A FAST AND ACCURATE ROBOTIC GRASP METHOD USING DEEP LEARNING

A Thesis

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

YUEQI YU

Marjorie Skubic, Thesis Supervisor

DECEMBER 2016

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

A FAST AND ACCURATE ROBOTIC GRASP METHOD USING DEEP LEARNING

presented by Yueqi Yu,

a candidate for the degree of Master of Science

and hereby certify that, in their opinion, it is worthy of acceptance.

MARJORIE SKUBIC

ZHIHAI HE

YUYI LIN

DEDICATION

Words cannot express the depth of gratitude that I have toward my family. Without their support, this journey would not have been possible. They gave me the ability to realize my dreams and express my thoughts.

ACKNOWLEDGMENTS

I would like to express my appreciation to the persons who helped me with this project. I would like to express my sincere gratitude to my adviser, Dr. Skubic, for offering me such a fantastic topic, which enabled me to work on a robotic project—an area of research that I have always been enthusiastic about. I am extremely grateful for her assistance and suggestions throughout my project and thesis. I would also like to thank my committee, Dr. He and Dr. Lin.

I would like to thank Erik Huo, who has guided me into a new world of robotics and has helped me with a lot of technical problems. I would also like to give special thanks to Xingfang Yuan who inspired me to use the deep learning method in my project and Zhi Zhang who helped me to better understand the method I used.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS AND TERMINOLOGIES

RGB-D data: Data acquired by Microsoft Kinect that contains both RGB image and depth image.

Depth Image: Images where pixel value at each position represents the distance of the corresponding real-world position to the camera.

Registered Depth Image: Depth image that contains (x, y, z) coordinates for corresponding points with respect to the camera in real world using the same coordinate system as an RGB image.

DOF: Degrees of freedom in this thesis usually used to describe the robotic arm.

GPU: Graphic Processing Unit is used as a processor to handle the deep neural networks in our system which can be faster than using CPU.

ROS Topics: We can consider this as a control and message access socket for specific components used in the system. For example, the topic named "/Joint3_Controller" represents the control socket for joint3. Sending a degree to this socket directs the corresponding joint to move accordingly.

Subscribe in ROS: Keep receiving messages from a specific ROS topic.

Publish in ROS: Send message to an ROS topic.

Plugin in Gazebo: Gazebo is a software program that enables Gazebo model to communicate with ROS master server and communicate with ROS topics.

Stall Torque: This is measured by N/m. To a servo, if it has a stall torque of 1N/m, which means it can generate maximum force of 1N at the far end of the bar if the bar is 1 m long and connected to the center of the servo and rotated by the servo.

Feature Map: Pictures we get from convolution.

Max Pool: Replace every n*n pixel area with a max pixel value within this area. This will shrink the image but reduces the later convolution cost for the system. For example, after performing a 2*2 max pooling on a 4*4 image, a 3*3 image will be available after max pooling.

Stride: When using a max pooling or convolution kernel, the pixel that the kernel moves between every operation is called stride. For example, when we perform 2*2 max pooling with a stride of 2 on a 4*4 image, then we will get a 2*2 image after max pooling.

Convolution Kernel: An N dimensional matrix containing weights for different elements in it that will be used in convolution with an N dimensional image.

Anchor: Considering every pixel on a feature map after the 5^{th} convolution layer as a center. We propose nine different bounding boxes by using three scales and three aspect ratios.

A FAST AND ACCURATE ROBOTIC GRASP METHOD USING DEEP LEARNING

Yueqi Yu

Dr. Marjorie Skubic, Thesis Supervisor

ABSTRACT

I have developed a system that is capable of quick and accurate detection of target objects using the deep learning method. Adding orientation with depth sensing then allows a robot to grasp the objects. The existing systems, which use the traditional deep learning method on RGB-D images for object detection and grasping point estimation and use more expensive and professional robotic platforms to grasp, are more time-consuming and less accurate in estimating the grasping point. To reduce the time spent for object and grasping point detection. I first adapted the Faster Region Convolutional Neural Network (Faster RCNN) method into my system, which uses only RGB images to do object detection and achieved a high speed. Using the limited area given by the RGB object detection results, along with a registered depth image provided by a Microsoft Kinect, the system achieved a high-speed success rate for robotic estimation of grasping points for each object. Then, using robotic kinematics, the system can quickly get the desired angle for each joint of the four degrees of freedom robotic arm. The arm can then pick up the object accurately following an appropriate path planning strategy for each object class. The system reduced the object detection and grasping point estimation time from over one second to an average of around 0.135 s. It also reduced the average absolute error rate in estimated grasping point to 1.59 mm instead of 18 mm although it achieved a slightly lower grasping and pickup performance at 87% compared to 90% achieved by the traditional method.

CHAPTER 1 INTRODUCTION

My graduate research here is developed in the context of eldercare technology, where the goal of the research is to serve older adults. In particular, this research is motivated by a fetch task being performed by an assistive robot, where various objects to be fetched are located on tabletop or desktop surfaces. Thus, the research methods presented here are targeting this type of object detection, grasp, and pickup that can be performed quickly and accurately for use by an interactive robot.

The fascination with robots and robotic grasping started becoming realistic in the 1940's, but the first patent on a robotic arm was filed by Goertz and Uecker in 1951 for the Atomic Energy Commission [41]. This first "tele-operated articulated arm" was a landmark in the development of force feedback technology, otherwise known as haptic technology [1]. Today, robotic technology has been widely implemented in many fields. In recent years, with the rapid development of deep learning technologies, many researchers have researched the possibility of combining robotic grasping with deep learning. The core problem of robotic grasping can be summarized as: Given a scene acquired by the robot's "eyes," how to locate the object we want to pick up in the scene and use the right grasping strategy to pick it up is shown in Figure 1-1. The problem can be divided into two parts: recognition and grasp. Traditional methods usually use hand-designed features to do the task. But these methods for recognizing and grasping are quite time-consuming and lack accuracy such as [1] and [2].

