝你們。

賴柏任 謹誌

一百零四年七月

## 中文摘要



隨著科技的進步，讓生活變得更自動化的需求是擋不住的浪潮。屆時，將有許多服務型機器人會深入到人類的環境中進行各式各樣的任務，例如在家中幫忙倒牛奶、在餐廳中幫忙端水等等。在我們的生活中，用到了許多透明的物體，包括玻璃杯、寶特瓶、甚至是玻璃門，若機器人沒有能力辨認透明物體，將會造成許多問題，這些問題包含機器人容易毀損玻璃杯、容易撞到玻璃門或窗戶等等，不僅僅會造成機器人工作上的不便利、損壞的玻璃更可能造成人類的危險。因此，在此篇論文中，我們提出了一個透明物體的姿態辨識系統，其中我們將討論的重心放在透明物體的辨識上，輔以討論姿態辨識的模組以及抓取的模組。之所以將重心放在透明物體的辨識上，是因為姿態辨識以及抓取的功能在非透明物體上已經有相當成熟的研究。然而，辨識透明物體的研究是近十幾年來才漸漸發展起來，而且論文數量相當稀少，我們若能發展出有效的透明物體辨識演算法，將場景中透明物體所在的位置標示出來，接下來的姿態辨識和抓取的方法就可以參考適用於非透明物體的技术了。

故關於辨識透明物體，我們討論了三種方法，第一種使用 RGBD 感測器來感測場景、利用感測器的特性加以辨認出透明物體的所在位置。第二種及第三種方法都使用一般的相機當作感測器，分別使用 Latent Dirichlet Analysis 以及 Deep Learning 的機器學習方法來學習辨識透明物體。雖然探討了三種方法，我們主要使用第一種方法辨識到的透明物體輪廓當作姿態辨認模組的輸入。

於是，我們可以使用已經儲存在資料庫裡的透明物體 3D 模型，配合前述方法所找到的透明物體輪廓，可以利用配準的方式進行姿態的估計，進而得到物體姿態的估計值。

關鍵字：服務型機器人、透明物體辨識、姿態辨識、機器人作業系統

# ABSTRACT



With the advancement of technology, the trend to make our lives more convenient by robot technology is unstoppable. In the future, many service robots will enter our living environments to do all kind of tasks from pouring milk for us in our home to serve water in restaurants. In our living environment, there are lots of transparent objects including cups made of glass, PET bottles and glass doors. If a robot who serve in our environment cannot recognize transparent objects, it might easily broke the transparent objects made by glass, it might not be able to open the door made of glass, it might bump into and broke glass windows and cause danger. As a result, we propose algorithms that make a robot be able to recognize and estimate the pose of transparent objects in this thesis. We emphasize on transparent object recognition because pose estimation and manipulation for non-transparent objects are relatively mature, while research on transparent object recognition just starts from a decade ago with a few papers discussing this problem. If we can develop effective algorithm for recognizing transparent object, we can take advantage of pose estimation and grasping for non-transparent object to build a complete system for grasping transparent objects.

For recognizing transparent object, we discuss three methods in this thesis. The first method which uses RGBD sensor to detect the transparent object is mainly used because the result is suitable for pose estimation.

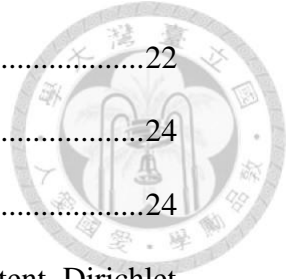
With the stored 3D model of transparent object and the silhouettes of transparent object, we can estimate the pose by matching the model and the silhouette. Experiments show that our method can be used to detect and estimate the pose of transparent objects.

**Keywords:** Service Robotics, Transparent Object Recognition, Pose Estimation, Robot Operating System

# CONTENTS



|  |           |
|--|-----------|
| 口試委員會審定書 .....   | #         |
| 誌謝 .....   | i         |
| 中文摘要 .....   | ii        |
| ABSTRACT .....   | iii       |
| CONTENTS .....   | iv        |
| LIST OF FIGURES .....                                  | vii       |
| LIST OF TABLES .....                                   | x         |
| <b>Chapter 1 Introduction.....</b>                     | <b>1</b>  |
| 1.1 Motivation.....                                    | 1         |
| 1.2 Problem statement .....                            | 3         |
| 1.3 Literature Review .....                            | 4         |
| 1.4 Contributions .....                                | 5         |
| 1.5 Thesis Organization .....                          | 6         |
| <b>Chapter 2 System Architecture .....</b>             | <b>7</b>  |
| 2.1 Robot Operating System (ROS) .....                 | 7         |
| 2.2 Hardware Introduction.....                         | 12        |
| 2.2.1 Personal Robot 2 (PR2).....                      | 12        |
| 2.2.2 Kinect .....                                     | 15        |
| <b>Chapter 3 Detection of Transparent Objects.....</b> | <b>17</b> |
| 3.1 Grabcut-Based Method .....                         | 17        |
| 3.1.1 Overall process of Grabcut-based method .....    | 17        |
| 3.1.2 Grabcut.....                                     | 19        |



|                  |   |           |
|------------------|---|-----------|
| 3.1.3            | Transparent Object Classification .....   | 22        |
| 3.2              | LDA-Based Method.....   | 24        |
| 3.2.1            | Gaussian Mixture Model (GMM) .....  | 24        |
| 3.2.2            | Probabilistic Latent Semantic Analysis (pLSA) & Latent Dirichlet Allocation (LDA) ..... | 26        |
| 3.2.3            | LDA for transparent object detection .....  | 31        |
| 3.3              | Deep Learning-Based Method.....   | 33        |
| 3.3.1            | Deep Neural Network & Convolutional Neural Network.....                                 | 34        |
| 3.3.2            | Regions with Convolutional Neural Network (R-CNN).....                                  | 38        |
| 3.3.3            | Selective Search .....  | 39        |
| <b>Chapter 4</b> | <b>Pose Estimation of Transparent Objects.....</b>                                      | <b>48</b> |
| 4.1              | Training.....   | 49        |
| 4.1.1            | Model Construction.....   | 49        |
| 4.1.2            | Silhouettes Generation .....  | 53        |
| 4.2              | Testing.....  | 54        |
| 4.2.1            | Test Silhouette Detection .....   | 54        |
| 4.2.2            | Initial Pose Estimation .....   | 54        |
| 4.2.3            | Pose Refinement.....  | 56        |
| <b>Chapter 5</b> | <b>Grasping.....</b>  | <b>57</b> |
| 5.1              | ROS tf for Coordinate Transform .....   | 57        |
| 5.2              | PR2 manipulation pipeline .....   | 61        |
| <b>Chapter 6</b> | <b>Experiment .....</b>   | <b>64</b> |
| 6.1              | Experiment Setup.....   | 64        |
| 6.2              | Detection of Transparent Objects .....  | 64        |
| 6.2.1            | Grabcut-based Method .....  | 64        |

|                  |  |           |
|------------------|--|-----------|
| 6.2.2            | Deep Learning Based Method.....              | 67        |
| 6.3              | Pose Estimation of transparent Objects ..... | 71        |
| <b>Chapter 7</b> | <b>Conclusion and Future Works .....</b>     | <b>73</b> |
| REFERENCE        | .....  | 74        |
| Curriculum Vitae | .....  | 78        |



# LIST OF FIGURES



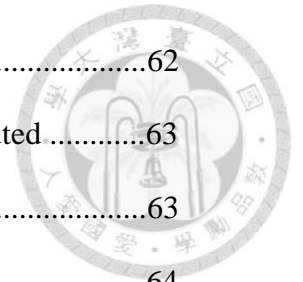
|           |  |    |
|-----------|--|----|
| Fig. 1.1  | Softbank's Pepper Robot .....  | 2  |
| Fig. 1.2  | Some commonly seen transparent objects .....   | 2  |
| Fig. 1.3  | A non-transparent object .....   | 3  |
| Fig. 1.4  | Some non-transparent objects cause NaN in depth image .....                                      | 5  |
| Fig. 2.1  | ROS Logo .....   | 7  |
| Fig. 2.2  | A diagram for basic ROS concept .....  | 8  |
| Fig. 2.3  | The robots that are compatible to ROS .....  | 9  |
| Fig. 2.4  | Willow Garage's PR2 .....  | 13 |
| Fig. 2.5  | PR2's links .....  | 14 |
| Fig. 2.6  | PR2's Control System .....   | 14 |
| Fig. 2.7  | Microsoft's Kinect .....   | 15 |
| Fig. 2.8  | Kinect's technical specification .....   | 16 |
| Fig. 3.1  | Top: The scene containing transparent objects. Bottom: Transparent candidates in the scene ..... | 18 |
| Fig. 3.2  | Three glasses and the detected highlights .....  | 19 |
| Fig. 3.3  | Two examples result of Grabcut algorithms .....  | 20 |
| Fig. 3.4  | An illustration of Graph cut .....   | 21 |
| Fig. 3.5  | The result of computing color similarity .....   | 23 |
| Fig. 3.6  | An example of GMM approximation .....  | 26 |
| Fig. 3.7  | An illustration of Singular Value Decomposition (SVD) .....                                      | 27 |
| Fig. 3.8  | A simple word-document co-occurrence matrix .....  | 27 |
| Fig. 3.9  | Use SVD to decompose the example word-document matrix .....                                      | 28 |
| Fig. 3.10 | A geometric view of pLSA ( $z_1$ to $z_3$ are three latent topics) .....                         | 29 |





|           |   |    |
|-----------|---|----|
| Fig 3.11  | The method used in [3]  | 31 |
| Fig 3.12  | The detection result in [3]   | 32 |
| Fig 3.13  | An illustration of a single hidden layer neural network                                     | 35 |
| Fig 3.14  | The difference between shallow and deep neural network                                      | 36 |
| Fig 3.15  | The architecture of common convolutional neural network                                     | 38 |
| Fig 3.16  | Object detection system overview  | 38 |
| Fig. 3.17 | An illustration of selective search   | 40 |
| Fig. 3.18 | Our input for selective search  | 40 |
| Fig 3.19  | The result of selective search on our example   | 40 |
| Fig. 4.1  | Flowchart of Grabcut-based pose estimation  | 48 |
| Fig. 4.2  | A transparent object and its wrapped-up copy  | 49 |
| Fig. 4.3  | The KinectFusion app in Windows   | 50 |
| Fig. 4.4  | An illustration of the built model  | 50 |
| Fig. 4.5  | The downloaded model is very sparse   | 51 |
| Fig. 4.6  | Applying midpoint adding algorithm to solve for sparsity                                    | 52 |
| Fig. 4.7  | Downsample the model to around 1000 points  | 52 |
| Fig. 4.8  | An illustration of silhouette generation  | 53 |
| Fig 4.9   | An illustration of pose estimation  | 56 |
| Fig 5.1   | PR2's 3D coordinate frames  | 57 |
| Fig 5.2   | An example of a simple robot  | 58 |
| Fig 5.3   | The relationship between laser and base of the simple robot                                 | 59 |
| Fig 5.4   | The tf tree for the simple robot  | 60 |
| Fig 5.5   | Interpolated IK path from pre-grasp to grasp planned for a grasp point of an unknown object | 62 |
| Fig 5.6   | A path to get the arm to the pre-grasp position has been planned using the                  |    |

|   |    |
|---|----|
| motion planner and executed .....   | 62 |
| Fig 5.7 The interpolated IK path from pre-grasp to grasp has been executed .....    | 63 |
| Fig 5.8 The object has been lifted .....  | 63 |
| Fig. 6.1 PR2 robot manipulates transparent object .....                             | 64 |
| Fig. 6.2 Five transparent objects used to test the performance of recognition ..... | 65 |
| Fig. 6.3 The original image and correct recognition result .....                    | 66 |
| Fig. 6.4 Some of the images in test dataset .....                                   | 67 |
| Fig. 6.5 A recognition result from R-CNN .....                                      | 69 |
| Fig. 6.6 Testing result of R-CNN method .....                                       | 69 |
| Fig. 6.7 One of the results of pose estimation .....                                | 72 |



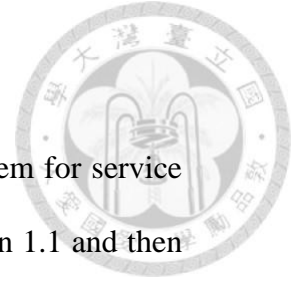
# LIST OF TABLES



Table 6.1 Recall and Precision of Grabcut-based Transparent Object Recognition .....67

Table 6.2 Success Rate of Pose Estimation .....72

# Chapter 1 Introduction



The main topic of this thesis is a transparent object recognition system for service robotics. In this chapter, the motivation of the research is stated in section 1.1 and then section 1.2 introduces the clear problem statement of the research. In section 1.3, the related work is described so that the state-of-the-art algorithms are presented. After showing the related work, the contributions are stated in section 1.4. Finally, in section 1.5, the overall organization of this thesis and the relationship among all chapters are illustrated.

## 1.1 Motivation

Robotic technologies advances fast in the past decade, conventional industrial robot manipulators are utilized in factory in order to replace human's work, from the simplest pick-and-place job to exquisite IC manufacturing. Nowadays, performance (precision, speed, stability) of industrial robots are very good, thus make people develop more sophisticated robots.

Recently, Softbank have launched a new social robot called "Pepper" (as shown in Fig 1.1), which is the first humanoid robot designed to live with humans. He is able to converse with you, recognize and react to your emotions, move and live autonomously. However, Pepper doesn't clean, doesn't cook and doesn't have the abilities to serve as a service robot.

Although the term "service robot" does not have a strict technical definition, The International Federation of Robotics (IFR) has proposed a tentative definition: "A service robot is a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations. For example, a service can do household chores and let people focus on more valuable

works.



Fig. 1.1 Softbank's Pepper Robot

In such context, we are thinking, what is the ability that a service robot must have. One of the functions that a service robot need is the ability to recognize and manipulate transparent objects. The reason is because transparent objects are almost everywhere, from our home and restaurants to laboratories. Some commonly seen transparent objects are shown in Fig 1.2.

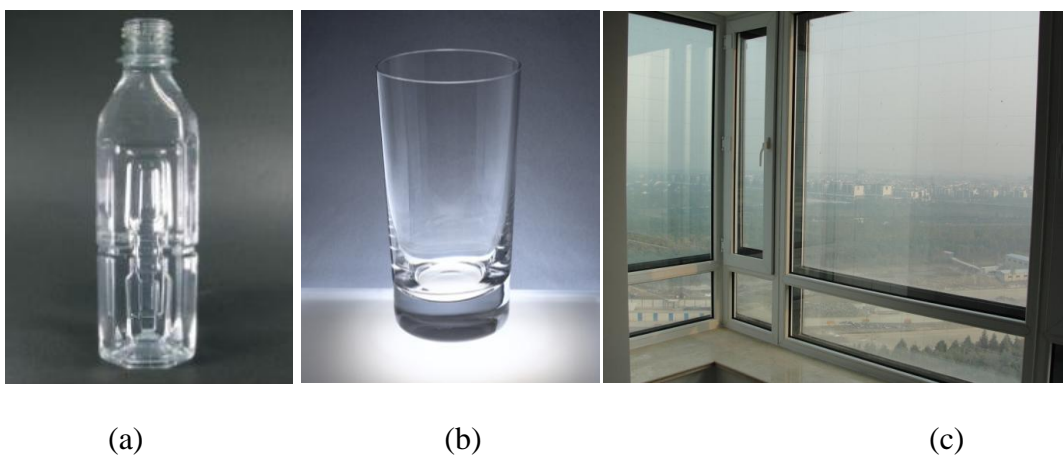
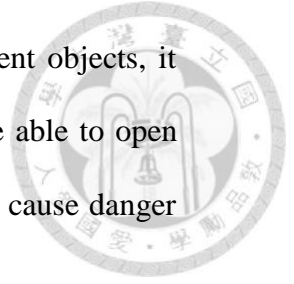


Fig. 1.2 Some commonly seen transparent objects  
(a) PET bottle (b) cup made of glass(c) windows



If a robot who serve in our environment cannot recognize transparent objects, it might easily broke the transparent objects made by glass, it might not be able to open the door made of glass, it might bump into and broke glass windows and cause danger in our environment.

As a result, we would like to explore the algorithms that make a robot be able to recognize, even manipulate transparent objects in this thesis.

## 1.2 Problem statement

Transparent materials are difficult to detect due to the appearances of transparent objects change over different backgrounds, their edges are implicit and contain strong highlights.

Here, we gave a stricter definition of transparent object. A transparent object is the object having the property of transmitting rays of light through its substance so that bodies situated beyond or behind can be distinctly seen. So if we have a bottle made of glass, but the glass has color, the bottle is not considered as transparent. An example can be seen in Fig 1.3.



Fig. 1.3 A non-transparent object.



### 1.3 Literature Review

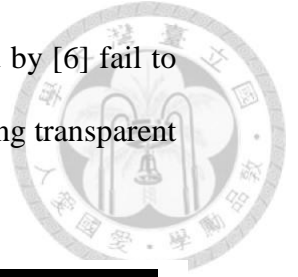
Transparent object recognition is a difficult problem for computer vision community for a long time, the research about this topic is not popular until 2003.

In 2003, Osadchy et al. [1] used specular highlights as a positive source of information to recognize shiny objects. But the process required a bright light source. McHenry et al. [2] proposed several features and characteristic of transparent object such as color similarity, blurring, overlay consistency, texture distortion and highlights. The method is effective to distinguish transparent objects. Although it successfully segment transparent object, the algorithms only adapted with non-cluttered scene without pose estimation. Fritz et al. [3] use an additive model of latent features to learned transparent local patch appearance. It successfully detects transparency in varying backgrounds too. However, all of the methods mentioned above gained no knowledge about object pose, thus unable to make robot grasp.

Phillips et al. [4] provide a new idea to detect semi-transparent objects by utilizing inverse perspective mapping. This method needs to capture more than one view and assumes that object is on a plane. The largest error of pose estimate was about 10.4 mm.

For pose estimation, Lysenkov et al. [5] detect transparent object by using Kinect sensor, while unknown depth information (shown as black area in depth image) is considered as transparent object. It proposed an algorithm to calculate poses of transparent objects. The improvement in [6] makes their algorithm be able to deal with overlapped instances and cluttered transparent objects. Although it proposed a method to handle pose estimation of transparent object, there are some cases that make detection fails. As shown in Fig.1.4, some non-transparent objects that are common in a laboratorial scene also generate unknown depth value so they would be considered as

transparent objects as well. These objects will make the method proposed by [6] fail to detect the transparent ones. And since [6] is the newest work on recognizing transparent objects, we improved the algorithms in [6].



(a)

(b)

Fig. 1.4 Some non-transparent objects cause invalid area in depth image.

(a) Kinect RGB image (b) Kinect Depth image

## 1.4 Contributions

To build a service robot system with functions of recognizing, estimating the pose and grasping transparent objects, there are three main contributions in this thesis. They are listed as follow:

- Improve the state-of-the-art pose estimation algorithm for transparent object so that the robot can correctly distinguish non-transparent objects which cause NaN (Not a Number, i.e. invalid) in depth image from transparent ones
- Investigate some other algorithms for recognizing transparent objects, including Latent Dirichlet Allocation and Regions with Convolutional Neural Network
- System design and the implementation of pose estimation systems for robots



## 1.5 Thesis Organization

In chapter 2, we will discuss the system architecture, including hardware and software. Next, in chapter 3, several techniques for detecting transparent object will be investigated. After detecting the transparent objects in the scene, we would like to estimate the pose of the detected objects so that they can be grasped by our robot. As a result, pose estimation will be explained in chapter 4. With pose of transparent objects estimated, the robot can grasp the transparent object with ease. In chapter 5, we will describe the teach-and-play algorithm for robot grasping. And in chapter 6, we will present some experiments about our system. Finally, we will make conclusions and state the future work in chapter 7.