Figure 1-1 An example for object detection and grasping point estimation

The most time-consuming part in the task has been in recognition. It takes more time to get good recognition results. The grasping speed is mainly related to the robotic arm itself and the grasping method. Several researchers have developed algorithms using a deep learning method to find the objects' location and grasping point [3]–[6]. However, they mainly attempted to increase the recognition accuracy and range, but only made little progress on the recognition speed. Faster Region Convolution neural network (Faster R-CNN) has been proved to be a fast and accurate 2D-image detection technology [7], and the Microsoft Kinect has been one of the best 3D sensors. In my work, I will present a new method which uses data acquired from a Microsoft Kinect and processed by Faster R-CNN in 2D space and is able to locate the object in 3D space. As opposed to previous works [5], which implemented RGB-D data that is four dimensional and has a higher computational cost, our approach only used RGB images and a graphics processing unit (GPU) for computation at the object detection stage, which greatly reduced the computational time cost.

I also developed a method that exploits depth data to find the grasping point of objects. Here, we designed an algorithm for each object class to find the best grasping point on the object after it was recognized.

I developed the whole system based on two computers. Both ran an ROS (robotic operating system) to control a robotic arm that could grab required objects. In this project, ROS is the main system for all the information transfers and message transfers since it supports many of the existing programming languages and allows programs in different languages to work together. Another advantage of ROS is that it is a distributed system which means it can handle multiple tasks on multiple computers. This feature greatly reduced the work for the computer of a single robot and increased the speed of handling tasks. The robot also gained more mobility since it did not need a powerful core to do the deep learning detection work.

So, to summarize the basic steps: This system first obtains both RGB and registered depth information of current scenes from a Microsoft Kinect. Then, an object detection system (based on a Faster R-CNN) is run on the RGB image to find the target object and return the coordinate of the object center in the image coordinate. Then, using the registered depth image we can find the corresponding 3D coordinates of the object center with respect to the RGB camera coordinate system of the Kinect. Next, using a fixed coordinate transfer matrix, we obtain the coordinate of the object with respect to the base of the robotic arm. Using robot kinematics, we obtain the angle parameter for each arm joint and allow selection of the object after estimating the grasping point and deciding the best path. I tested my detection network on a mixture of some images from ImageNet, and labeled some

pictures and tested entire system with a 4 DOF robotic arm and 5 target objects. The main contributions of this thesis can be summarized as:

1. A new method is introduced to implement the Faster R-CNN method in finding an objects' 3D location using the Microsoft Kinect.
2. The new system greatly reduces the time used for recognizing the right grasping point of an object.
3. The system was developed and tested using a 4 DOF robotic arm and has shown a high grasping performance.

The rest of this thesis is organized as follows: I first introduce some related works and basic information about the equipment and technologies used in the work in Chapter II. Chapter III presents the details of how I built the system and implemented the new algorithms. The experiment results and discussion are in Chapter IV. The summary of findings and conclusions are in Chapter V.

CHAPTER 2 BACKGROUND

2.1 Related Work

2.1.1 Object Detection

Object Detection has been a popular topic in the machine learning field. Researchers have tried multiple ways to build a system that could detect multiple target objects from, for example, an input image and locate their areas in the image.

In the last decade, SIFT [26] and HOG [27] with blockwise orientation histograms features have been widely used over various visual based object recognition researches. These methods have been tested in visual recognition tasks such as PASCAL VOC object detection [29, 30], most of them were implemented on RGB images and the progress has been slow during past years.

These methods mentioned above focus mostly on single-view-based algorithms, which takes only one image as input each time and give corresponding detection results. However, some researchers have developed multi-view-based algorithms that input several scenes from different view point and give the detection result. This is quite practical when used on robotics. One implementation is to combine object detection with the Simultaneous Localization and Mapping (SLAM) algorithm [43]. There are several approaches proposed using object-based SLAM such as [44-48]. Some methods also combine HOG with multi-view detection using RGB-D data, these methods used HOG feature sliding window detectors which was generated by results trained from an RGB-D dataset that contain these objects [49] [50].

Several image labeling projects shared over the Internet [35-37] have recently made Cloud-based object recognition one of the popular methods in the object detection area. Ben Kehoe et al. [25] has developed a system, using the PR2 robot, that integrates Google's image recognition system with a sampling-based grasping algorithm to recognize and grasp objects. The advantages for cloud based object recognition is that it not only makes use of a huge library of annotated images[38] but can also handle massive parallel computation tasks[39] which is common in the object detection field. However, this requires an expensive robotic platform such as the PR2 and a good continuous internet connection that can ensure such a high data exchange speed.

Convolution Neural Networks (CNNs) were once a popular method in the 1990s [40], but due to the limitation of computer development, the research focus soon turned to the rapidly developed support vector machines (SVM). However, in 2012, work done by Krizhevsky et al. [31] which achieved high classification accuracy on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [32, 33] brought back interests in using CNNs to do classification or detection works.

In recent years, many researchers have attempted to solve the detection problems using the deep learning method such as CNNs in solving object detection problems for robots [4]–[6]. However, with all the recent advances in this area, region-based proposal methods have become one of the most popular ways to do object detection [8]. Furthermore, this method was introduced to work with convolutional neural networks (CNNs) as Region proposals with CNNs (R-CNNs) [9]. The R-CNN method was the first to show that a CNN-based method can achieve dramatically higher object detection accuracy on PASCAL VOC compared to systems based on HOG-like features [9]. However, R-CNNs are still a quite

time-consuming and computationally expensive method even though they take advantage of sharing convolutions across region proposals which already have greatly reduced costs. Much research has been conducted based on region methods, and many have achieved some speed and accuracy improvement, such as [10], but they still cannot meet the detection speed requirements for an interactive robot. The Fast Region Convolution Neural Network (Fast R-CNN) [11] was introduced to reduce the time and computational cost spent on the detection network, and its use enabled this algorithm to achieve near real-time detection rates if the time spent on region proposals is ignored.

A region proposal-based method usually incorporates features that are computationally economical. So, Selective Search [12] was used to propose possible regions since it focuses on greedily searching for inexpensive features. But it still takes around two seconds per image in a common CPU implementation [11]. So, a more advanced method was introduced as a Faster R-CNN [7], which keeps the Fast R-CNN structure but develops another deep learning network called a region proposal network (RPN). RPNs share the convolution layers with detection networks, which makes time spent on the region proposal as low as 10 ms per image. This makes the whole Faster R-CNN reach a real-time detection speed. I adapted a Faster R-CNN into our project to improve the system performance.

2.1.2 Robotic Grasping

Much of the previous work in robotic grasping tends to train the whole system with full 2D or 3D information of specific shaped objects and gives grasping points where needed. The force and form-closure is one of those examples [13].

It is quite common for people to train the whole system with both images of objects and the ground truth grasping point for each object in the scene [3]; some would even train with several ground truth poses of the robotic arm [33]. This is time-consuming and inefficient in preparing the training dataset. For the system to achieve a high accuracy and be able to handle novel objects, a large, comprehensive dataset is required.

Recent works have taken advantage of the rapid development of research in point clouds and trained the system using 3D models made in AutoCAD [6]. Other works focuses on finding grasping points independently in an image without classifying the object such as Saxena's work [6] and Andreas's work [51]. Saxena et al. used a sliding window to determine whether there are good grasping points in the scene based on the features they previously trained over a wide range of feature collections. Later, Lenz et al explored this method using RGB-D data [52]. He also developed a two-cascade deep neural network to detect the grasping point for the robotic arm and claims to have reached an 84% grasping success rate on Baxter and 92% on PR2, which all have robotic arms with 7 degrees of freedom [3]. Fischinger et al. also developed a similar method using "heightmaps" instead of depth images [53] [54]. They claimed to have achieved a 92% grasping success rate over 50 trials using the PR2[54].

Another approach developed by Klingbeil et al. only searches for geometric shapes that are considered as a “good region to grasp” using depth image from a single frame acquired by a 3D sensor [55]. This approach does not need to build 3D models to train the system for detection, but just searches for regions that can match with the inside shape of the gripper.

In contrast, our system only needs a depth image based on the object region generated from the object detection steps. This reduces much of the computational load and increases the speed for detection. Also, our method proposes a specific grasping technique since we aim at detecting and grasping objects on tables specifically for the elderly. We can accurately find a grasping point for each class of objects simply based on a depth image in the bounding box given by our object detection system.

2.2 The Recognizing and Grasping System

I proposed a recognizing and grasping system, using Kinect RGB images and a Faster-R-CNN to detect objects and used the object's depth image to find a corresponding 3D location of the object center and appropriate grasping points. Then using a 4 degrees of freedom (DOF) robotic arm with calculated angle and path, I can direct the robot to pick up the object. The system work flow chart is shown in Figure 2-1.

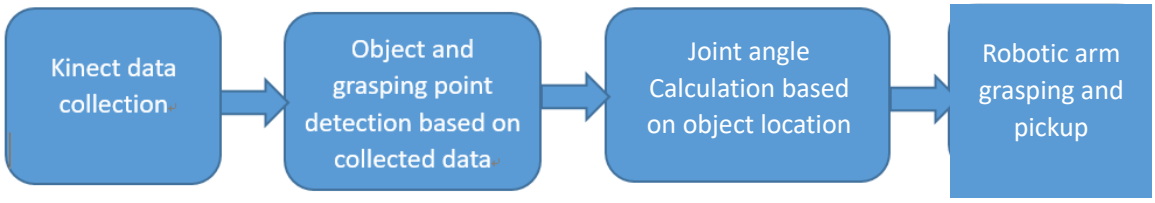


Figure 2-1 Work Flow Chart

The system component organization shown in Figure 2-2 consists of an ROS system as a master system of information transfer and preprocessing, Microsoft Kinect was used to collect data, a desktop computer was the server used to process data (object detection, 3D location calculation, and robotic arm joint angle calculation). A movable laptop was also used as a controller and data collector to control the robotic arm and send Kinect data.

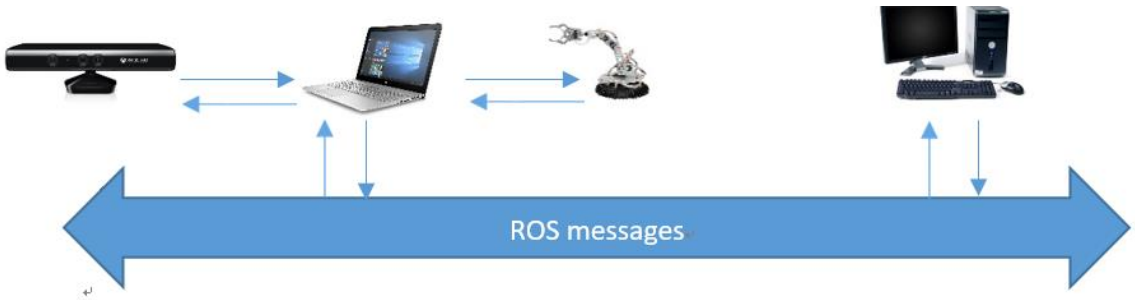


Figure 2-2 System organization

2.3 Microsoft Kinect (Kinect for Windows Version 1)

In February 1, 2012, Microsoft officially released Kinect for Windows, which can both collect depth and RGB data from the scene. Both its depth sensor and video cameras have a 640 ×480-pixel resolution and runs at 30 FPS (frames per second). Figure 2-3 shows the structure and appearance of the Kinect depth sensor and video cameras used.