## Chapter 2 System Architecture



In this chapter, the overall system is described. We use Robot Operating System (ROS) as the tool for developing and running our algorithms. So we will introduce ROS in section 1. As for the hardware platform, we use Personal Robot 2 (PR2), which will be covered in section 2. In the last section, we then explain the software architecture of a transparent object grasping robot.

### 2.1 Robot Operating System (ROS)

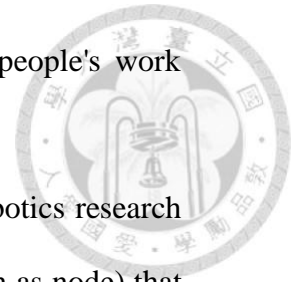
ROS [7] is a set of utilities and libraries for implementing all different kinds of functionality on robots. In short, it is a meta operating system from WillowGarage, which is designed for usage with distributed robot systems. It's called a meta operating system, because it needs another operating system to run. It's mainly developed for Ubuntu (a Linux distribution), but it also supports other Operating Systems like Windows and Mac OS, but the support for them can still be considered as experimental.



Fig. 2.2 ROS Logo.

Development in Robotics is slow in the past decades because people distributed their efforts to different systems. For instance, the research team in University of Tokyo may develop their own robots, but these robots are not compatible to Stanford's robots. For normal people as you and me, it is even not possible to get their source codes and provide any help to the development in robotics. In that context, making huge progress

in Robotics is quite difficult. So ROS came out, aiming at gathering people's work together and then develop more advanced robots on top of them.



As a result, the primary goal of ROS is to support code reuse in robotics research and development. ROS is a distributed framework of processes (as known as node) that enables executables to be individually designed and loosely coupled at runtime. The basic idea is shown in Fig 2.2. Each node is a process, and nodes can communicate to each other by “ROS topic” [8] or “ROS service” [9].

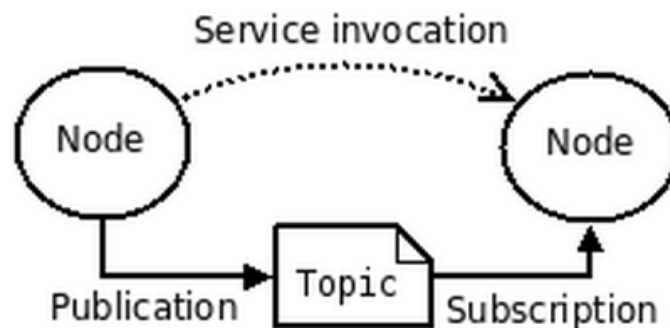


Fig. 2.2 A diagram for basic ROS concept.

ROS has lots of great features, but there are two features must be mentioned:

- Code Reuse

ROS is open source and provides strong package management system. You can easily build your own system by combining packages released on ROS. The example in the next point would make this point clearer.

























- Executables can be individually designed and easily connected at runtime

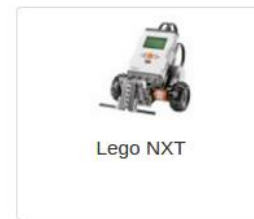
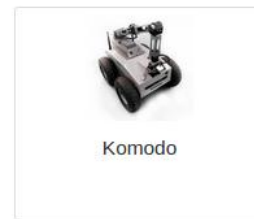
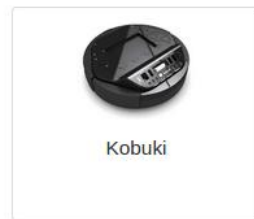
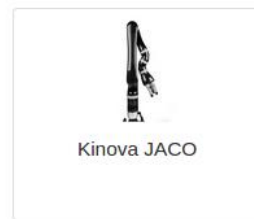
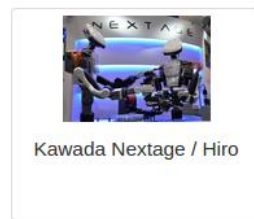
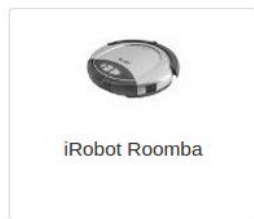
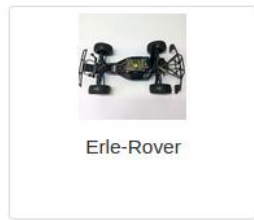
This feature is explained by an example. We can imagine that if Peter is an expert of robot navigation, he wrote a program A that can let robot go anywhere it want and avoid obstacles smoothly. However, Peter need to use image so that the robot know how to plan its moving trajectory (let's say the program that can process camera data be program B). If Peter uses ROS, he can run other people's camera data processing

program as program B, and connected program A and program B in the environment provided by ROS. And vice versa, other people can use the first-class navigation program provided by Peter instead of writing a poor-performance navigation program by their own.



Being such a powerful tool, ROS can be run on many robots, as shown in Fig 2.3.

|  |  |   |  |
|--|--|---|--|
|  <p>ABB Robotics (ROS-Industrial)</p> |  <p>Adept MobileRobots Pioneer family (P3DX, P3AT, ...)</p> |  <p>Adept MobileRobots Pioneer LX</p>              |  <p>Adept MobileRobots Seekur family (Seekur, Seekur Jr.)</p> |
|  <p>Aldebaran Nao</p>                 |  <p>Allegro Hand SimLab</p>                                 |  <p>AMIGO</p>                                      |  <p>AscTec Quadrotor</p>                                      |
|  <p>Barrett Hand</p>                |  <p>BipedRobin</p>  |  <p>Bitcraze Crazyflie</p>                      |  <p>Clearpath Robotics Grizzly</p>                          |
|  <p>Clearpath Robotics Husky</p>    |  <p>Clearpath Robotics Jackal</p>                         |  <p>Clearpath Robotics Kingfisher</p>            |  <p>CoroWare Corobot</p>                                    |
|  <p>Cyton-Gamma</p>                 |  <p>Denso VS060</p>                                       |  <p>Dr. Robot Jaguar</p>                         |  <p>Eddiebot</p>  |
|  <p>Erle-Brain</p>                  |  <p>Erle-Copter</p>                                       |  <p>Erle-Copter Ubuntu Core special edition</p> |  <p>Erle-HexaCopter</p>                                     |



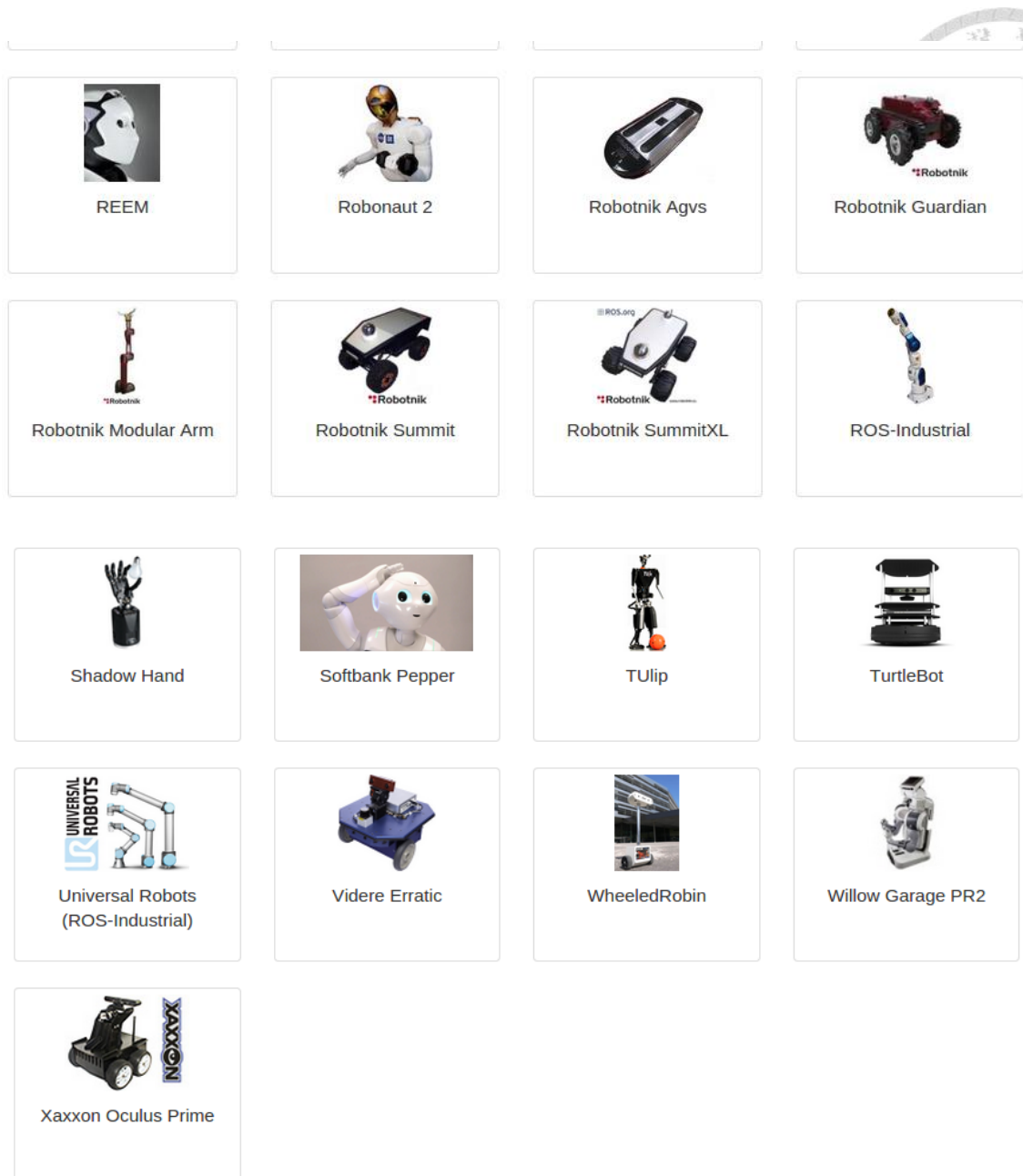


Fig. 2.3 The robots that are compatible to ROS.

Software in the ROS Ecosystem can be separated into three groups:

- language- and platform-independent tools used for building and distributing ROS-based software;
- ROS client library implementations such as roscpp, rospy, and roslisp;
- packages containing application-related code which uses one or more ROS client libraries.

ROS was originally developed in 2007 under the name switchyard by the Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot STAIR project.

From 2008 until 2013, development was performed primarily at Willow Garage, a robotics research institute/incubator. During that time, researchers at more than twenty institutions collaborated with Willow Garage engineers in a federated development model.

In February 2013, ROS stewardship transitioned to the Open Source Robotics Foundation. And since then, OSRF is the primary maintainer of ROS.

## **2.2 Hardware Introduction**

In this thesis, we use Willow Garage's Personal Robot 2 (PR2) as our platform to test our algorithm. As for the sensor, we use the Kinect mounted on PR2's head to get RGB and depth image for recognizing transparent objects.

### **2.2.1 Personal Robot 2 (PR2)**

It is of a size similar to a human. PR2 is designed as a common hardware and software platform for robot researchers. The PR2 has two 7-DOF arms with a payload of 1.8 kilograms (4.0 lb). Sensors include a 5-megapixel camera, a tilting laser range finder, and an inertial measurement unit. The "texture projector" projects a pattern on the environment to create 3D information for capture by the cameras. This approach is also known as structured light. The head-mounted laser scanner measures distance by time-of-flight. The two computers located in the base of the robot are 8-core servers, each of which has 24 Gigabytes of RAM, for a total of 48 GB. The battery system consists of 16 laptop batteries.



Fig. 2.4 Willow Garage's PR2.

In June 2010, Willow Garage made two-year loans of a PR2 to 11 research teams. Each PR2 was to include two arms, a "rich sensor suite", a mobile base, 16 CPU cores, and the company's free, open-source Robot Operating System (ROS) framework, which controls the PR2 and comes with software libraries for perception, navigation, and manipulation. The teams were to have a chance not only to program a general-purpose robot but also to contribute their work on Willow Garage's open-source robotics platform to a wide community of researchers. In August 2010, Willow Garage announced that the PR2 robot was available for purchase. As for now, PR2 is an important platform for researchers in the world to collaborate on pushing the limit of robot technology.



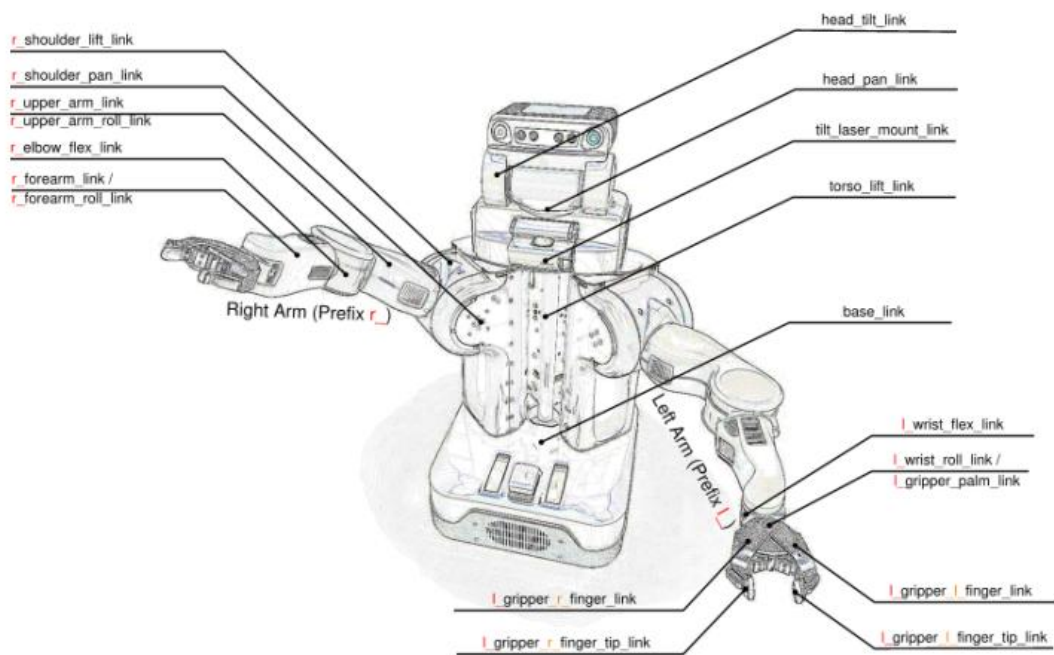


Fig. 2.5 PR2's links.

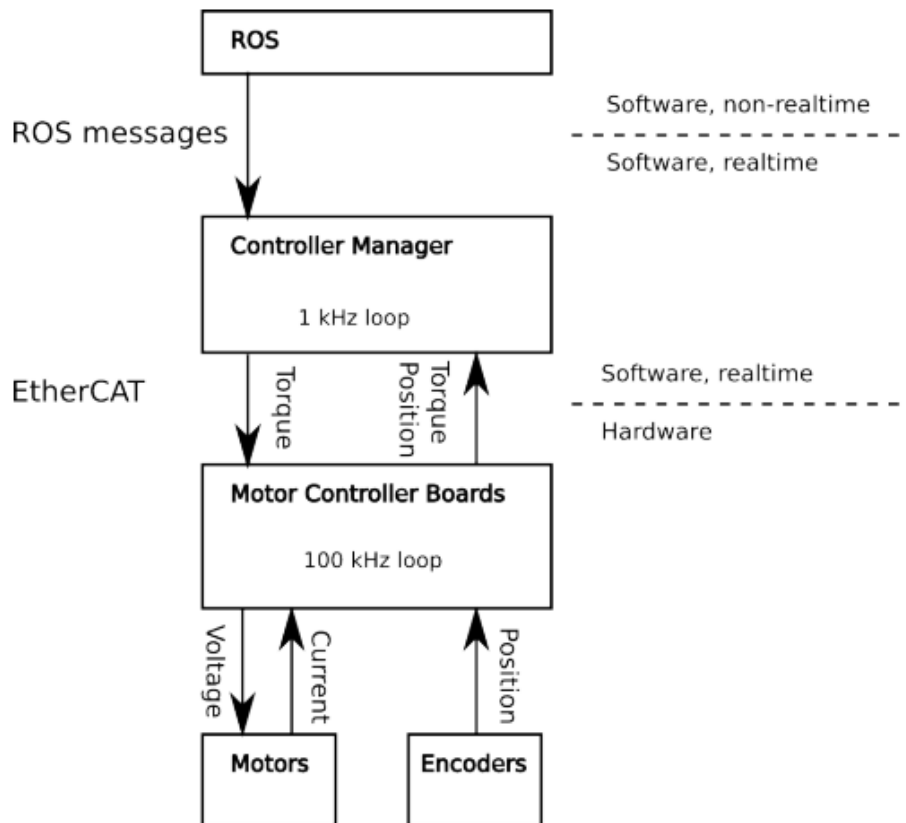


Fig. 2.6 PR2's Control System.



### 2.2.2 Kinect

Kinect [10] is a RGB-D sensor produced by Microsoft for the Xbox 360 game console and Windows PCs. It enables users to control and interact with Xbox without touching joystick or keyboard. Through gesture and speech command, Kinect becomes an input device that can be applied for human robot interaction applications. And most importantly, the stable 3D sensing ability makes it a popular sensor in robotics. Through the infrared and camera mounted on Kinect (see Fig. 2.7), RGB and depth information of the environment can be obtained, and this property supports us to implement our transparent object recognition algorithm. And because the price of Kinect is much cheaper than traditional 3D cameras, it is widely used in robotic systems.

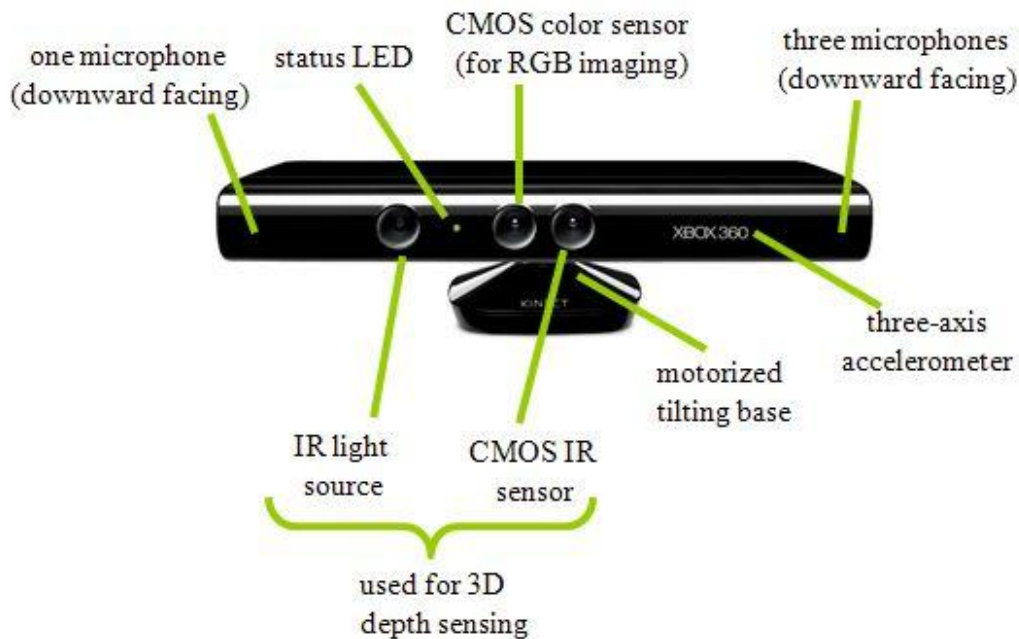


Fig. 2.7 Microsoft's Kinect.



| <b>KINECT TECHNICAL SPECIFICATION</b> |  |
|---------------------------------------|--|
| <b>Sensor</b>                         | Colour and depth-sensing lenses  |
|                                       | Voice microphone array   |
|                                       | Tilt motor for sensor adjustment   |
| <b>Field of View</b>                  | Horizontal field of view: 57 degrees   |
|                                       | Vertical field of view: 43 degrees   |
|                                       | Physical tilt range: $\pm 27$ degrees  |
|                                       | Depth sensor range: 1.2m - 3.5m  |
| <b>Data Streams</b>                   | 320x240 16-bit depth @ 30 frames/sec   |
|                                       | 640x480 32-bit colour @ 30 frames/sec  |
|                                       | 16-bit audio @ 16 kHz  |
| <b>Skeletal Tracking System</b>       | Tracks up to 6 people, including 2 active players                                |
|                                       | Tracks 20 joints per active player   |
|                                       | Ability to map active players to Xbox LIVE Avatars                               |
| <b>Audio System</b>                   | Xbox LIVE party chat and in-game voice chat (requires Xbox LIVE Gold Membership) |
|                                       | Echo cancellation system enhances voice input                                    |
|                                       | Speech recognition in multiple languages   |

Fig. 2.8 Kinect's technical specification.