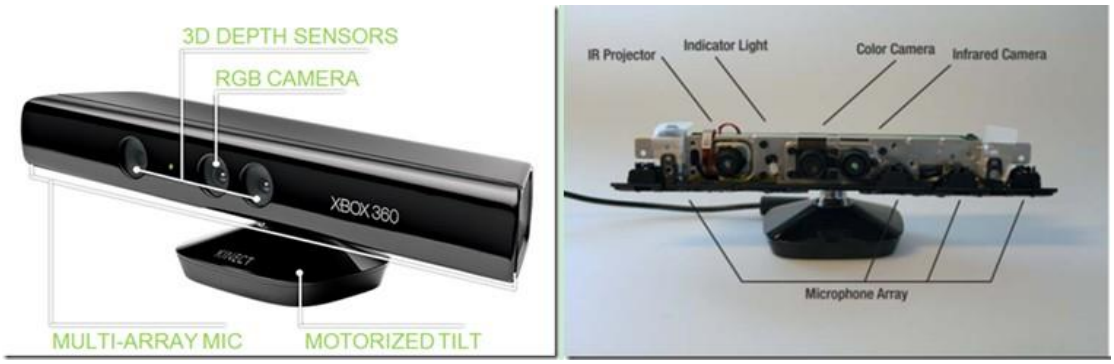


Figure 2-3 Microsoft Kinect appearance and structure: (a) the depth sensors, RGB camera, and multi-array microphone; (b) the cameras, projector, indicator light, and microphone array.

The RGB camera is in the middle, and the component on the left is an infrared radio (IR) projector, the infrared camera on the right collects the reflected infrared radio signal and builds corresponding depth images.

The depth image contains three values at each pixel location. These values represent the x, y, z coordinate in the real world measured in meters.

There are two coordinate systems for the Microsoft Kinect, one for an RGB camera and one for a depth camera. The depth coordinate system is shown in Figure 2-4 below. The origin of the depth image is at the center of the image. However, the RGB image has an origin at the top left corner as shown in Figure 2-5.

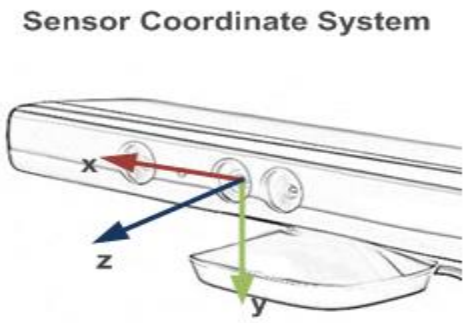


Figure 2-4 Depth Image and its Coordinate

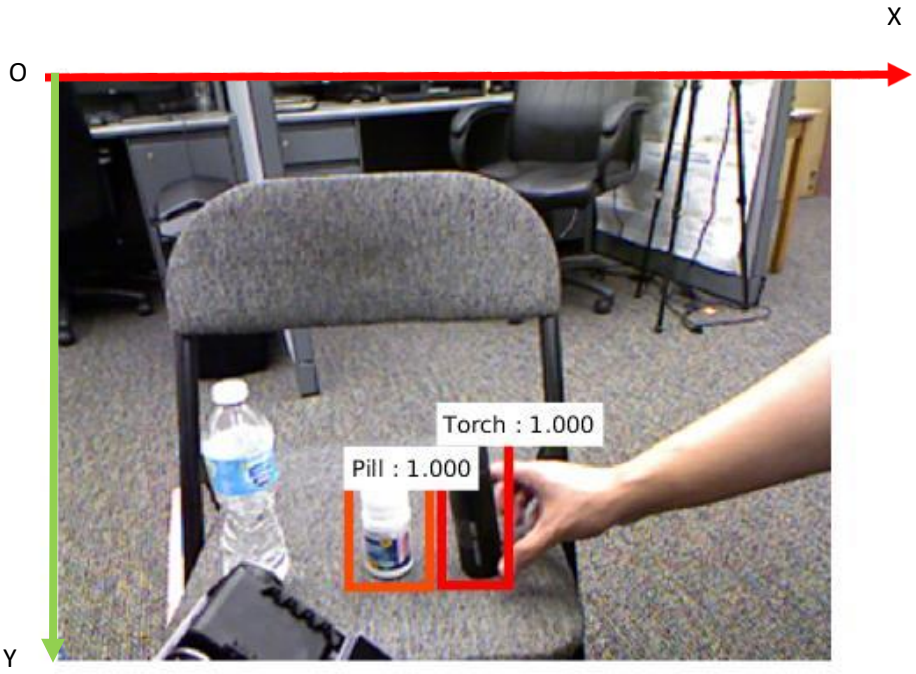


Figure 2-5 Image Coordinate System

To register the depth coordinate system to the image coordinate system, we need to use the built-in function provided by the development environment for Kinect. The function will be able to find the corresponding points on the depth image for each location on an RGB image. There are currently four development environments.

1. Official Kinect for Windows SDK
2. CL NUI Platform.

3. OpenKinect/Libfreenect
4. OpenNI(Opennatural Interface)

The official SDK only supports development on the Windows platform. After testing, I chose OpenNI as the development library since it has the most stable support by the ROS package and returns the most useful data.

2.4 Robot Operating System (ROS) and its advantages

The Robot Operating System (ROS) [15] was first introduced by the Willow Garage Team in 2010, and this open source software suite led to a revolution in robot development. The advances under this initiative made programming easier and more adaptable. ROS is a distributed processing system. The main purpose of ROS is to make the robot program more reusable. In other words, the shared free open source robot program used different programming languages to make the concept easier. The basic communication mechanism is shown in Figure 2-6 [14].