## Chapter 3 Detection of Transparent Objects



In this chapter, we introduce three kinds of method to detect transparent objects in color image. We introduce Grabcut-based method because it is further used for pose estimation and grasping. However, this method requires RGBD sensor as input sensor. We explore LDA-based method and Deep Learning-based method so that we can detect transparent objects by using a monocular camera. This can be helpful in the case that RGBD sensor cannot be acquired.

### 3.1 Grabcut-Based Method

#### 3.1.1 Overall process of Grabcut-based method

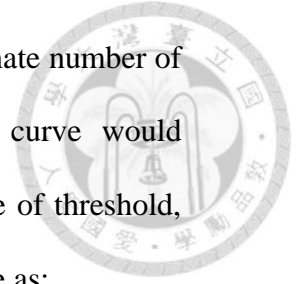
For Grabcut-based method, we use RGB image and depth image of Kinect as the input of algorithm. To retrieve transparent candidates from depth image, we first extract all unknown values from input depth image. Next, we use some morphological operation to remove some noise. Then we extract connected components as possible regions of transparency. However, in this stage, we often have several regions that are non-transparent. In [6], they assume transparent objects are on a table, so they extract the table plane and rule out regions that are not on the table. On the other hand, we take advantage of strong highlight, which is a visual cue of transparent object, to rule out regions that does not contain highlight. We use highlight because every transparent object contains highlight in a scene with light source. Both method works fine, but calculating highlight is easier and efficient. After extracting the regions containing highlight, we use each of these regions as seeds for Grabcut to segment transparent candidates as shown in Fig 3.1.



Fig. 3.1 Top: The scene containing transparent objects.

Bottom: Transparent candidates in the scene.

As for the highlight, in general, transparent object reflects light such that it produces multiple highlights on its surface. Various empirical models can be used to count the local highlight points on a surface, such as Phong model [12], which is commonly used in 3D computer graphics. To find the highlight, a way is to build hypothesized 3D shape and search through a large set of candidate image highlights [1]. By binary threshold, one can detect highlight regions of transparent object, but threshold value is hard to be determined. We then used the method described in [2] as it is an efficient but useful method for searching highlight.



The first step is to threshold image by value from 0 to 255, and estimate number of perimeter pixels for each image after threshold. The slope of the curve would significantly decrease at the value close to 255. To find the critical value of threshold, we produce first order polynomial to fit the straight line of perimeter curve as:

$$P = aT + b$$

Variable T represents the threshold value. We fit the perimeter from threshold 255 to 0, iteratively. For each fit, we estimate the mean square error and plot another curve of error. Finally, we compare the slope of error curve from threshold 255 to 0. When there is a significant increase of slope, the value is the proper value of threshold. As shown in Fig. 3.2, only highlights remain in image after threshold.



Fig 3.2 Three glasses and the detected highlights.

In summary, for retrieving transparent candidates, we improve the method of [6], and the main difference between the previous method and ours is how we rule out possible regions of transparency. We use highlight thus accelerate the process of candidate retrieval.

### 3.1.2 Grabcut

GrabCut [13] is an iterative image segmentation technique based upon the Graph Cut algorithm [14]. GrabCut extends Graph Cut to color images and to incomplete trimaps. See Fig 3.3, first player and football is enclosed in a blue rectangle. Then some

final touchups with white strokes (denoting foreground) and black strokes (denoting background) is made. And we get a nice result from Grabcut algorithm.

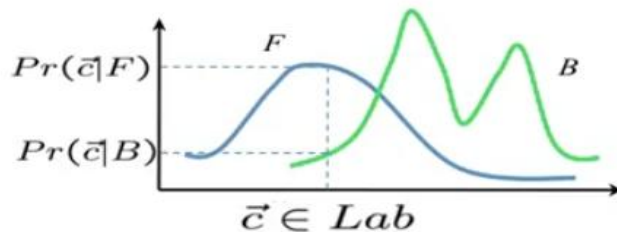


Fig 3.3 Two examples of Grabcut algorithms.

Let us briefly introduce the process of Grabcut algorithm. First, user creates an initial trimap by selecting a rectangle. Pixels inside the rectangle are marked as unknown. Pixels outside of rectangle are marked as known background. Then computer creates an initial image segmentation, where all unknown pixels are tentatively placed in the foreground class and all known background pixels are placed in the background class. Next, Gaussian Mixture Models (GMMs) are created for initial foreground and background classes. Each pixel in the foreground class is assigned to the most likely Gaussian component in the foreground GMM. Similarly, each pixel in the background is assigned to the most likely background Gaussian component. The iterative part comes now, the GMMs are thrown away and new GMMs are learned from the pixel sets created in the previous set. A graph is built and Graph Cut is run to find a new tentative



foreground and background classification of pixels as shown in Fig 3.4.

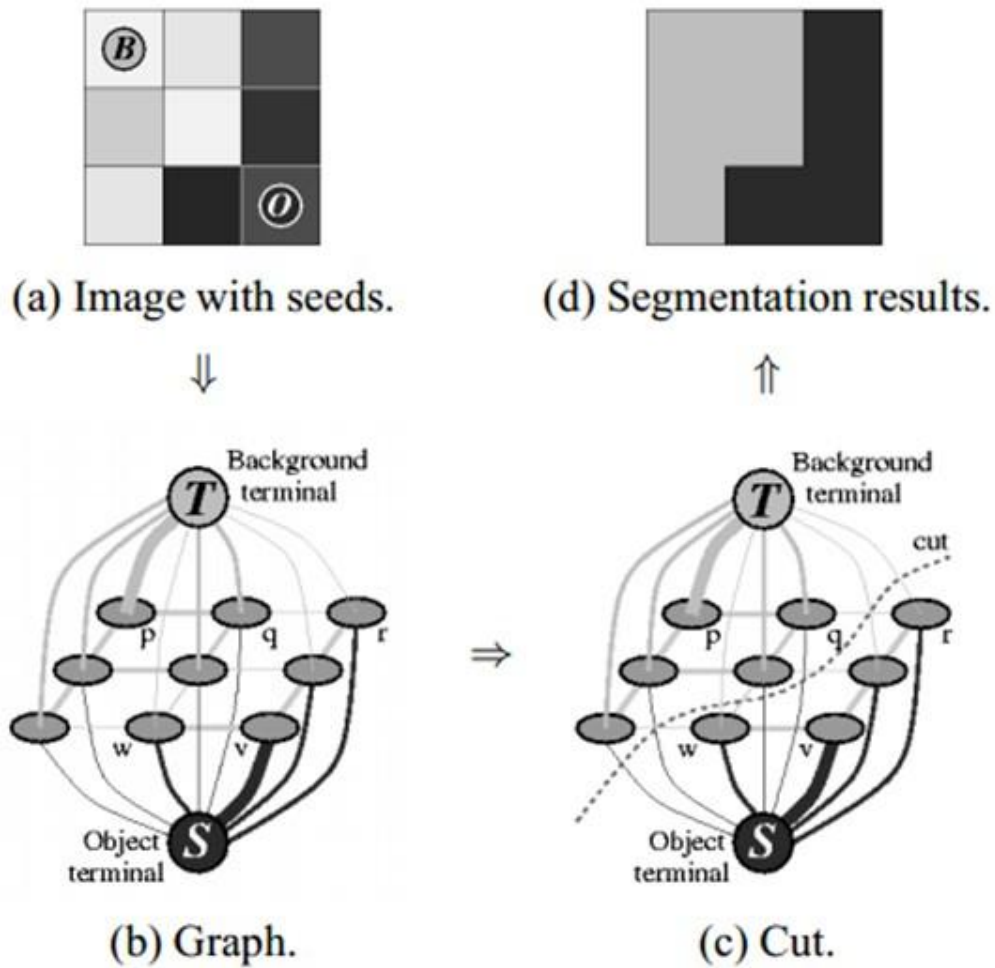


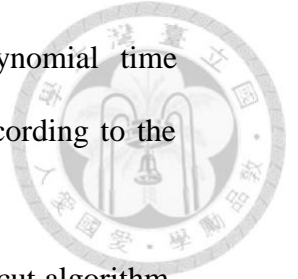
Fig 3.4 An illustration of Graph cut.

The process of Graph Cut algorithm is stated as follows: the user first delineates pixels in the background and foreground regions using a few strokes of an image brush (Figure 3.61). These pixels then become the seeds that tie nodes in the S–T graph to the source and sink labels S and T (Figure 3.4). Seed pixels can also be used to estimate foreground and background region statistics (intensity or color histograms).

The capacities of the other edges in the graph are derived from the region and boundary energy terms, i.e., pixels that are more compatible with the foreground or background region get stronger connections to the respective source or sink; adjacent pixels with greater smoothness also get stronger links. Once the



minimum-cut/maximum-flow problem has been solved using a polynomial time algorithm [15], pixels on either side of the computed cut are labeled according to the source or sink to which they remain connected.



As a result, by feeding the NaN area in depth image as seed for Grabcut algorithm, we can crop the transparent object in the image, and thus we get our transparent candidates. However, since some of the NaN areas in depth image might not be transparent, we need the method stated in section 3.3 to further classify these transparent candidates.

### 3.1.3 Transparent Object Classification

Some transparent candidates are non-transparent but contains highlight, so there may be some candidates that are not transparent, such as the one in the bottom left of Fig. 3.1. As result, we need to further evaluate if these candidates are transparent. So algorithms to detect characteristics possessed by transparent objects are implemented for classification.

- Color Similarity

One of the important features of transparent object is that the color tends to be similar on both side of the edge. Because the object is transparent, the object presents the color of background, which is similar to the color around object.

To check if the colors on both sides are similar, we calculate the histogram of hue in HSV color space on both sides. After retrieving transparent candidate as shown in Fig. 3.5(b), we can calculate the pixels around the candidate, and the result is shown in Fig. 3.5(c). Then we can calculate the hue histogram of pixels inside transparent candidate and pixels around the candidate. Fig. 3.5(d) shows the histogram of the pixels inside candidate and pixels around candidate. To compare the similarity of the two histograms, we view these two histograms as vectors. By comparing the Euclidean distance, we are



able to compare the similarity, if the value of distance is low, then indicate the color on both side are similar.

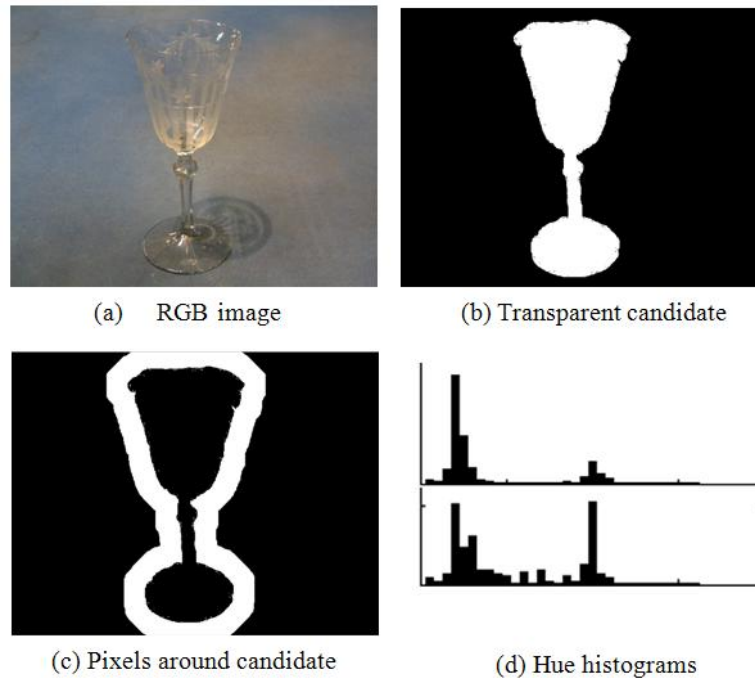


Fig 3.5 The result of computing color similarity.

- Intensity Consistency

Except for the color similarity of transparent objects, yet another characteristic of transparent object is that the pixel intensity around the edge is consistent. In other words, the intensity distribution inside transparent object is constrained by the intensity distribution outside the transparent object. So we use a mask that covers part of the edge, which includes pixels on both side of the edge. Apparently, local standard deviation of pixel intensity inside the mask around boundaries of transparent objects should be a small value. Traditional segmentation methods are difficult to detect the boundaries of transparent object. Based on the candidates retrieved, we are able to get accurate boundaries, and find out the local standard deviation value of transparent boundaries using dynamic mask:



$$\mu = \frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N I(x, y)$$

$$\sigma = \sqrt{\frac{1}{MN} \sum_{x=1}^M \sum_{y=1}^N [I(x, y) - \mu]^2}$$

$$\text{SD-Value} = \sum_{i=1}^C \sigma_i.$$

For the dynamic mask,  $M$  is the length of row,  $N$  is the length of columns, and  $I(x,y)$  is the intensity of pixel at  $(x,y)$ . As the mask moving around the boundaries of transparent candidate, we calculate one local standard deviation at each move,  $C$  is the total times we calculate the local standard deviation. Finally we have a SD-value, when it is small, it indicates the candidate is possible to be transparent object.

After calculating the color similarity value and SD-value, we got two measurements on transparent classification. A simple way to combine the two measurements is to take average, and it works well in our scenario. Using 15 images, we trained the classifier and get the threshold value of 0.6. That is, if a candidate's value is under 0.6, it is considered transparent.

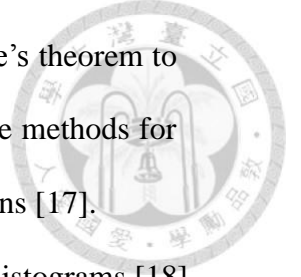
## 3.2 LDA-Based Method

### 3.2.1 Gaussian Mixture Model (GMM)

Gaussian Mixture Model is the base of Latent Dirichlet Allocation, so we make a brief introduction here for understanding LDA in section 3.2.2.

Mixture models are a type of density model which comprise a number of component functions, usually Gaussian. These component functions are combined to provide a multimodal density. They can be employed to model the colors of an object in order to perform tasks such as real-time color-based tracking and segmentation [16]. These tasks may be made more robust by generating a mixture model corresponding to

background colors in addition to a foreground model, and employing Baye's theorem to perform pixel classification. Mixture models are also amenable to effective methods for on-line adaptation of models to cope with slowly-varying lighting conditions [17].



Mixture models are a semi-parametric alternative to non-parametric histograms [18] (which can also be used as densities) and provide greater flexibility and precision in modeling the underlying statistics of sample data. They are able to smooth over gaps resulting from sparse sample data and provide tighter constraints in assigning object membership to color-space regions. Such precision is necessary to obtain the best results possible from color-based pixel classification for qualitative segmentation requirements.

Once a model is generated, conditional probabilities can be computed for color pixels. Gaussian mixture models can also be viewed as a form of generalized radial basis function network in which each Gaussian component is a basis function or 'hidden' unit. The component priors can be viewed as weights in an output layer. Finite mixture models have also been discussed at length elsewhere [19-24] although most of this work has concentrated on the general studies of the properties of mixture models rather than developing vision models for use with real data from dynamic scenes.

Let the conditional density for a pixel  $\xi$  belonging to a multi-colored object  $O$  be a mixture with  $M$  component densities:

$$p(\xi|O) = \sum_{j=1}^M p(\xi|j)P(j)$$

where a mixing parameter  $P(j)$  corresponds to the prior probability that pixel  $\xi$  was generated by component  $j$  and where  $\sum_{j=1}^M p(\xi|j) = 1$ . Each mixture component is a Gaussian with mean  $\mu$  and covariance matrix  $\Sigma$ :



$$p(\xi|j) = \frac{1}{2\pi|\Sigma_j|^{1/2}} \exp\left\{-\frac{1}{2}(\xi-\mu_j)^T \Sigma_j^{-1}(\xi-\mu_j)\right\}$$

For example, the function  $f(x)$  in Fig 3.6 can be approximate by three Gaussian components.

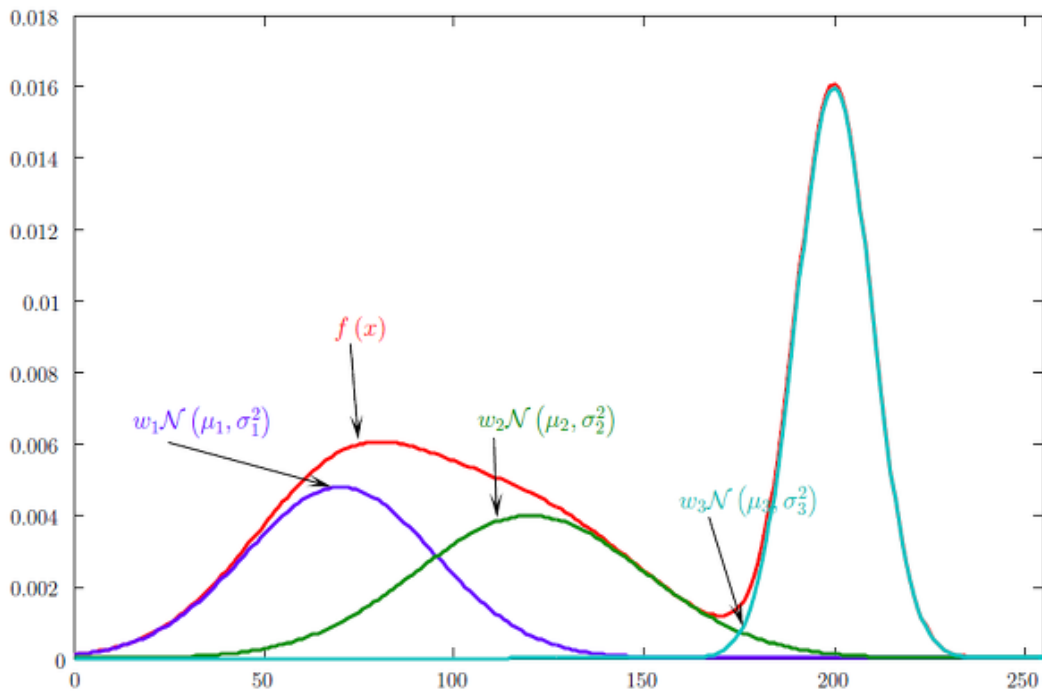
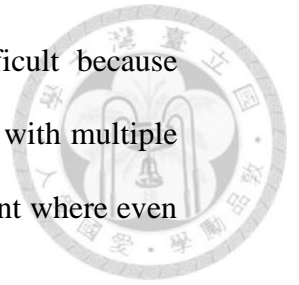


Fig 3.6 An example of GMM approximation.