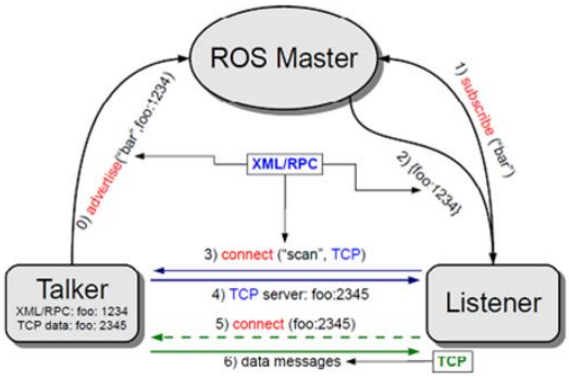


Figure 2-6 Communication mechanism of ROS [14]

I chose to use ROS in my project for the following reasons:

1. P2P mechanism

Our system includes a robotic arm control module, robotic vision perception module, Gazebo model module, simulation plugin module, object detection module, and a robot kinematics module. Putting all modules in one computer would be a real burden. Point to Point (P2P) design of the ROS system can distribute the tasks to different computers which can increase the processing speed of dealing with these tasks. Figure 2-7 [14] shows the P2P model of ROS.

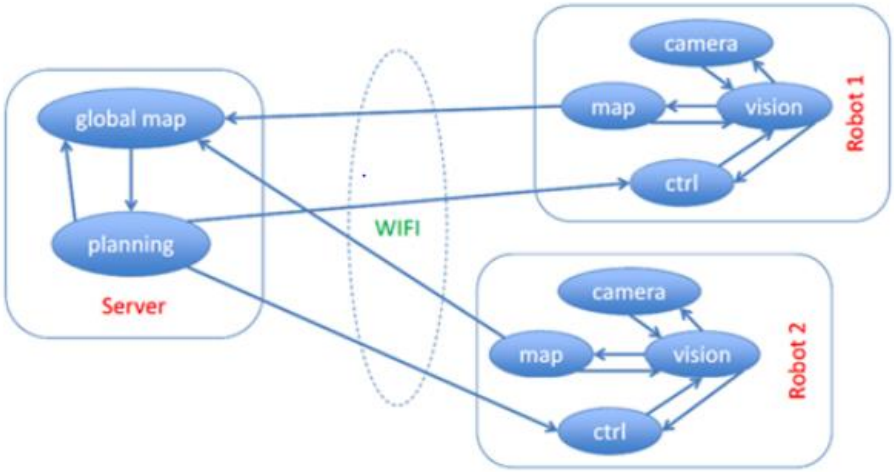


Figure 2-7 P2P model of ROS [14]

2. Multi-language Support

When dealing with a specific task, different programming languages may have their own advantages and disadvantages. For example, Faster-R-CNN only supports Python and MATLAB® code, while I'm familiar with C++ to control the robotic arm. In this case, ROS can transfer the data between the object detection module written in MATLAB® and the control module written in C++.

3. Sufficient Toolbox

ROS has included a lot of useful tools which made it much easier to debug and develop my project. It has a RViz toolbox which can visualize multiple kinds of Kinect messages such as depth image, point cloud, RGB image. Also, the Gazebo Simulator helped me build the whole simulation model of the robotic arm which can avoid damage to the servo motor when they are overloaded due to program problems.

2.5 Gazebo Simulation Environment

Gazebo is a standalone 3D simulation environment for multiple robotic platforms. Its powerful physical engine made it a popular simulation environment among robotic application developments. ROS is one of the platforms that can work with Gazebo. When installing ROS Indigo on my computer, the default Gazebo was version 2.2.3, which is not a stable one. So, I updated the Gazebo version to 2.2.5. The Gazebo system is formed by Gazebo master, communication library, physics library, rendering library, sensor generation, graphical user interface (GUI), and plugins.

An example is shown below in Figure 2-8. This model is simply for the robotic arm being used. I will need to add an object and Kinect in later work.

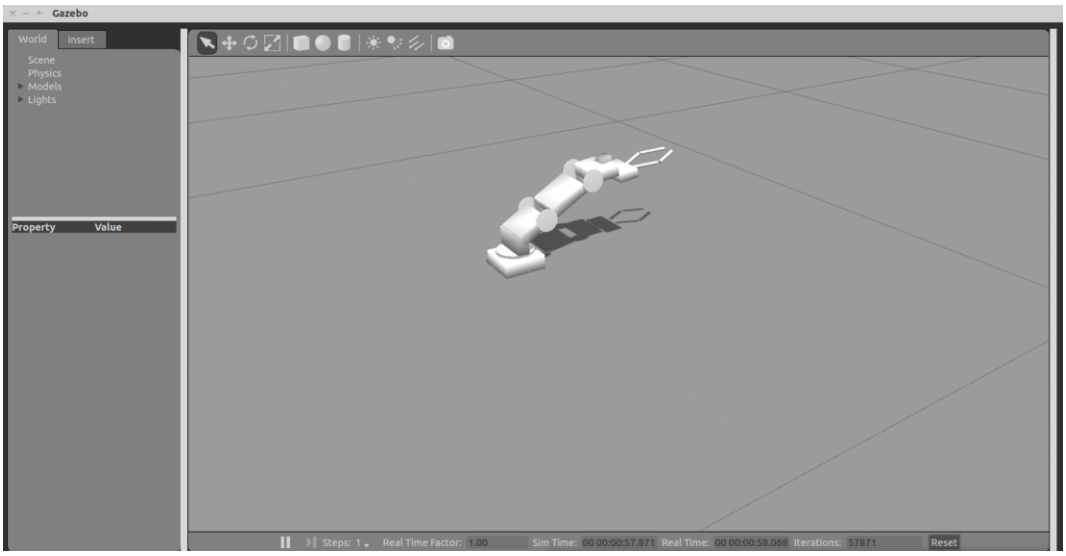
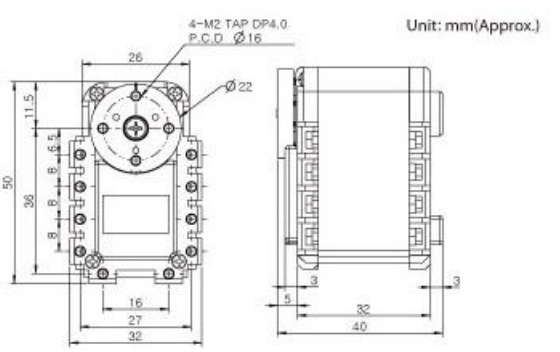


Figure 2-8 Example of model in Gazebo GUI

The Gazebo model is written in an SDF file, which describes each component of the model such as cylinders, box, joint type, weight, etc. To make it possible to communicate with the model, we need to write plugins defining the function of the joints or camera and link them to the corresponding part in the SDF file. The detail will be discussed later in further methodology discussion.

2.6 Dynamixel Motors

I used the Dynamixel motor AX-12A shown in Figure 2-9 as the servo motor for the robotic arm. The basic parameters are shown in Figure 2-9 [16]. Joint 1 consists of one motor. Joint 2 and 3 consists of two motors since they need to lift the object and support the weight of the arm. Joint 4 has one motor and it only deals with the rotation at the end of the robotic arm. One motor is used for gripper open-and-close control.



	Unit	Data
Weight	g	54.6
Dimension	mm	32 x 50 x 40
Gear Ratio	type/material	254:1 (Enpla)
Network	-	TTL
Position Sensor (resolution)	-	Potentiometer (300°/ 1024)
Motor	-	Cored
Operation Voltage	V	9~12
Stall Torque	N.m	1.5 at 12V
Stall Current	A	1.5 at 12V
No Load Speed	RPM	59 at 12V

Figure 2-9 Dynamixel motor AX-12A taken from [16]

The Dynamixel motor is quite popular among small robotic arms instead of traditional servo motors. It consists of a reduction gear, controller, driver and network, all of which are illustrated in Figure 2-10 [16].

Figure 2-10 Basic formation of Dynamixel motor [16]

AX-12 is a small but powerful servo motor with a 360 degree rotation capability, which can satisfy most movement requirements of the robotic arm. Also, the larger stall torque, which is 1.5 N/m, ensures it can handle some heavy objects compared to a traditional servo motor like Futaba, which has a max stall torque of 0.98 N/m. AX-12 can also resist some impact from the outside which would make its performance steady when there is a sudden force applied to the robotic arm. The serial network of AX-12 greatly improved the way traditional servo motors connect and communicate with the controller. AX-12 also has speed, temperature and angle feedback through its serial network, which can monitor the motor status and further improve the robotic arm performance.

2.7 Faster-R-CNN

In recent years, Ross Girshick and his team has made great progress in object detection, from R-CNN [9] to Fast R-CNN [11], and then Faster R-CNN [7]. They have made this kind of object detection method much faster and more accurate. The Faster R-CNN has reached a speed of 17 fps on simple neural network object detection tasks, with a 59.9% accuracy on PASCAL VOC, and 5 fps speed on complicated neural networks with an accuracy of 78.8%.

The Convolutional Neural Network (CNN) approach has been introduced to the image processing field for a long time. It has greatly reduced the computational cost since one of its core technologies shares the weight parameters such that we do not have to compute a full connected neural network, which would be extremely time consuming when dealing with images.

Faster R-CNN [7] was developed step by step from the R-CNN (region-convolution network) to Fast R-CNN and then Faster R-CNN. The key concept of this development is to make the deep neural network do more work, such as classification and region proposing, making it possible to finally achieve all tasks within the network. The key features of these three networks are shown in Figure 2-11 [24] below.

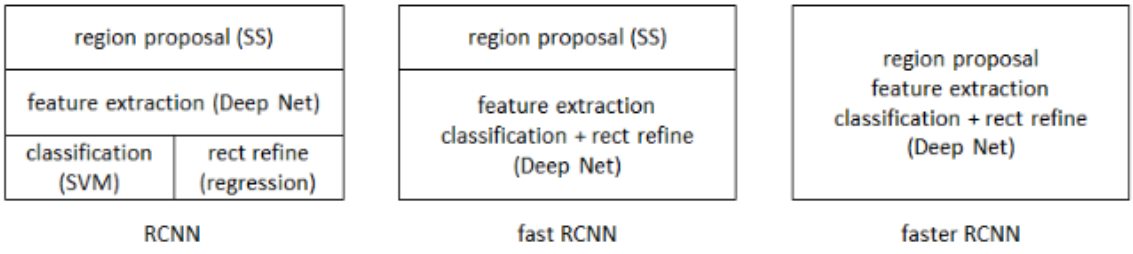


Figure 2-11 Key features R-CNN, Fast R-CNN, and Faster R-CNN [24]

The main frame work of Faster R-CNN is shown below in Figure 2-12 [7]. Faster R-CNN, as defined by the author, is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

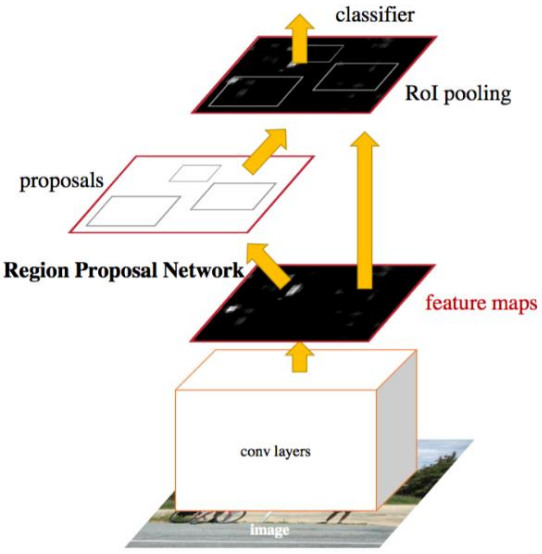


Figure 2-12 Faster R-CNN Main Framework [7]