### 3.2.2 Probabilistic Latent Semantic Analysis (pLSA) & Latent Dirichlet Allocation (LDA)

pLSA and LDA are developed to model text corpora for information retrieval, which can be used in Google's keyword search or natural language processing. Note that LDA is an improved version of pLSA, so we will introduce pLSA first.

pLSA is based on LSA (Latent Semantic Analysis), also known as Latent Semantic Indexing (LSI) literally means analyzing documents to find the underlying meaning or concepts of those documents. If each word only meant one concept, and each concept was only described by one word, then LSA would be easy since there is a simple



mapping from words to concepts. Unfortunately, this problem is difficult because English has different words that mean the same thing (synonyms), words with multiple meanings, and all sorts of ambiguities that obscure the concepts to the point where even people can have a hard time understanding.

The key concept of LSA is to form a word-document co-occurrence matrix, and by doing singular value decomposition (SVD, as shown in Fig 3.7), it can construct the latent semantic space for these documents. Let's see a simple example of LSA by first see the word-document co-occurrence matrix shown in Fig 3.8.

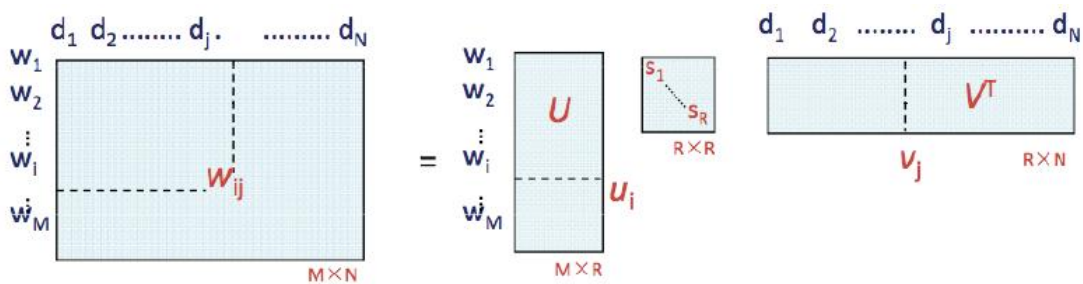


Fig. 3.7 An illustration of Singular Value Decomposition (SVD).

| term     | data | information | retrieval | brain | lung |
|----------|------|-------------|-----------|-------|------|
| document |      |             |           |       |      |
| CS-TR1   | 1    | 1           | 1         | 0     | 0    |
| CS-TR2   | 2    | 2           | 2         | 0     | 0    |
| CS-TR3   | 1    | 1           | 1         | 0     | 0    |
| CS-TR4   | 5    | 5           | 5         | 0     | 0    |
| MED-TR1  | 0    | 0           | 0         | 2     | 2    |
| MED-TR2  | 0    | 0           | 0         | 3     | 3    |
| MED-TR3  | 0    | 0           | 0         | 1     | 1    |

Fig. 3.8 A simple word-document co-occurrence matrix.

We can use SVD to decompose the matrix in Fig 3.8, and get the result shown in Fig 3.9. One can observe that because the rank of the original word-document co-occurrence matrix's rank is 2, so there are two singular values (two topics also). And since different kind of documents in this toy example have non-overlapped words, so

the result is very good.

$$\begin{array}{c} \text{CS} \\ \updownarrow \\ \text{MED} \end{array}
 \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}
 =
 \begin{bmatrix} 0.18 & 0 \\ 0.36 & 0 \\ 0.18 & 0 \\ 0.90 & 0 \\ 0 & 0.53 \\ 0 & 0.80 \\ 0 & 0.27 \end{bmatrix}
 \times
 \begin{bmatrix} 9.64 & 0 \\ 0 & 5.29 \end{bmatrix}
 \times
 \begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \end{bmatrix}$$

**doc by term**
**doc by topic**
**topic by topic**
**topic by term**

Fig. 3.9 Use SVD to decompose the example word-document matrix.

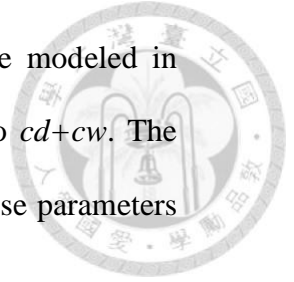
However, LSA has some problems that will be inconvenient for calculation. The major problem of LSA is the bad handling of large scale information retrieval. For example, if we have millions of documents, the word-document co-occurrence matrix would be very huge, and computation time of SVD would be very long.

Compared to standard latent semantic analysis which stems from linear algebra and downsizes the occurrence tables (via singular value decomposition), pLSA is based on a mixture decomposition derived from a latent class model.

Considering observations in the form of co-occurrences  $(w,d)$  of words and documents, pLSA models the probability of each co-occurrence as a mixture of conditionally independent multinomial distributions:

$$P(w, d) = \sum_c P(c)P(d|c)P(w|c) = P(d) \sum_c P(c|d)P(w|c)$$

where  $c$  is the words' topic. The first formulation is the symmetric formulation, where  $w$  and  $d$  are both generated from the latent class  $c$  in similar ways (using the conditional probabilities  $P(w|c)$  and  $P(d|c)$ ). And the second formulation is the asymmetric formulation, where, for each document  $d$ , a latent class is chosen conditionally to the document according to  $P(c|d)$ , and a word is then generated from that class according to  $P(w|c)$ . Although we have used words and documents in this



example, the co-occurrence of any couple of discrete variables may be modeled in exactly the same way. As a result, the number of parameters is equal to  $cd+cw$ . The number of parameters grows linearly with the number of documents. These parameters are learned using the EM algorithm.

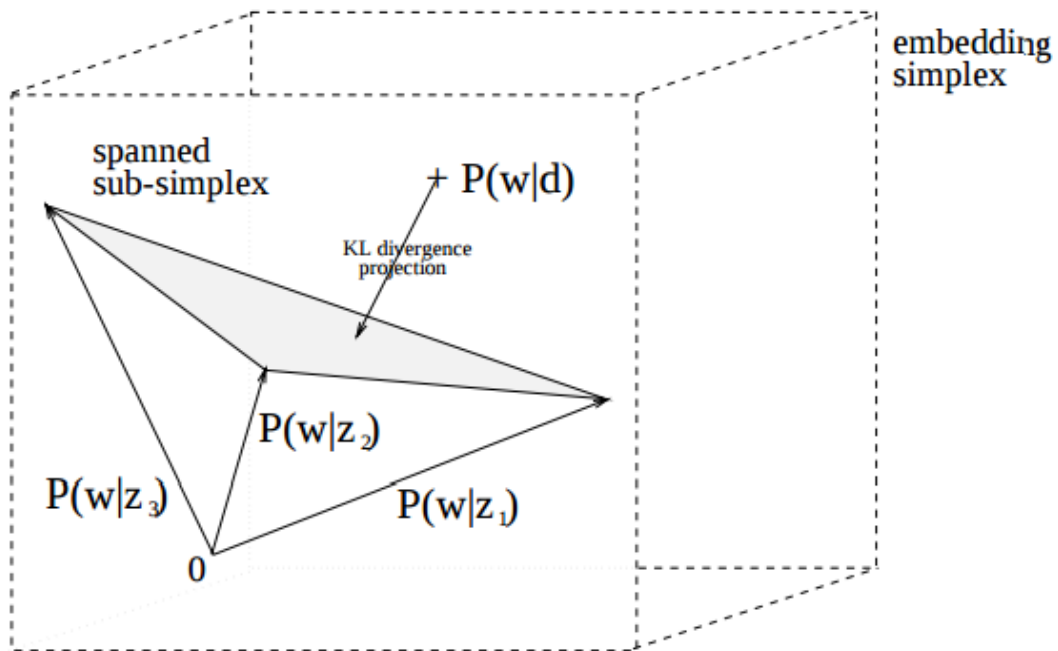


Fig 3.10 A geometric view of pLSA ( $z_1$  to  $z_3$  are three latent topics).

In Fig 3.10,  $P(w|z_1) \sim P(w|z_3)$  can be viewed as three vector spanning a linear space to describe the real document and word  $P(w|d)$ .

Based on pLSA, LDA is a more powerful model to discover the latent topic of documents. Let us consider a simple example, suppose you have the following set of sentences:

- A service robot should be able to grasp transparent objects.
- I don't care if a service robot can recognize transparent objects or not, I just want it to cook for me.
- Taiwan needs to develop robotic industry.





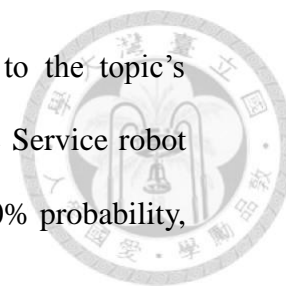
- Taiwan's economy is stagnant and need a new industry such as robotics to make economic growth.
- If we can develop a service robot capable of grasping transparent objects, we can help the robot industry in Taiwan.

Given these sentences and asked for 2 topics, LDA might produce something like

- **Sentences 1 and 2:** 100% Topic A
- **Sentences 3 and 4:** 100% Topic B
- **Sentence 5:** 60% Topic A and 40% Topic B.
- **Topic A:** 40% service robot, 25% grasp, 25% transparent object, 5% cook, ...  
(at which point, we could interpret topic A to be about service robot)
- **Topic B:** 40% Taiwan, 30% industry, 20% robotic, ... (at which point, we could interpret topic B to be about Taiwan's economy)

LDA perform this discovery by representing documents as mixtures of topics that spit out words with certain probabilities. It assumes that documents are produced in the following fashion: when writing each document, you

- Decide on the number of words  $N$  the document will have (say, according to a Poisson distribution).
- Choose a topic mixture for the document (according to a Dirichlet distribution over a fixed set of  $K$  topics). For example, assuming that we have the two service robot and Taiwan's economy topics above, you might choose the document to consist of  $1/3$  service robot and  $2/3$  Taiwan's economy.
- Generate each word in the document by:
  - First picking a topic (according to the multinomial distribution that you sampled above; for example, you might pick the service robot topic with  $1/3$  probability and the Taiwan's economy topic with  $2/3$  probability).



- Using the topic to generate the word itself (according to the topic's multinomial distribution). For example, if we selected the Service robot topic, we might generate the word “service robot” with 40% probability, “grasp” with 25% probability, and so on.

Assuming this generative model for a collection of documents, LDA then tries to backtrack from the documents to find a set of topics that are likely to have generated the collection.

### 3.2.3 LDA for transparent object detection

In [3], they proposed to discover latent topics which are characteristic of particular transparent patches and quantize the SIFT space into transparent visual words according to the latent topic dimensions. We did not further improve their method, but for completeness of this thesis, we will briefly introduce how they employ LDA to discover the visual word of transparent objects for transparent object recognition.

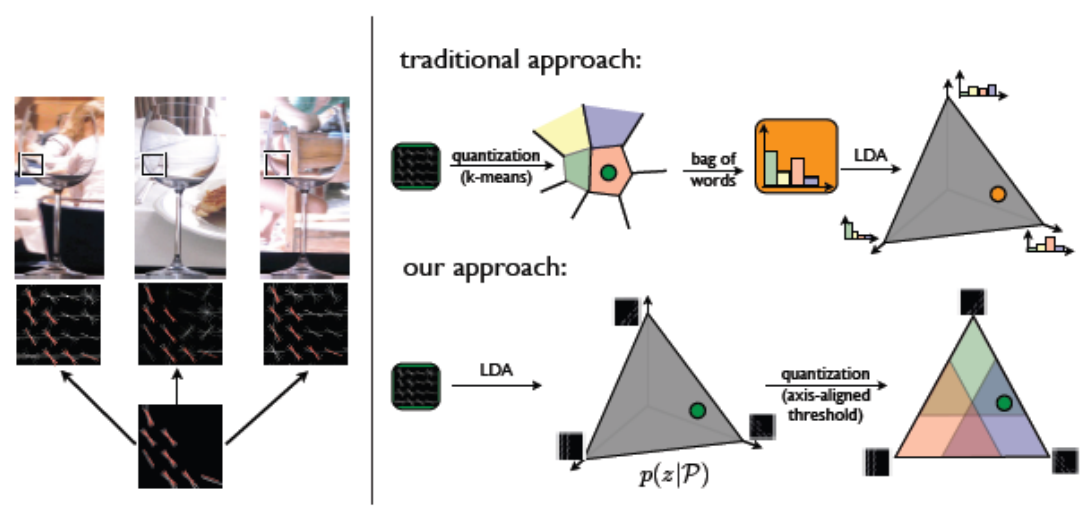


Fig 3.11 The method used in [3].

As can be seen in the left of Fig 3.11, there are images of a transparent object in different environments. A point on the object is highlighted in each image, and the local orientation edge energy map is shown. While the background dominates the local patch,

there is a latent structure that is discriminative of the object. The model proposed in [3] finds local transparent structure by applying a latent factor model (e.g., LDA) before a quantization step. In contrast to previous approaches that applied such models to a quantized visual word model, they apply them directly to the SIFT representation, and then quantize the resulting model into descriptors according to the learned topic distribution.

This is because local transparent patch appearance can be understood as a combination of different processes that involve illuminants in the scene, overall 3D structure, as well as the geometry and material properties of the transparent object. Many of these phenomena can be approximated with an additive image formation model, subject to certain deformations. The detailed method of employing LDA to recognize the visual word of transparent objects will not be stated here, but the result is shown in Fig 3.12.

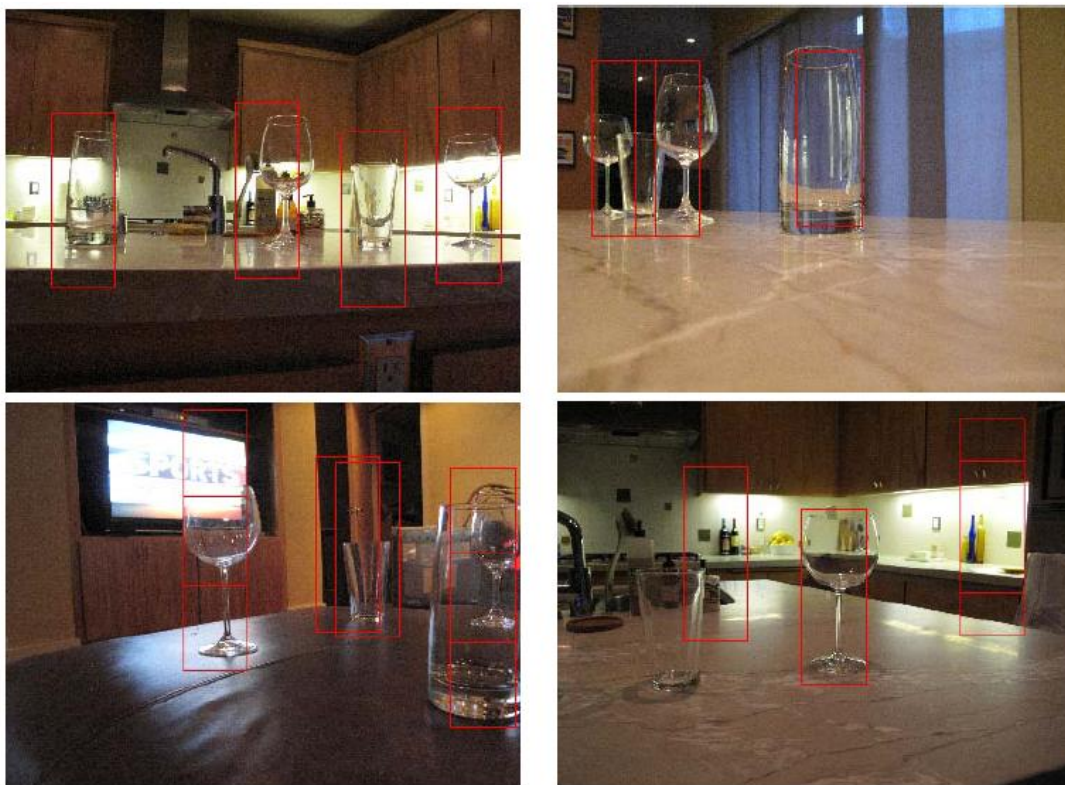


Fig 3.12 The detection result in [3].



### 3.3 Deep Learning-Based Method

Deep Learning is the hottest trend now in AI and Machine Learning. It is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using model architectures, with complex structures or otherwise, composed of multiple non-linear transformations.

Deep learning is part of a broader family of machine learning methods based on learning representations of data. An observation (e.g., an image) can be represented in many ways such as a vector of intensity values per pixel, or in a more abstract way as a set of edges, regions of particular shape, etc. Some representations make it easier to learn tasks (e.g., face recognition or facial expression recognition) from examples. One of the promises of deep learning is replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction.

Research in this area attempts to make better representations and create models to learn these representations from large-scale unlabeled data. Some of the representations are inspired by advances in neuroscience and are loosely based on interpretation of information processing and communication patterns in a nervous system, such as neural coding which attempts to define a relationship between the stimulus and the neuronal responses and the relationship among the electrical activity of the neurons in the brain.

Various deep learning architectures such as deep neural networks, convolutional deep neural networks, deep belief networks and recurrent neural networks have been applied to field of computer vision where they have been shown to produce state-of-the-art results on various tasks.

As a result, we would like to explore the possibility to use this technique on

recognizing transparent objects.

### 3.3.1 Deep Neural Network & Convolutional Neural Network

Deep neural network is special group of neural network. So before introducing deep neural network, we should address the definition of a neural network.

Neural Network is a family of statistical learning models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Since the real function is unknown, neural network takes a big amount of pairs of input and corresponding output to automatically learn the relationship between the given inputs and outputs. This is possible because artificial neural networks are generally presented as systems of interconnected "neurons" which send messages to each other. And the connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs so that it is capable of learning.

For example, a neural network for handwriting recognition is defined by a set of input neurons which may be activated by the pixels of an input image. After being weighted and transformed by a function (determined by the network's designer), the activations of these neurons are then passed on to other neurons. This process is repeated until finally, an output neuron is activated. This determines which character was read. An illustration of a single hidden layer neural network is presented in Fig 3.13.



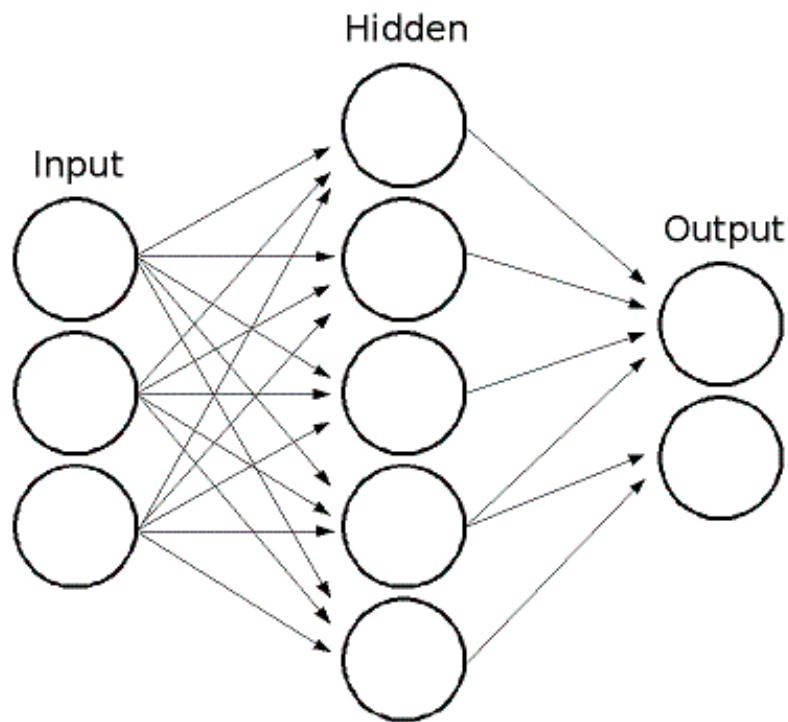
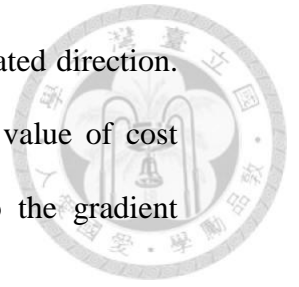


Fig 3.13 An illustration of a single hidden layer neural network.