The structure of the Zeiler and Fergus model (ZF model), which I utilized by implementing the first five convolution layers as shown below in Figure 2-13 [18]. The network will work per the following steps:

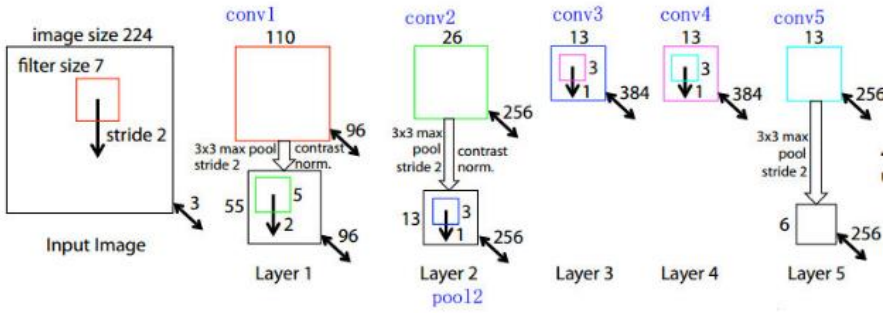


Figure 2-13 First Five Faster RCNN layers [18]

1. As per example, took input image 3×224×224 (RGB uses 3 channels)

2. Used 7×7×3×96 kernel to do convolution on the resized image.

3. After the convolution, we obtained 110×110×96 feature map, and then used 3×3 max pool with stride of 2, thereby obtaining a 55×55×96 feature map, which is the first hidden layer. Here we used the Zeiler and Fergus model (ZF model) as an example, which has five sharable convolution network layers. However, larger networks are available such as VGG16 which has 13 sharable layers. Another thing we need to mention is that we used the pad to overcome the edge problem as the author did.

4. Using the same kernel to increase convolution by four more times, yields the remaining four layers of neural networks. The final output is a 13×13×256 feature map, which will be sent to the RPN network and Fast R-CNN network.

2.7.1 RPN Network

Figure 2-14 [7] below shows the main frame of an RPN and its example for illustration

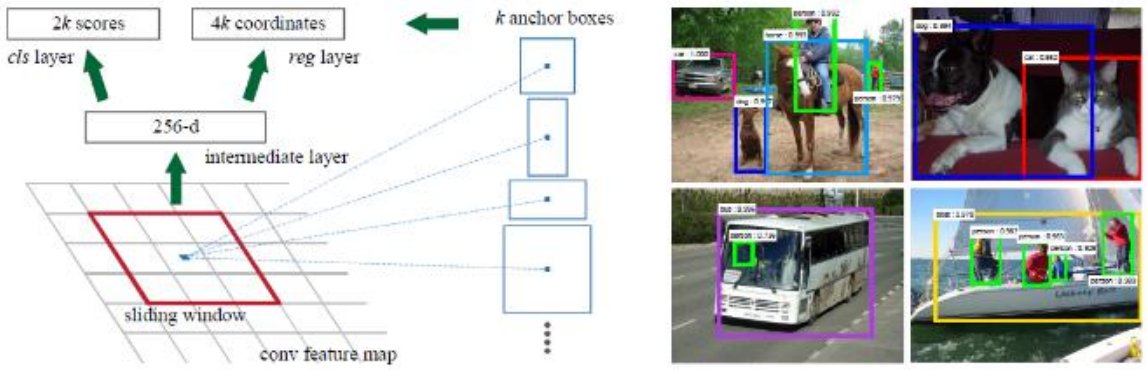


Figure 2-14 RPN and example for illustration [7]

Since the input convolution feature map has a dimension of 13×13×256, using a 3×3×256×256 sliding window, for each window, we can get a 1×1×256 vector. Then we fully connect this vector to a box-classification (cls) layer and a regression layer.

For each window center, we generate k anchors (3 scales (1×, 2×, 3×) ×3 length-width ratio (1:1, 1:2, 2:1) =9 here). The cls layer will give the possibilities of the anchor being included as an object or not. The regression layer will give the two diagonal point coordinates. So, for each sliding window center, there are 2k cls layer outputs and 4k regression layer outputs.

To determine if an anchor has an object inside, we must determine if the anchor has an overlap of over 0.7 with the ground truth bounding box; then, we assign it a positive label. If the overlap is less than 0.3, we give it a negative label. The remaining anchors will be abandoned if the overlaps are between 0.3 and 0.7. So, using the parameters from the cls layer and regression layers, we can build the loss function:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) \ +\lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*). \qquad (1)$$

In the loss function above, Pi and Pi* are outputs from the cls layer. Pi is the possibility of the anchor anticipating the right region. If the anchor obtains a positive label, then Pi*=1, otherwise Pi*=0. t_i is the predicted bounding box for anchor i, and $t_i{}^*$ represents the ground truth bounding box coordinates. Since N_{cls} is roughly 256 in a minibatch and N_{reg} is roughly 2400 in a minibatch. Then λ here is 10 in practice so that both terms have equal weights in the calculation. L_{cls} is the log loss function, and L_{reg} is the smooth L1 loss function. They are defined in (2) and (3) below:

$$L_{cls}(p_i,p_i{}^*) = -\log[p_i{}^*p_i + (1-p_i{}^*)(1-p_i)] \qquad (2)$$

$$L_{reg}(t_i, t_i^*) = \begin{cases} 0.5(t_i - t_i^*)^2 & if\ |t_i - t_i^*| < 1 \\ |t_i - t_i^*| - 0.5 & otherwise \end{cases} \quad (3)$$