As shown in Fig 3.13, the first layer has input neurons which send data via synapses to the hidden layer of neurons, and then via more synapses to the third layer of output neurons. More complex systems will have more layers of neurons, some having increased layers of input neurons and output neurons. The synapses store parameters called "weights" that manipulate the data in the calculations.

As for the learning process, training a neural network model essentially means selecting one model from the set of allowed models that minimizes the cost criterion. There are numerous algorithms available for training neural network models; most of them can be viewed as a straightforward application of optimization theory and statistical estimation.

Note that most of the algorithms used in training artificial neural networks employ some form of gradient descent, using back-propagation to compute the actual gradients. This is done by simply taking the derivative of the cost function with respect to the



network parameters and then changing those parameters in a gradient-related direction. And since the gradient direction equals the direction of increasing the value of cost function, the optimization process would take the direction inverse to the gradient direction to find the parameters that minimize the cost function.

Now we know the definition of neural networks, we can now introduce the difference between common neural network and deep neural network. Usually, the common neural network has only one or two hidden layers. While deep neural network has more hidden layers than common neural network, or referred to be shallow neural network in Fig 3.14. Note that the H1 in Fig 3.14 means hidden layer 1.

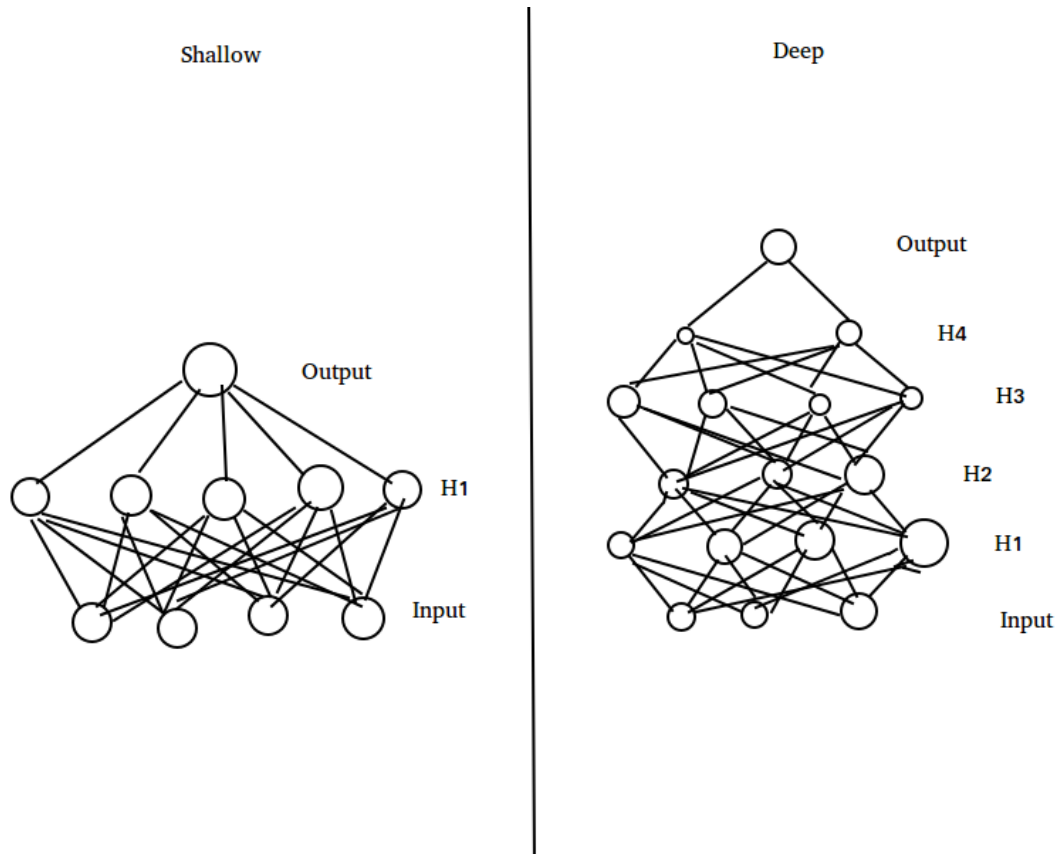
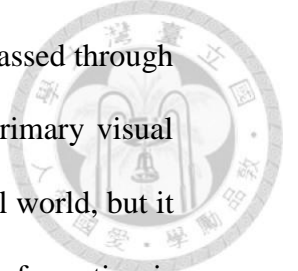


Fig 3.14 The difference between shallow and deep neural network.

In deeper ANNs, each “layer” (i.e., H1, H2, etc) learns to extract important features about the layer before. This is not unlike the brain, especially when it comes to vision. The eye converts light energy into electrical energy that the brain uses for



communication via photoreceptors in the retina. This information is then passed through different layers of in the brain. The first layer in the visual system is primary visual cortex (V1). V1 is most famous for extracting oriented edges in your visual world, but it does a lot more than that of which we won't explain in this thesis. This information is then passed onto different layers throughout the visual system. But it's important to keep in mind that biological nervous systems are substantially more complicated than ANNs. In short summary, deep neural networks are neural networks that contain more hidden layers. However, fully connected deep neural network does not take advantage of the spatial information in image. As a result, for image-related tasks such as image classification, image captioning and video recognition, convolutional neural network is a better choice.

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. A common architecture of CNN is shown in Fig 3.15.



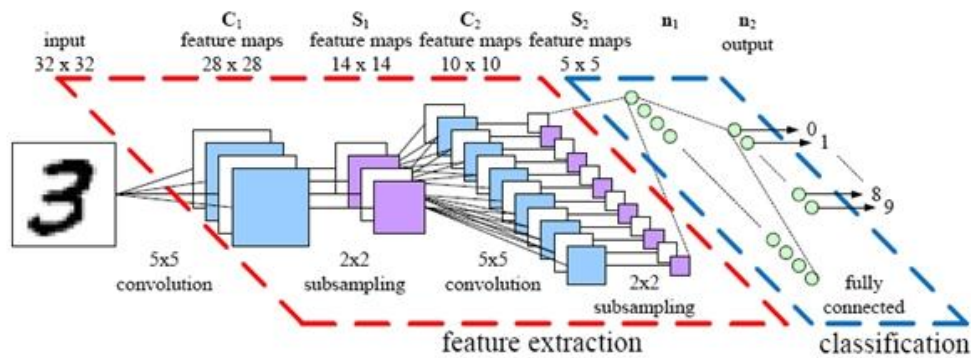


Fig 3.15 The architecture of common convolutional neural network.

### 3.3.2 Regions with Convolutional Neural Network (R-CNN)

Although CNN is very powerful in image classification task, it cannot handle tasks like object detection and segmentation. It is because CNN take the whole image as input and does not deal with the local patches of image. As a result, R-CNN [29] is proposed to deal with the problem mentioned above.

The overall process of R-CNN is shown in Fig 3.16:

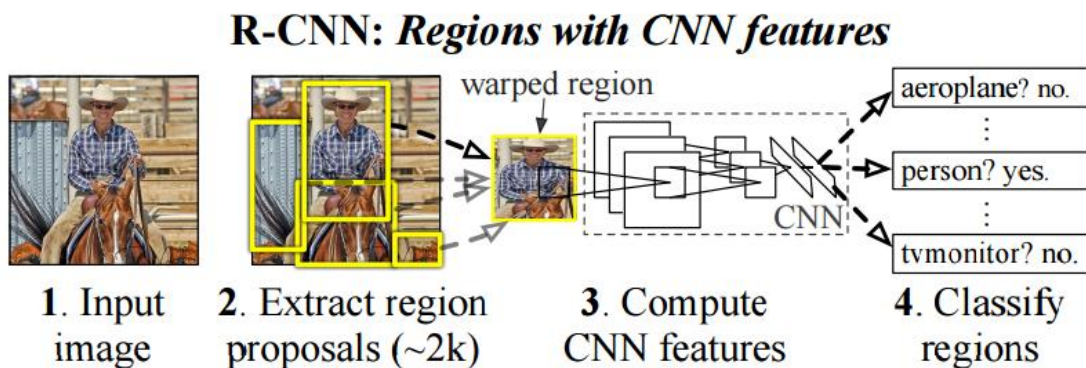
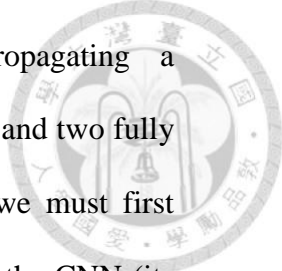


Fig 3.16 Object detection system overview.

R-CNN solve the CNN localization problem by operating within the “recognition using regions” paradigm [30], which has been successful for both object detection [31] and semantic segmentation [32].

For feature extraction, R-CNN extracts a 4096-dimensional feature vector from each region proposal using the Caffe [33] implementation of the CNN described by



Krizhevsky et al. [34]. Features are computed by forward propagating a mean-subtracted  $227 \times 227$  RGB image through five convolutional layers and two fully connected layers. In order to compute features for a region proposal, we must first convert the image data in that region into a form that is compatible with the CNN (its architecture requires inputs of a fixed  $227 \times 227$  pixel size). Of the many possible transformations of our arbitrary-shaped regions, R-CNN uses the simplest. Regardless of the size or aspect ratio of the candidate region, it warp all pixels in a tight bounding box around it to the required size. Prior to warping, the tight bounding box is dilated so that at the warped size there are exactly  $p$  pixels of warped image context around the original box ( $p = 16$ ).

At test time, R-CNN generates around 2000 category-independent region proposals for the input image, extracts a fixed-length feature vector from each proposal using a CNN, and then classifies each region with category-specific linear SVMs. We use a simple technique (affine image warping) to compute a fixed-size CNN input from each region proposal, regardless of the region's shape.

### 3.3.3 Selective Search

The first problem in the testing stage is to select the candidate regions for scoring, the techniques used in R-CNN is selective search [31]. Selective search addresses the problem of generating possible object locations for use in object recognition. They introduce selective search which combines the strength of both an exhaustive search and segmentation. Like segmentation, they use the image structure to guide the sampling process. Like exhaustive search, the paper aim to capture all possible object locations. By using selective search, we don't need to exhaustively enumerate all possible regions for scoring.

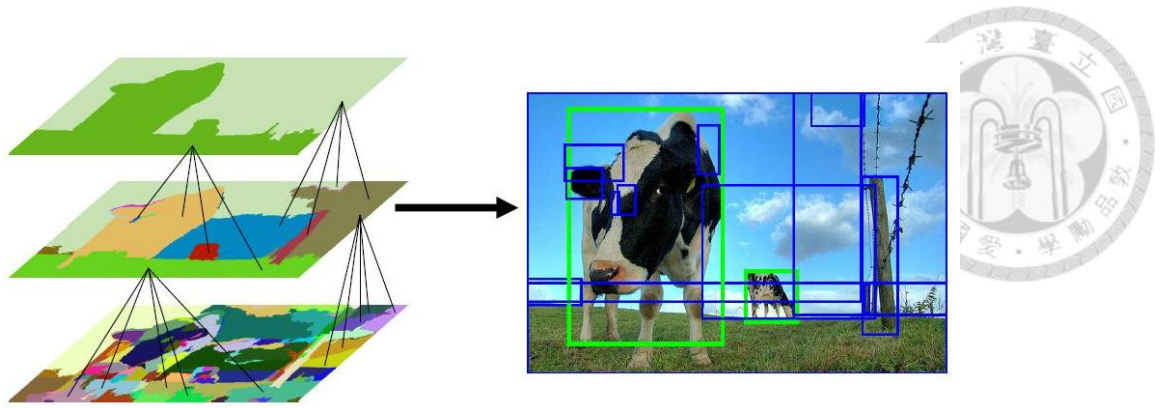


Fig. 3.17 An illustration of selective search.

Since we are interested in using this method on recognizing transparent objects, we have to test if selective search can crop transparent object or not. We use Fig 3.18 as input to selective search and get the result in Fig 3.19.



Fig. 3.18 Our input for selective search.





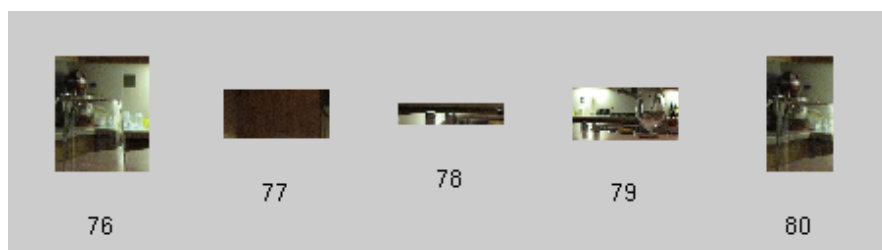










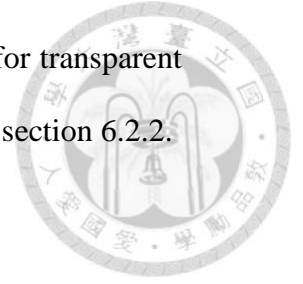




Fig 3.19 The result of selective search on our example.

As shown in Fig 3.19, some proposals (in red rectangle) are good, although there

are still many regions that are not transparent, the result is good enough for transparent object detection. The experiment of detection with R-CNN is presented in section 6.2.2.



## Chapter 4 Pose Estimation of Transparent Objects

In this chapter we introduce the algorithm for pose estimation. The overall process is shown in Fig 4.1. Our algorithm is template-based method, so it needs a predefined 3D model of the transparent object we want to detect. As a result, we divide the process into two separated stages – training stage and testing stage. In training stage, the goal is to prepare 3D model of transparent objects and store some useful information in a database for matching the model in the image containing transparent objects. In testing stage, the information stored in the database will be used to estimate the pose of transparent objects. The two stages will be explained in this chapter.

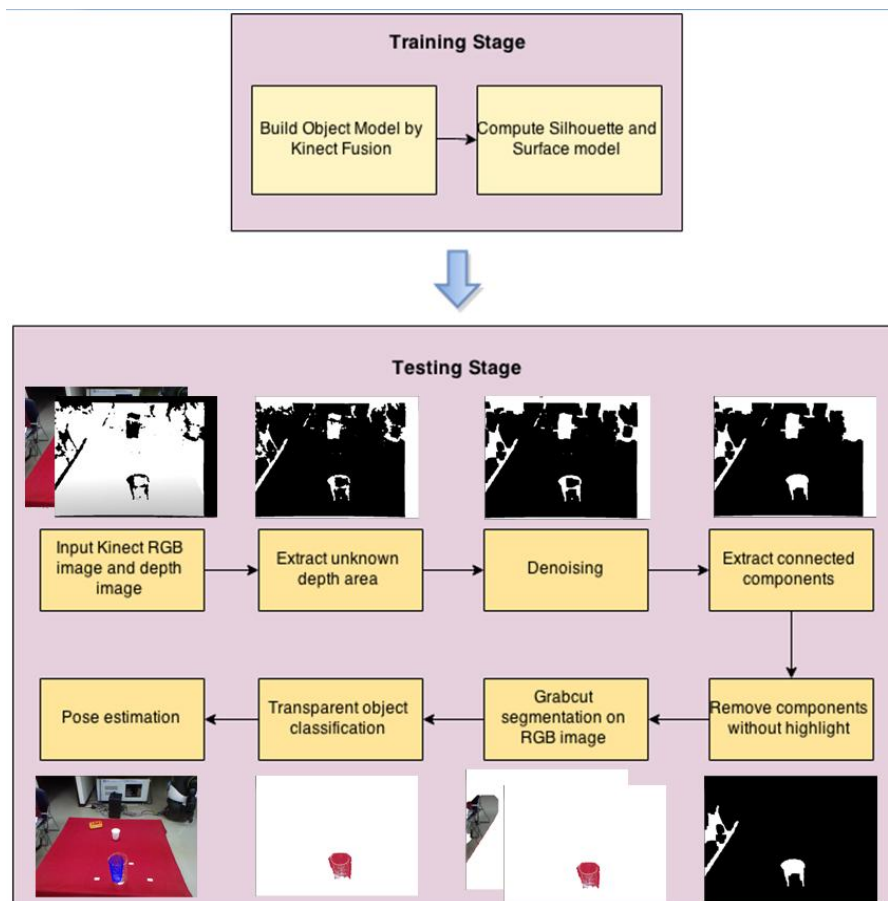


Fig. 4.1 Flowchart of Grabcut-based pose estimation.



## 4.1 Training

### 4.1.1 Model Construction

This algorithm needs a predefined 3D model of the transparent object you want to detect. So you need to provide your model, either by using KinectFusion to construct one or downloading from web.

If we want to use KinectFusion to reconstruct the model, we should first make the transparent object non-transparent. This can be done by paint the object or wrap some paper on it, as shown in Fig 4.2.



Fig. 4.2 A transparent object and its wrapped-up copy.

After we have the non-transparent copy of object, we can put this the non-transparent copy on top of a table, and use KinectFusion provided in Windows operating system for construction. (See Fig 4.3)

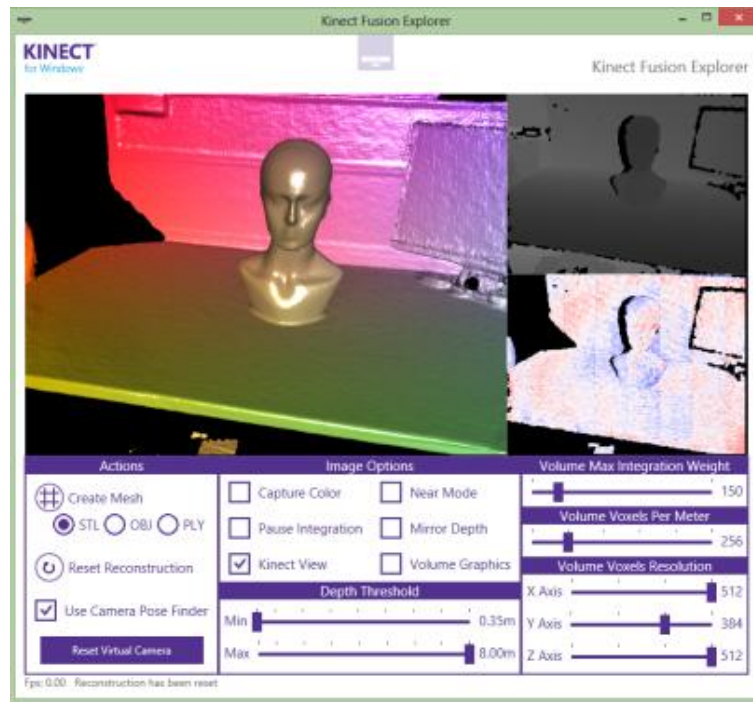


Fig. 4.3 The KinectFusion app in Windows.

One of the constructed models is shown in Fig 4.4. Due to the imperfection of KinectFusion, the model's surface is not very smooth. As a result, we'll introduce another way of model construction by downloading model from web.

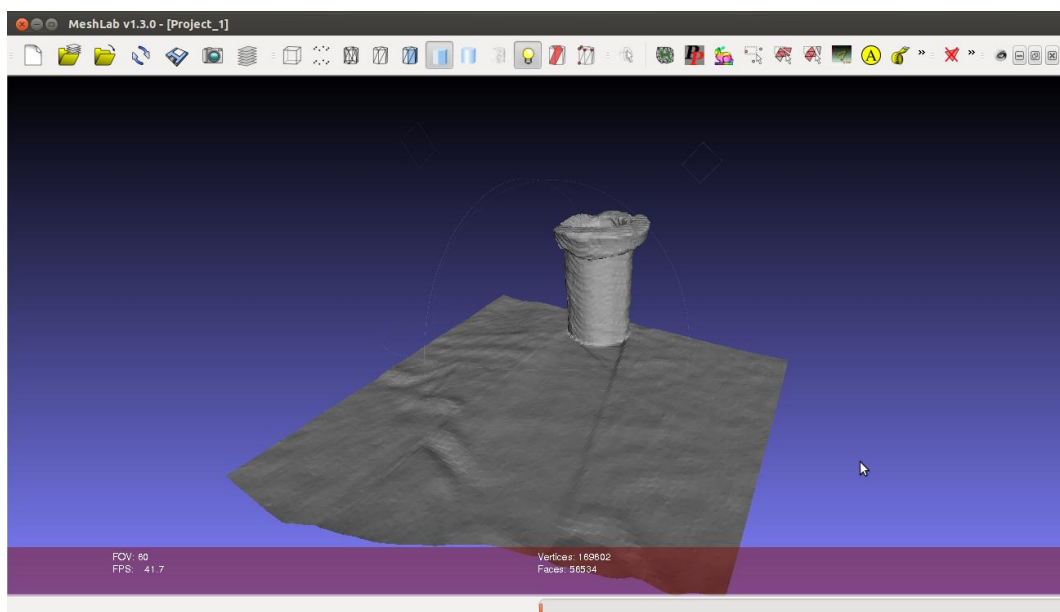
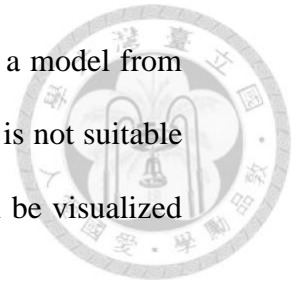


Fig. 4.4 An illustration of the built model.

We use a model of test tube as example to introduce how download a model from the web. The first step is to download a PLY model. However, this model is not suitable for silhouettes generation because the point is too sparse. The model can be visualized in Meshlab, as shown in Fig 4.5.



The reason that the sparse model is not suitable for silhouette generation is stated in the bottom of section 4.1.2.

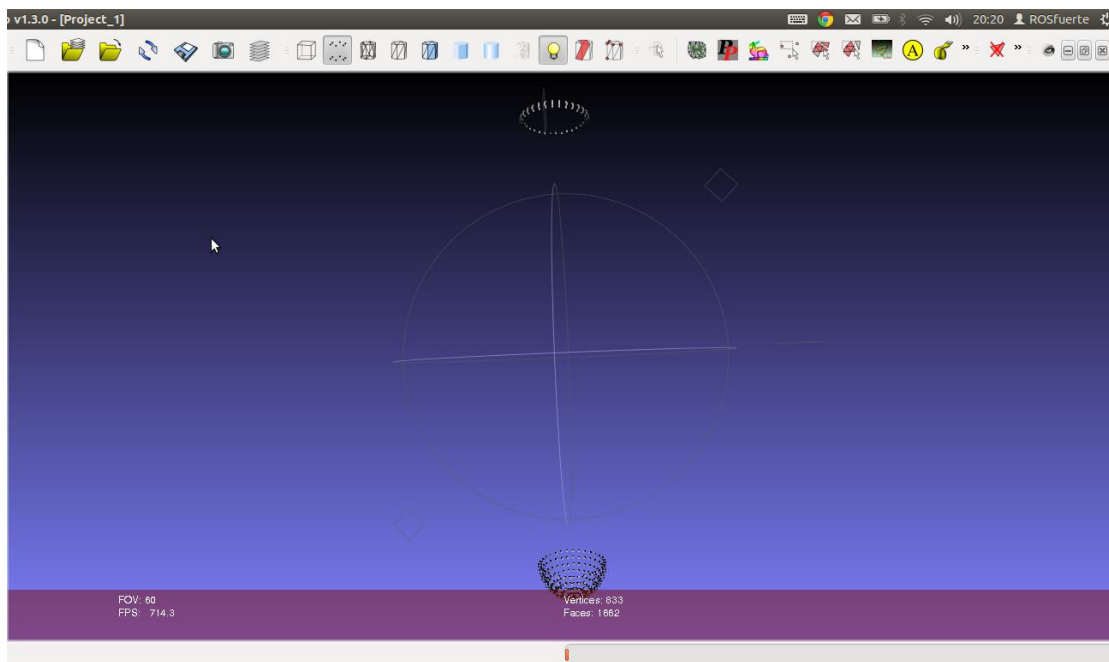


Fig. 4.5 The downloaded model is very sparse.

To solve the sparsity, one can use the points adding function provided in Meshlab (Filters -> Remeshing, simplification and reconstruction -> Subdivision surfaces: Midpoint). The result is shown in Fig. 4.6.

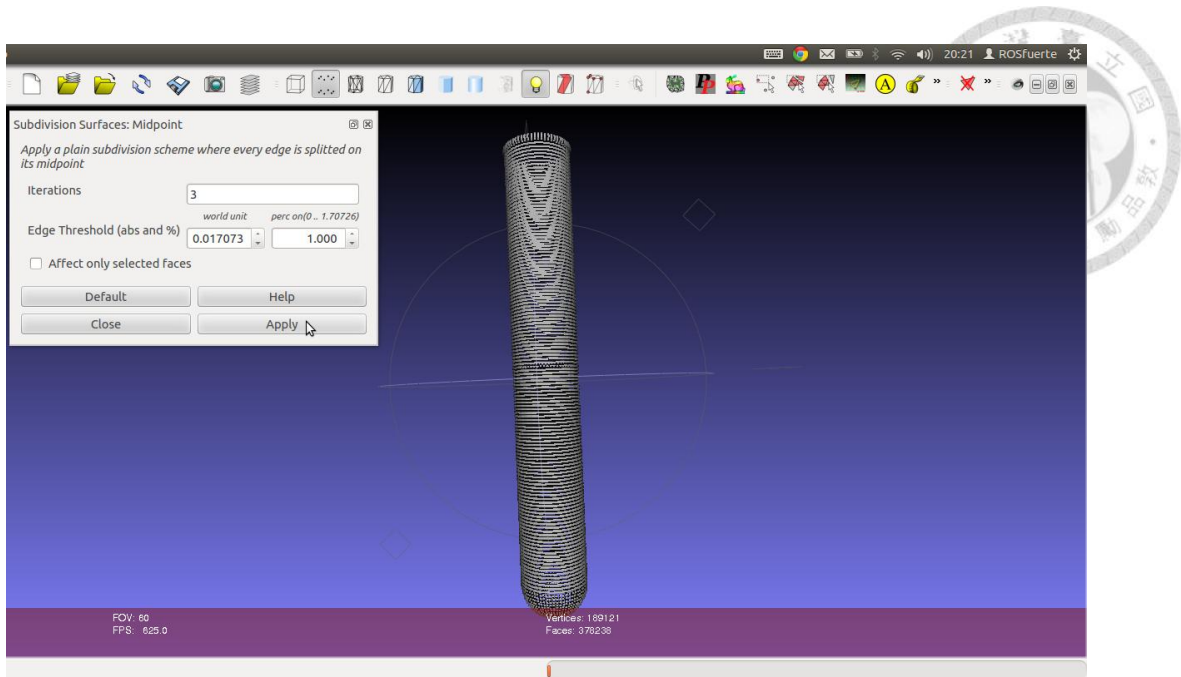


Fig. 4.6 Applying midpoint adding algorithm to solve for sparsity.

However, the point added model usually contains too many points. One can down sample the model to around 1000 vertices by sub-sampling function provided in Meshlab (Filters->Sampling->Mesh Elements Subsampling). The result of sub-sampling is shown in Fig 4.7.

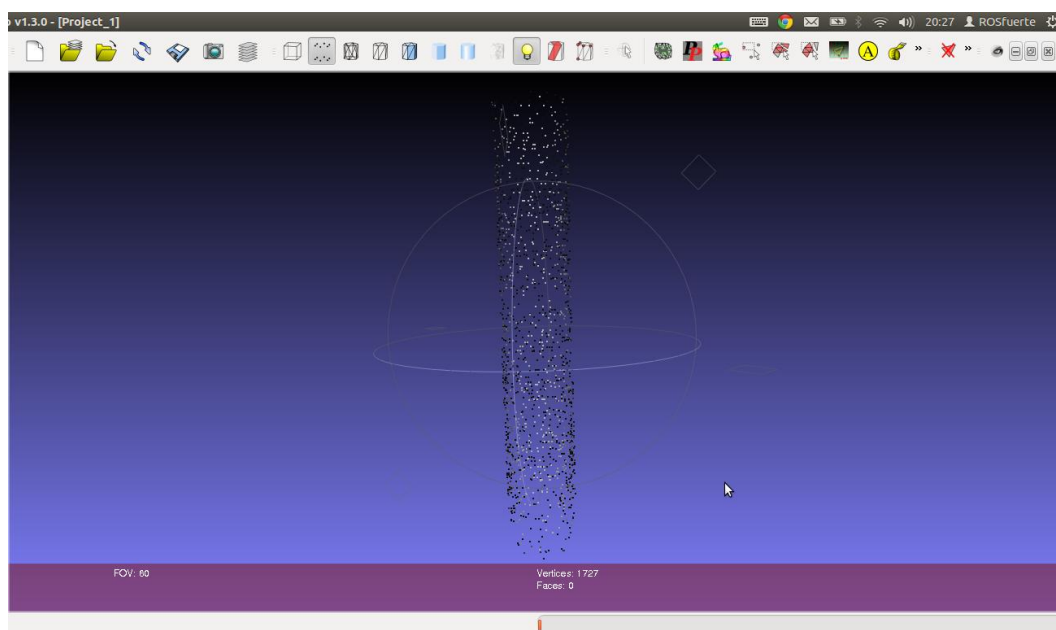


Fig. 4.7 Downsample the model to around 1000 points.

If the 3D model contains too many points, it will cause the huge memory usage and computation power during runtime. Normally, after adding midpoint, the model contains around 200,000 points, this will consume around more than 8 GB memory space during runtime, and the hardware of general laptop cannot handle that. By an empirical study, we found that a 3D model should contain around 1000 points.

Now the model is almost ready, and one needs not forget that to scale the model to the same size as the real transparent object. This is important because the pose estimation algorithm estimate the pose of transparent object by fitting the model in the test image. And this can be done by using the scaling function in Meshlab (Filters -> Normals, Curvatures and Orientation -> Transform: Scale).

#### 4.1.2 Silhouettes Generation

Once we have the model, the training algorithm will rotate the 3D model in different viewpoint, and store the silhouette of each viewpoint in database. (The silhouettes are different in different viewpoints, as shown in Fig 4.8) This is because in testing stage, the test silhouette (the silhouette in the scene) can be matched to the silhouettes in database (training silhouettes). The best match provide hint about the pose of test silhouette.

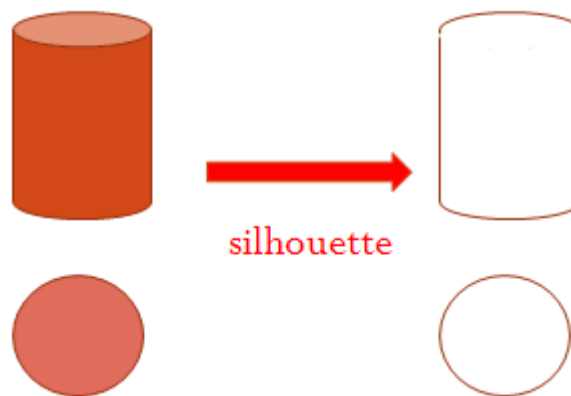
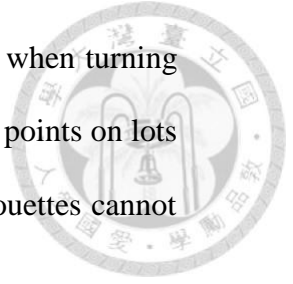


Fig 4.8 An illustration of silhouette generation.



The sparse model is not suitable for silhouette generation is because when turning the model in different viewpoint, the sparse model might contain very few points on lots of silhouettes. This make pose estimation fail because many training silhouettes cannot be used to match the test silhouette.



## 4.2 Testing

### 4.2.1 Test Silhouette Detection

In testing stage, first the depth map from Kinect is used to search the candidate region of transparency. The idea is that transparent object cause NaN in depth map (black region in depth map), so the NaN regions in depth map can be candidates of transparency. As described in section 3.1, once the candidate regions are fetched, GrabCut is used to crop them.

However, some of the candidates might not be transparent, so we use a transparent classification algorithm to further classify between the transparent candidates and get test silhouettes.

### 4.2.2 Initial Pose Estimation

The test silhouette is now at hand. As mentioned earlier in section 4.1, we build the model and store many silhouettes of different  $R_x$ ,  $R_y$ . To determine the pose in the test image, first we want to find the translation and  $R_z$  for each silhouette in the database. After finding the geometric relationship between the silhouette in the test image and each of the silhouettes in database, we use Chamfer Distance [24] to evaluate which relationship best fits to the measured silhouette.

So firstly, we would like to find the translation and  $R_z$  for each silhouette in the database. To do that, a two-dimensional similarity transform between train and test silhouettes should be estimated. This can be estimated by Procrustes Analysis [25]. This



2D shape matching algorithm includes three steps – centering, scaling and rotation. 2D translation is estimated by centering (aligning centroids):

$$(\bar{x}, \bar{y}) = \frac{1}{n} \sum_{i=1}^n (x_i, y_i)$$

, where n is the number of points in a silhouette.

Next, scaling is to align the scatters of the points in train and test silhouettes:

$$s = \sqrt{\frac{1}{n} \sum_{i=1}^n [(x_i - \bar{x})^2 + (y_i - \bar{y})^2]}$$

, then the rotation can be estimated using 2D-2D ICP [26].

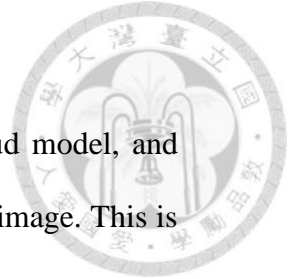
Now we can proceed to compute  $R_z$  and 3D translation which maps the point of the stored 3D model to the location in the test image. However, as mentioned in [6], there is no such transform under the perspective camera model. So the weak perspective projection model [27] is used. This model assumes all points of the 3D model have the same depth. As a result, the point on the model should be put into the right hand side of:

$$\frac{1}{\bar{z}} SK \begin{bmatrix} x \\ y \\ \bar{z} \end{bmatrix} = \frac{1}{\bar{z} + t_z} K \begin{bmatrix} r_{11} & r_{12} & 0 & t_x \\ r_{21} & r_{22} & 0 & t_y \\ 0 & 0 & 1 & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ \bar{z} \\ 1 \end{bmatrix}$$

where  $K$  is the matrix of intrinsic parameters,  $S$  is the similarity transformation obtained from Procrustes Analysis. Note that the equation should be solved for all  $x$  and  $y$  simultaneously. And

$$[R_z \quad t] = \begin{bmatrix} r_{11} & r_{12} & 0 & t_x \\ r_{21} & r_{22} & 0 & t_y \\ 0 & 0 & 1 & t_z \end{bmatrix}$$

After calibration and Procrustes Analysis, we have  $S$  and  $K$ . So we can solve the translation and  $R_z$  for each training silhouette. If a training silhouette match well to the testing silhouette, we can get some plausible pose for further pose refinement.



### 4.2.3 Pose Refinement

We have silhouette and surface edges from provided 3D point cloud model, and they should be aligned with the test silhouette and canny edges in the test image. This is a problem of 3D-2D registration and we use a robust variant of Levenberg-Marquardt Iterative Closest Point (LM-ICP [28]) to solve it. The result of this algorithm is the refined pose.

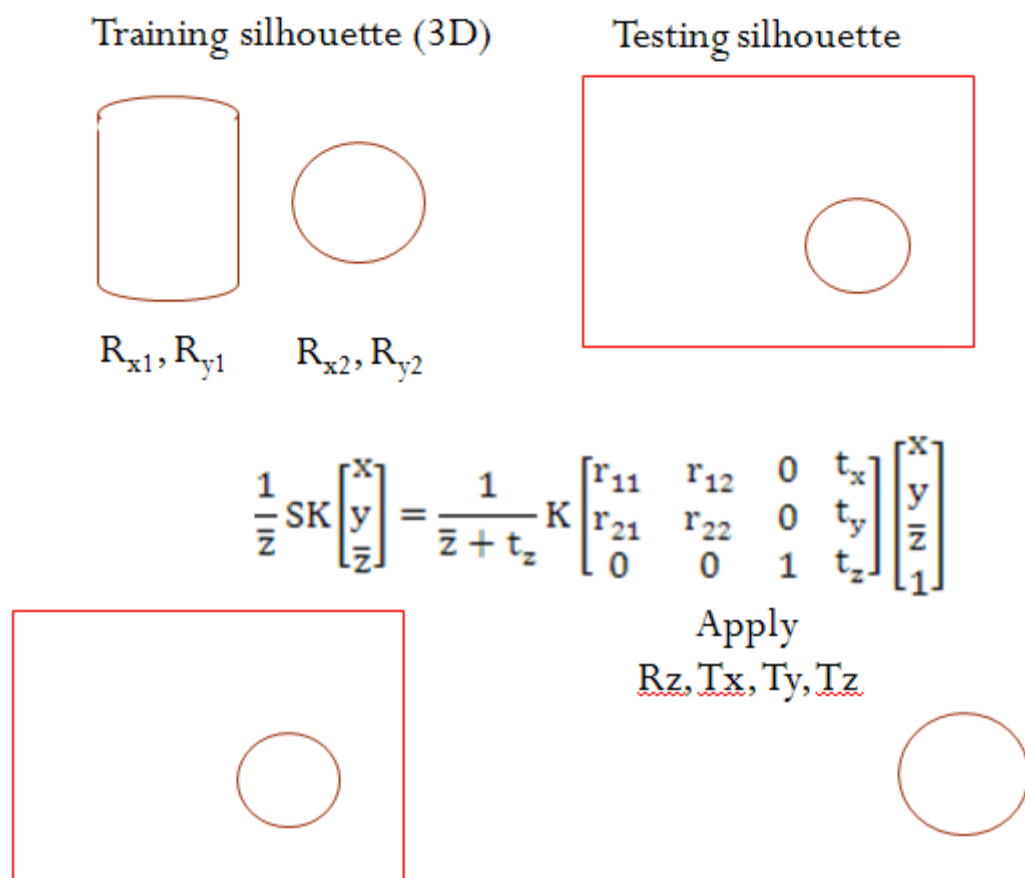
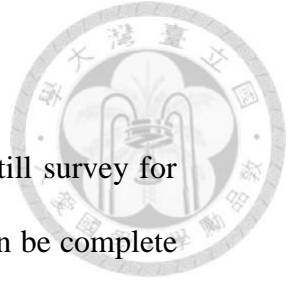


Fig 4.9 An illustration of pose estimation.



## Chapter 5 Grasping

Although we have not implemented the grasping function yet, we still survey for the grasping function so that the whole pipeline proposed in this thesis can be complete for a service robot to manipulate transparent objects. Since the estimated pose of transparent objects is in RGBD sensor's coordinate, so we need to transform the pose to gripper's coordinate frame to let robot grasp. This can be done by ROS tf introduced in section 5.1. Also, there is a manipulation pipeline in ROS that can be used for manipulating objects, which will be covered in section 5.2.

### 5.1 ROS tf for Coordinate Transform

A robotic system typically has many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, head frame, etc. (See Fig 5.1) tf keeps track of all these frames over time, and it can operate in a distributed system. This means all the information about the coordinate frames of a robot is available to all ROS components on any computer in the system.

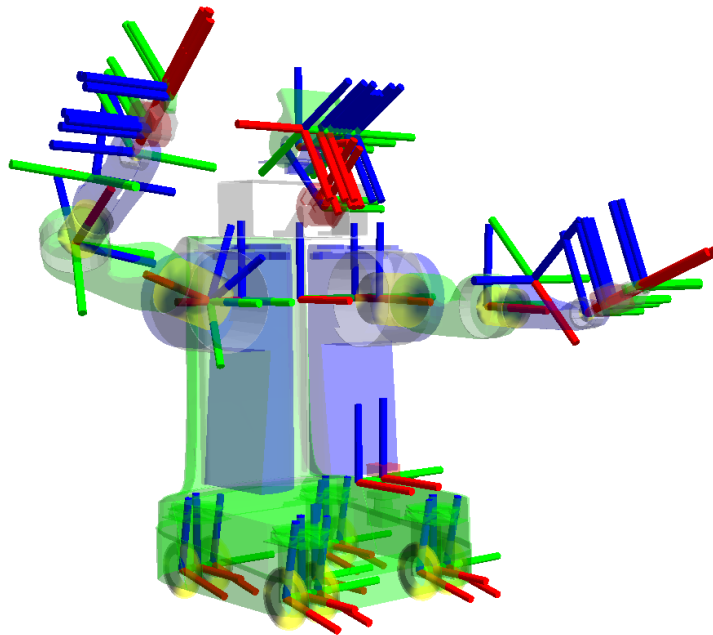


Fig 5.1 PR2's 3D coordinate frames.

Many ROS packages require the transform tree of a robot to be published using the tf software library. At an abstract level, a transform tree defines offsets in terms of both translation and rotation between different coordinate frames. To make this more concrete, consider the example of a simple robot that has a mobile base with a single laser mounted on top of it, as shown in Fig 5.2. In referring to the robot let's define two coordinate frames: one corresponding to the center point of the base of the robot and one for the center point of the laser that is mounted on top of the base. Let's also give them names for easy reference. We'll call the coordinate frame attached to the mobile base "base\_link" (for navigation, its important that this be placed at the rotational center of the robot) and we'll call the coordinate frame attached to the laser "base\_laser."

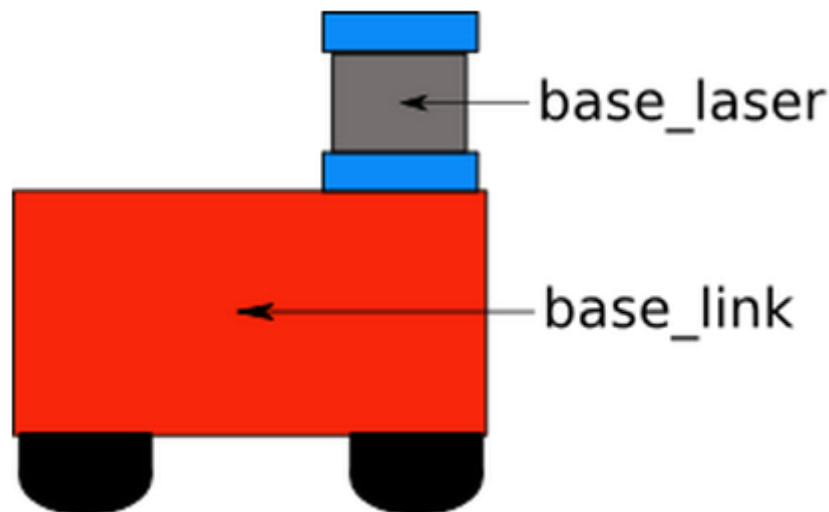
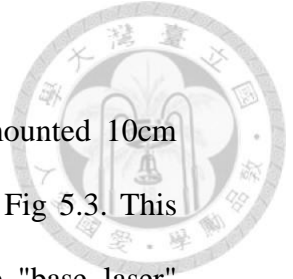


Fig 5.2 An example of a simple robot.

At this point, let's assume that we have some data from the laser in the form of distances from the laser's center point. In other words, we have some data in the "base\_laser" coordinate frame. Now suppose we want to take this data and use it to help the mobile base avoid obstacles in the world. To do this successfully, we need a way of transforming the laser scan we've received from the "base\_laser" frame to the "base\_link" frame. In essence, we need to define a relationship between the "base\_laser"



and "base\_link" coordinate frames.

In defining this relationship, assume we know that the laser is mounted 10cm forward and 20cm above the center point of the mobile base shown in Fig 5.3. This gives us a translational offset that relates the "base\_link" frame to the "base\_laser" frame. Specifically, we know that to get data from the "base\_link" frame to the "base\_laser" frame we must apply a translation of (x: 0.1m, y: 0.0m, z: 0.2m), and to get data from the "base\_laser" frame to the "base\_link" frame we must apply the opposite translation (x: -0.1m, y: 0.0m, z: -0.20m).

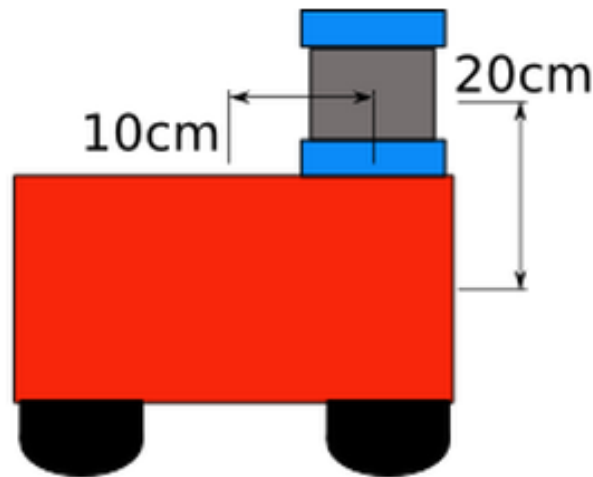


Fig 5.3 The relationship between laser and base of the simple robot.

We could choose to manage this relationship ourselves, meaning storing and applying the appropriate translations between the frames when necessary, but this becomes a real pain as the number of coordinate frames increase. Luckily, however, we don't have to do this work ourselves. Instead we'll define the relationship between "base\_link" and "base\_laser" once using tf and let it manage the transformation between the two coordinate frames for us.

To define and store the relationship between the "base\_link" and "base\_laser" frames using tf, we need to add them to a transform tree. Conceptually, each node in the

transform tree corresponds to a coordinate frame and each edge corresponds to the transform that needs to be applied to move from the current node to its child. Tf uses a tree structure to guarantee that there is only a single traversal that links any two coordinate frames together, and assumes that all edges in the tree are directed from parent to child nodes.

To create a transform tree for our simple example, we'll create two nodes, one for the "base\_link" coordinate frame and one for the "base\_laser" coordinate frame. To create the edge between them, we first need to decide which node will be the parent and which will be the child. Remember, this distinction is important because tf assumes that all transforms move from parent to child. Let's choose the "base\_link" coordinate frame as the parent because as other pieces/sensors are added to the robot, it will make the most sense for them to relate to the "base\_laser" frame by traversing through the "base\_link" frame. This means the transform associated with the edge connecting "base\_link" and "base\_laser" should be  $(x: 0.1m, y: 0.0m, z: 0.2m)$ . With this transform tree set up, converting the laser scan received in the "base\_laser" frame to the "base\_link" frame is as simple as making a call to the tf library. Our robot can use this information to reason about laser scans in the "base\_link" frame and safely plan around obstacles in its environment.

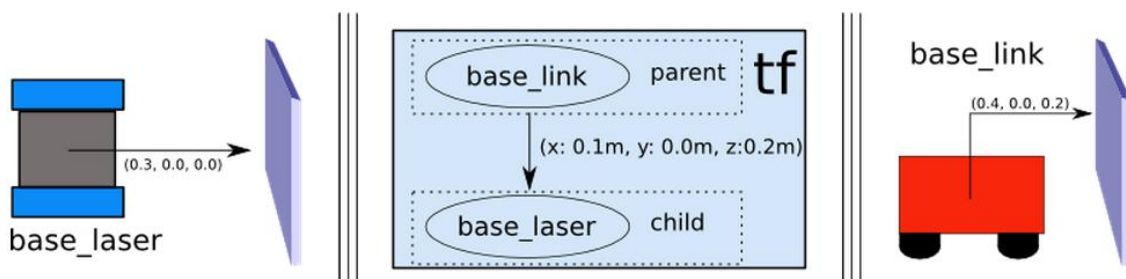
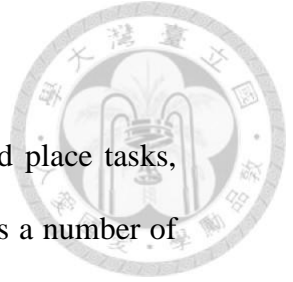


Fig 5.4 The tf tree for the simple robot.



## 5.2 PR2 manipulation pipeline

Object manipulator [35] provides the core-functionality for pick and place tasks, implemented in a robot-independent way. The object manipulator assumes a number of actions and services are available for it to call. For the PR2 robot equipped with a gripper, default implementations are available for all of these.

Chronologically, the process of grasping an object goes through the following stages:

- the target object is identified in sensor data from the environment
- a set of possible grasp points are generated for that object
- a collision map of the environment is built based on sensor data
- a feasible grasp point (no collisions with the environment) is selected from the list
- a collision-free path is generated and executed, taking the arm from its current configuration to a pre-grasp position for the desired grasp point
- the final path from pre-grasp to grasp is executed
- the gripper is closed on the object and tactile sensors are used to detect presence or absence of the object in the gripper
- the object is lifted from the table

We can identify the transparent objects by the algorithms stated in the previous sections. And the detailed description on other modules can be found in [35]. So we just present an example from [35] in Fig 5.5 to Fig 5.8.



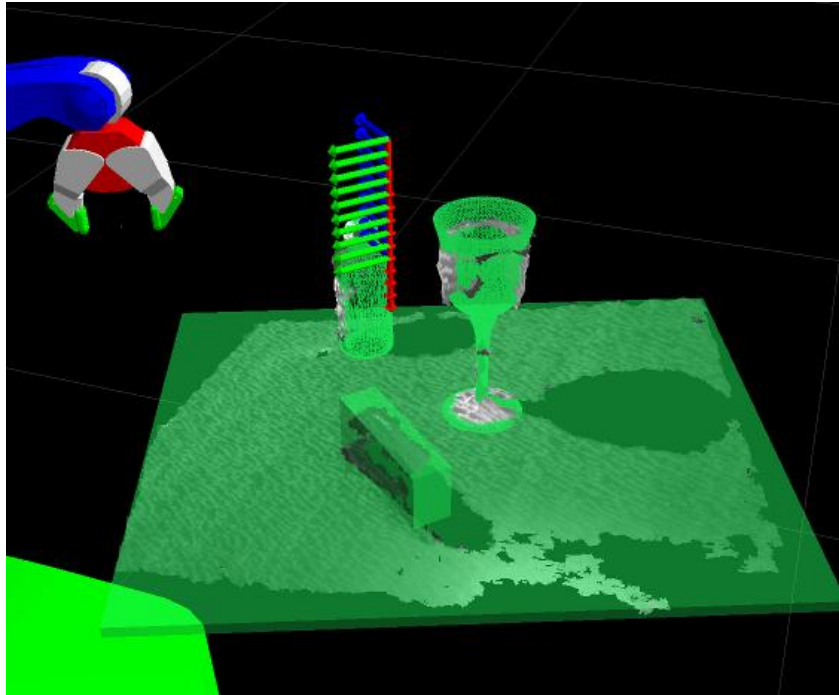


Fig 5.5 Interpolated IK path from pre-grasp to grasp planned for a grasp point of an unknown object.

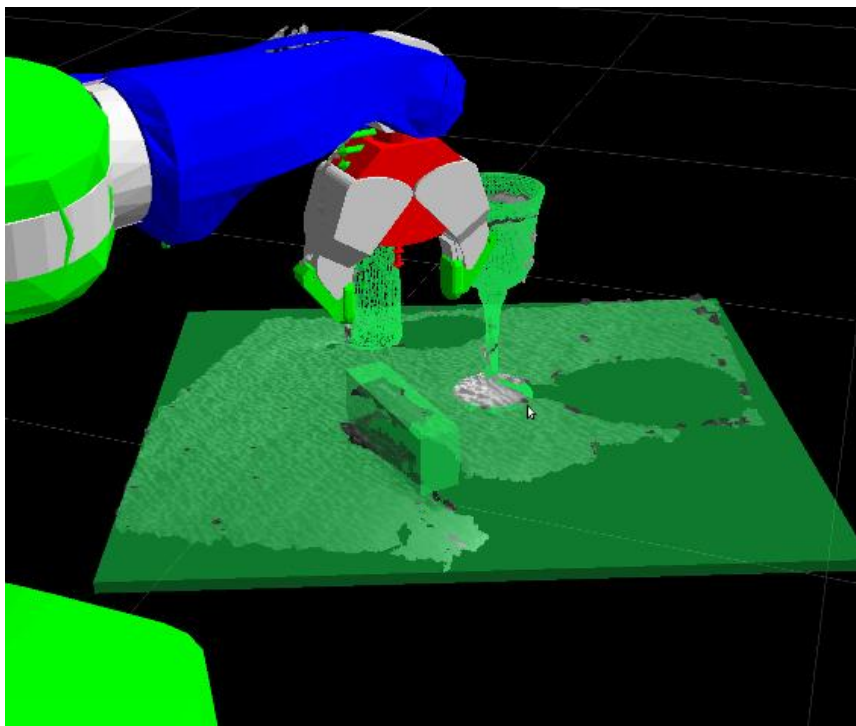


Fig 5.6 A path to get the arm to the pre-grasp position has been planned using the motion planner and executed.

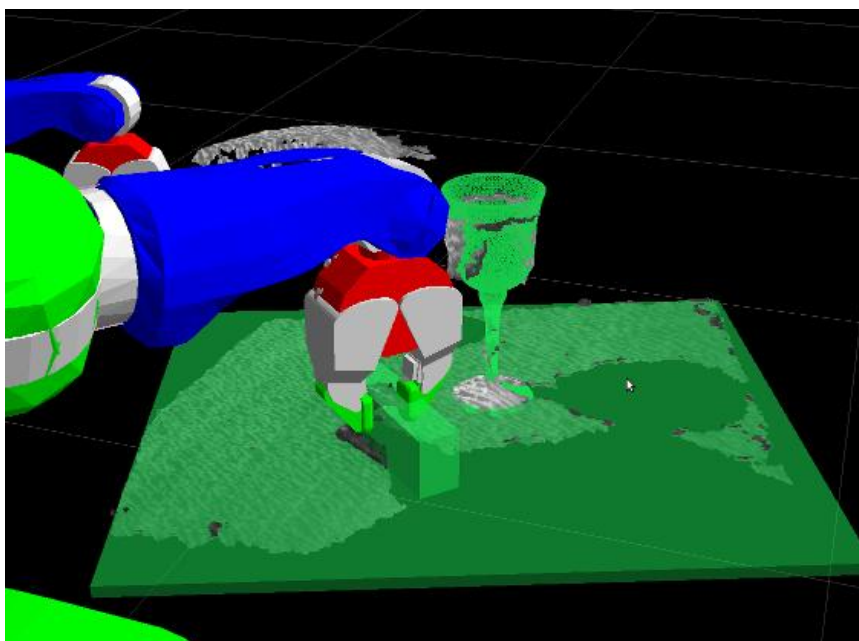


Fig 5.7 The interpolated IK path from pre-grasp to grasp has been executed.

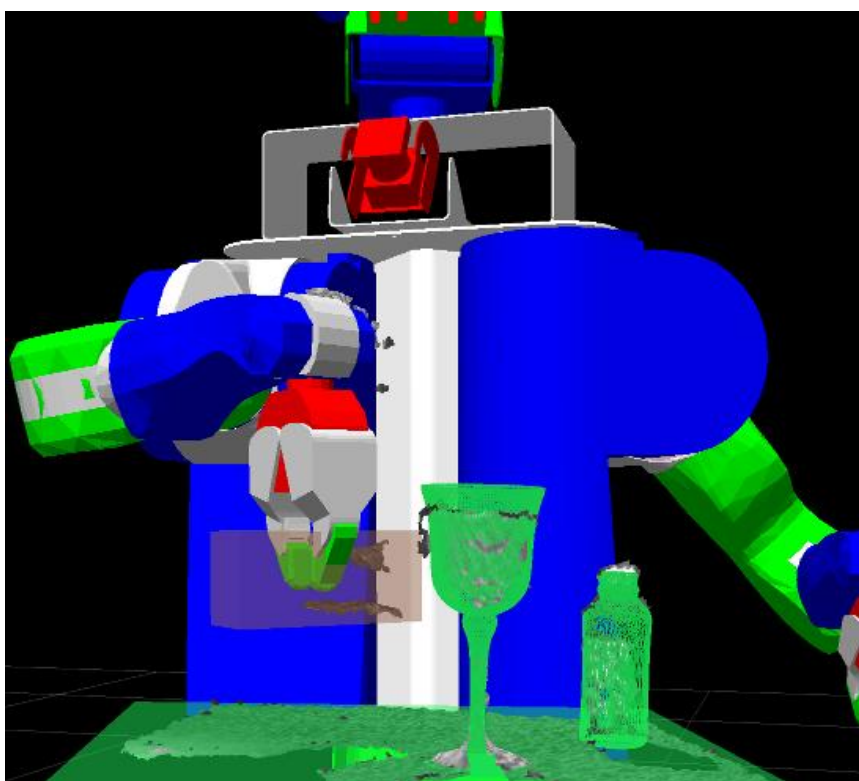


Fig 5.8 The object has been lifted.

## Chapter 6 Experiment



### 6.1 Experiment Setup

As shown in Fig.6.1, we put the transparent objects on the table in front of PR2 for recognition and pose estimation.



Fig. 6.1 PR2 robot manipulates transparent object.

### 6.2 Detection of Transparent Objects

In this section, we present the experiment on transparent object detection

#### 6.2.1 Grabcut-based Method

Here we test the performance of the transparent candidate recognition, and compare the result with the method proposed in [6]. The inputs of this test are RGB and depth images, and output is the marked regions that are considered as transparent

objects. In this test, we use recall and precision to evaluate the performance. Recall is the ratio of the number of correctly retrieved transparent object over the number of all transparent objects should be retrieved. Precision is the ratio of the number of correctly retrieved transparent object over the number of all retrieved. So in the ideal case, one can expect that the output regions are always transparent objects (100% precision) and all transparent objects in the scene are all retrieved (100% recall).

We randomly put 5 transparent objects as shown in Fig.6.2 (2 different kinds of glass goblets, beaker, graduate cylinder, test tube) and other non-transparent objects that will cause unknown depth value in the scene. Then we test if our transparent candidate retrieving module can correctly retrieve the transparent ones. In Fig.6.3, the green silhouette means the algorithm think that region is transparent. If the green silhouette contains the region that is a non-transparent object, we view it as error.



Fig. 6.2 Five transparent objects used to test the performance of recognition.

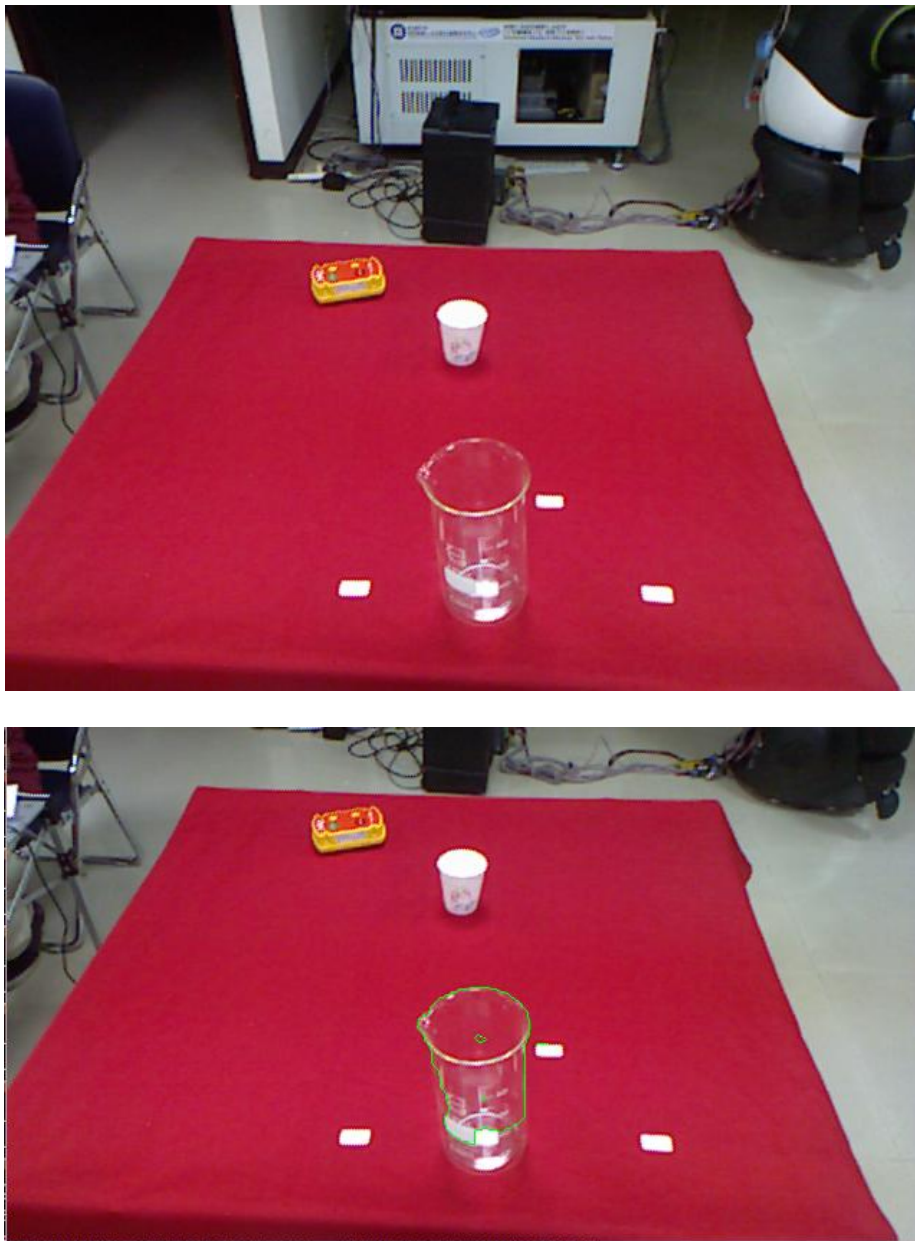


Fig. 6.3 The original image and the correct recognition result.

In the test, the total retrieved candidates are over 200. The result is shown in table 6.1. As can be seen in table 6.1, recalls of both methods are similar, which means if there are transparent objects in the scene, both method can retrieved them in most cases. However, our method outperforms the previous method on precision since our method better rules out non-transparent regions which have unknown value in the depth image. Since the method in [6] is not focusing on tackling this problem, so the result can be



expected.



Table 6.1 Recall and Precision of Grabcut-based Transparent Object Recognition

| <i>Method</i> | <i>Recall</i> | <i>Precision</i> |
|---------------|---------------|------------------|
| Method in [6] | 86.11%        | 38.24%           |
| our method    | 86.11%        | 93.93%           |

## 6.2.2 Deep Learning Based Method

To test if our algorithm can detect transparent objects in color image, we use the test dataset used in [3], which contains 14 images. These images are taken in a normal house scenario, with different lighting conditions and occlusions. Three of them are shown in Fig 6.4. Note that this experiment is not related to the PR2 and the experiment setup stated in section 6.1.





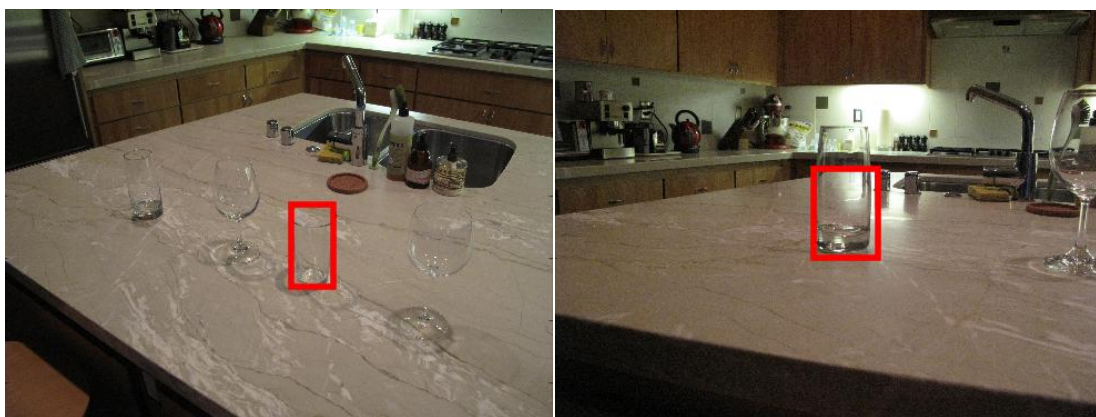
Fig. 6.4 Some of the images in test dataset.

As can be seen in Fig. 6.5, the red rectangle contains a transparent object and the label is beaker. Although there is a blue rectangle recognized, the label is axe, so it is not related to transparent object. Note that R-CNN can actually recognize multiple objects in the scene instead of recognize transparent object only.

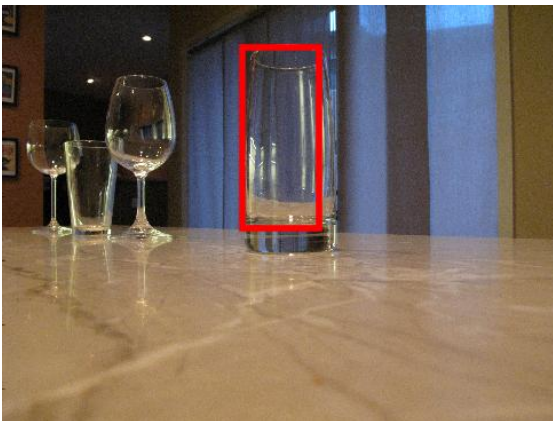
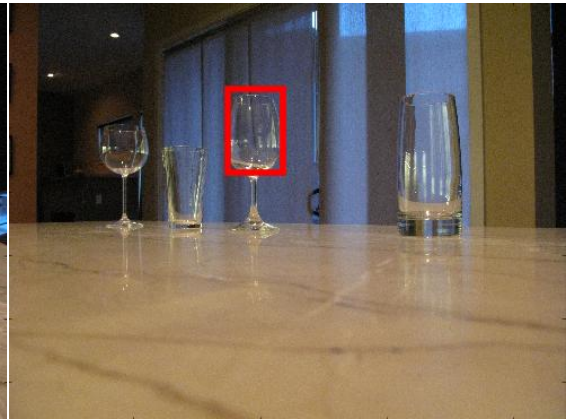


Fig. 6.5 A recognition result from R-CNN.

The results are shown in Fig. 6.6. In Fig. 6.6, we only show rectangles contains beaker. As can be seen, in most of the cases, transparent object can be detected in red rectangle. The result shows that R-CNN can be used to recognize transparent objects in color image.







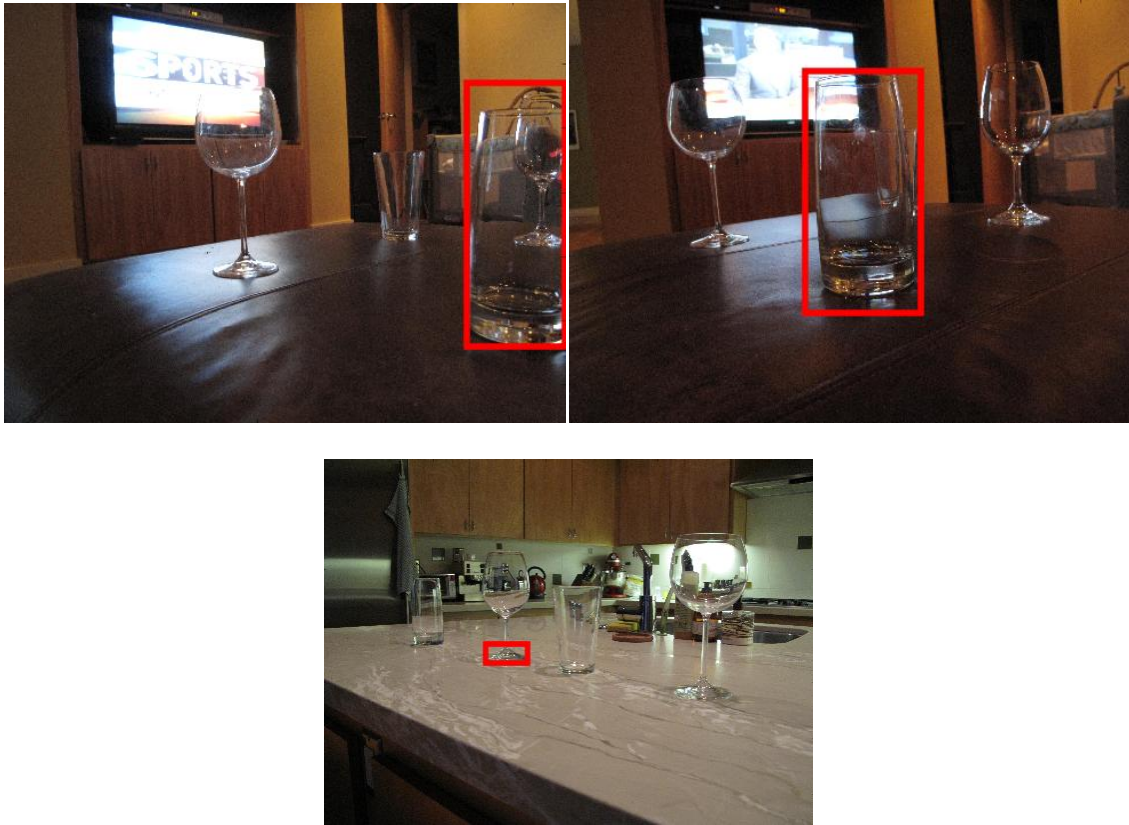


Fig. 6.6 Testing result of R-CNN method.

### 6.3 Pose Estimation of transparent Objects

We evaluate the pose estimation module on transparent equipment. Since we want to apply pose estimation in a real bio-laboratory application, so it is essential to test the work space of pose estimation module. There is no need to further evaluate the accuracy due to the evaluation is already done in [6]. In this test, we use three transparent objects, beaker, graduate cylinder and test tube to verify. Since we predefined the grasp pose of these objects, so we evaluate the correctness of pose estimation by checking if PR2 can correctly grasp the recognized objects. One of the correct results is shown in Fig 6.7.



Fig. 6.7 One of the results of pose estimation.

The result is shown in table 6.2. We found that if the object is placed closely to the robot, the success rate would be very high. But if object is placed a little bit far from the robot, the success rate would drop quickly. The main reason is that Kinect is mounted on PR2's head, so the distance from Kinect to object is longer than the distance from robot to object. Once the distance exceeds the limitation of Kinect, there'll be lots of unknown value on the depth image, thus interfere the process of transparent candidate retrieval. Though the limitation of workspace is an issue to be further investigated, in a normal bio-laboratory application, the transparent equipment wouldn't be placed far from the manipulator so that he or she can manipulate equipment without moving.

Table 6.2 Success Rate of Pose Estimation

| <i>Distance</i>                 | <i>Success rate</i> |
|---------------------------------|---------------------|
| $D < 0.5\text{m}$               | 87.32%              |
| $0.5\text{m} < D < 0.9\text{m}$ | 28.65%              |
| $0.9\text{m} < D$               | 0%                  |

## Chapter 7 Conclusion and Future Works



### Conclusions

In conclusion, we have investigated the algorithms for recognizing and estimating the pose of transparent objects in the scene. We improved the key functions of recognition of transparent equipment to deal with non-transparent objects that cause unknown value in Kinect depth image. These objects are very possible to be falsely recognized as transparent by previous method, but our method can deal with this problem properly.

Apart from the recognition and pose estimation, we also discuss the manipulation of a service robot to grasp transparent objects, making the whole thesis more complete.

### Future Work

Future work includes integrating more function module such as navigation and grasping. Also, grasping point estimation for fragile transparent objects is an important topic to be further investigated.

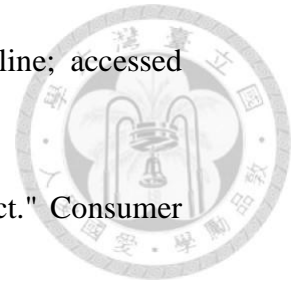


## REFERENCE

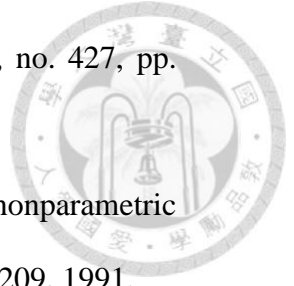


- [1] M. Osadchy, D. Jacobs, and R. Ramamoorthi, "Using specularities for recognition," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 2003, pp. 1512-1519.
- [2] K. McHenry, J. Ponce, and D. Forsyth, "Finding glass," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005, pp. 973-979.
- [3] M. Fritz, G. Bradski, S. Karayev, T. Darrell, and M. J. Black, "An additive latent feature model for transparent object recognition," in *Advances in Neural Information Processing Systems*, 2009, pp. 558-566.
- [4] C. J. Phillips, K. G. Derpanis, and K. Daniilidis, "A novel stereoscopic cue for figure-ground segregation of semi-transparent objects," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, 2011, pp. 1100-1107.
- [5] I. Lysenkov, V. Eruhimov, and G. Bradski, "Recognition and pose estimation of rigid transparent objects with a kinect sensor," *Robotics*, p. 273, 2013.
- [6] I. Lysenkov and V. Rabaud, "Pose estimation of rigid transparent objects in transparent clutter," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 162-169.
- [7] ROS.org, <http://www.ros.org/> [Online; accessed 30-July-2015].
- [8] Understanding ROS Topics, <http://wiki.ros.org/action/fullsearch/ROS/Tutorials/UnderstandingTopics> [Online; accessed 30-July-2015].
- [9] Understanding ROS Services and Parameters,

<http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams>[Online; accessed 30-July-2015].



- [10] Smisek, Jan, Michal Jancosek, and Tomas Pajdla. "3D with Kinect." *Consumer Depth Cameras for Computer Vision*. Springer London, 2013. 3-25.
- [11] Meshlab, <http://meshlab.sourceforge.net/> [Online; accessed 30-July-2015].
- [12] B. Tuong-Phong, "Illumination for computer-generated images," University of Utah, pp. 29-51, 1973.
- [13] Rother, Carsten, Vladimir Kolmogorov, and Andrew Blake. "Grabcut: Interactive foreground extraction using iterated graph cuts." *ACM Transactions on Graphics (TOG)* 23.3 (2004): 309-314.
- [14] Boykov, Yuri, Olga Veksler, and Ramin Zabih. "Fast approximate energy minimization via graph cuts." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 23.11 (2001): 1222-1239.
- [15] Goldberg, Andrew V., and Robert E. Tarjan. "A new approach to the maximum-flow problem." *Journal of the ACM (JACM)* 35.4 (1988): 921-940.
- [16] Y. Raja, S. McKenna, and S. Gong, "Segmentation and tracking using color mixture models," in *Asian Conference on Computer Vision*, Hong Kong, January 1998.
- [17] S. McKenna, Y. Raja, and S. Gong, "Object tracking using adaptive color mixture models," in *Advances in Color Machine Vision, ACCV Spec. Sess.*, Hong Kong, January 1998.
- [18] C. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.
- [19] G. J. McLachlan and K. E. Basford, *Mixture Models: Inference and Applications to Clustering*, Marcel Dekker Inc., New York, 1988.



- [20] C. E. Priebe, "Adaptive mixtures," *J. Amer. Stat. Assoc.*, vol. 89, no. 427, pp. 796-806, 1994.
- [21] C. E. Priebe and D. J. Marchette, "Adaptive mixtures: Recursive nonparametric pattern recognition," *Pattern Recognition*, vol. 24, no. 12, pp. 1197-1209, 1991.
- [22] C. E. Priebe and D. J. Marchette, "Adaptive mixture density estimation," *Pattern Recognition*, vol. 26, no. 5, pp. 771-785, 1993.
- [23] D. M. Titterton, A. F. M. Smith, and U. E. Makov, *Statistical Analysis of Finite Mixture Distributions*, John Wiley, New York, 1985.
- [24] G. Borgefors, "Hierarchical chamfer matching: A parametric edge matching algorithm," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 10, pp. 849-865, 1988.
- [25] I. L. Dryden and K. V. Mardia, *Statistical shape analysis vol. 4*: Wiley Chichester, 1998.
- [26] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Robotics-DL tentative*, 1992, pp. 586-606.
- [27] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*: Cambridge university press, 2003.
- [28] A. W. Fitzgibbon, "Robust registration of 2D and 3D point sets," *Image and Vision Computing*, vol. 21, pp. 1145-1153, 2003.
- [29] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." *Computer Vision and Pattern Recognition (CVPR)*, 2014 IEEE Conference on. IEEE, 2014.
- [30] C. Gu, J. J. Lim, P. Arbelaez, and J. Malik. Recognition using regions. In *CVPR*, 2009.
- [31] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for

object recognition. IJCV, 2013.

- [32] J. Carreira and C. Sminchisescu. CPMC: Automatic object segmentation using constrained parametric min-cuts. TPAMI, 2012.
- [33] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. <http://caffe.berkeleyvision.org/>, 2013.
- [34] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.
- [35] ROS Object manipulator, [http://wiki.ros.org/object\\_manipulator](http://wiki.ros.org/object_manipulator) [Online; accessed 30-July-2015].





# Curriculum Vitae



姓名：賴柏任

學歷：

1. 民國 104 年 國立台灣大學電機工程學研究所畢業
2. 民國 102 年 國立台灣大學電機工程學系畢業
3. 民國 98 年 國立新竹科學工業園區實驗高級中學畢業

發表著作：

1. Luo, Ren C., Po-Jen Lai, and Vincent Ee Wei Sen. "Transparent Object Recognition and Retrieval for Robotic Bio-Laboratory Automation Applications." Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE, 2015. (accepted)
2. Luo, Ren C., Wai Un Chan, and Po-Jen Lai. "Intelligent robot photographer: Help people taking pictures using their own camera." System Integration (SII), 2014 IEEE/SICE International Symposium on. IEEE, 2014.

榮譽事蹟：

- 民國 104 年 參加「2015 智慧型機器人產品創意競賽」獲 國產機器人組冠軍  
民國 103 年 參加「2014 智慧型機器人產品創意競賽」獲 國產機器人組冠軍