2.7.2 Fast R-CNN

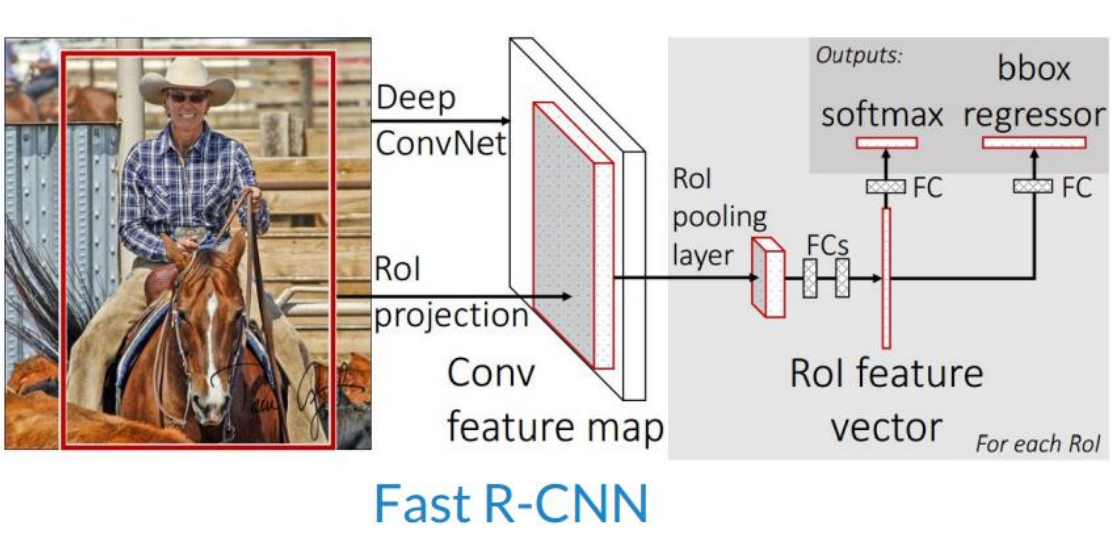


Figure 2-15 Fast R-CNN structure [11]

Figure 2-15 [11] above shows the basic structure of a Fast R-CNN network. The ROI in Fast-R-CNN would be generated from the RPN network instead of selective search [7] that is previously used. The network itself is quite straight forward, the Loss function for Fast R-CNN is:

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{loc}}(t^u, v) \quad (4)\ [11]$$

True box coordinates
Predicted box coordinates
True class scores
Predicted class scores
Log loss
Smooth L1 loss

In Fast R-CNN the positive samples are defined as those that have an intersection-over-union (IoU) with the ground truth bounding box that is larger than 0.5. The "negative

samples" are extracted from those proposals that have an IoU with a ground truth box ranging in [0.1, 0.5). The ratio for positive and negative samples in a minibatch is 1/3.

2.7.3 Training Process

The total training process for Faster R-CNN can be concluded in four steps:

Step 1. Set the initial parameters for the first five convolution layers the same as an "ImageNet" pre-trained model. Then train the RPN and generate proposals for next step.

Step 2. Use proposals generated from Step1 to train the Fast R-CNN network.

Step 3. Use the parameters of the model generated from Step 2 to initialize the RPN network and train it again.

Step 4. Fine tune (using a smaller learning rate) fully connected (FC) layers of Fast R-CNN using the same shared convolutional layers as in Step 3.

2.7.4 Training Results Evaluation

To evaluate the training result for a system, we usually use the precision and recall curve (PR Curve) to evaluate the performance of a system. In a system related to classification or detection problems, we usually need to have several parameter thresholds to classify different objects. For example, we have two kinds of samples: Some of them are positive, some are negative. They all have a values between the range [-1, 1]. We consider

those that have values smaller then 0 as negative and those that have values equal to or larger than 0 as positive samples. Now we have a trained system to classify these samples. The result returned by the system and the whole data set is shown in Figure 2-16 [19].

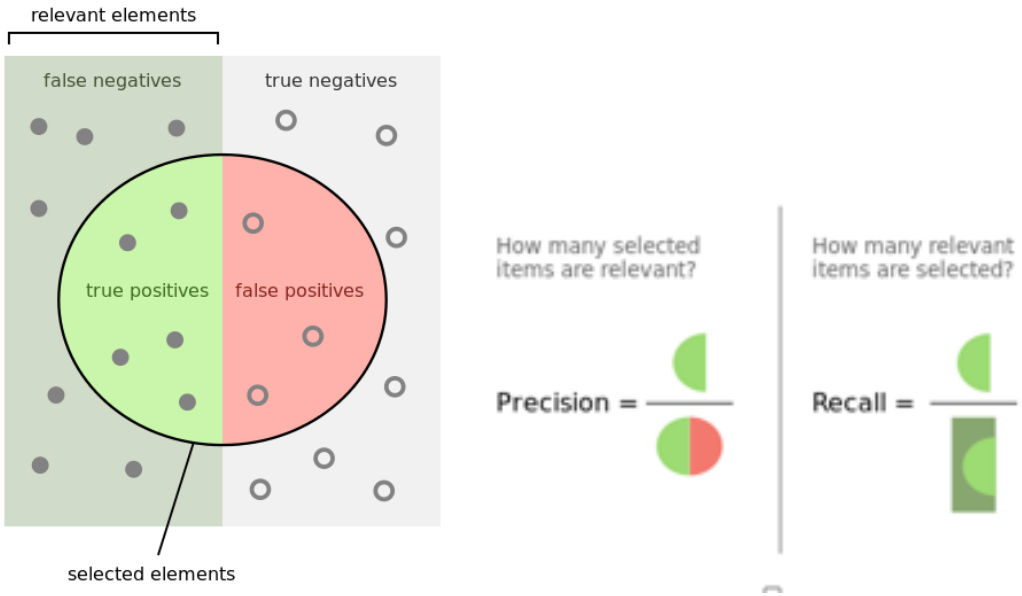


Figure 2-16 Results and whole data set

In Figure 2-16, “False negatives” are those values that are positive but labeled as negative by the system. “True Positives” represent samples that are positive and labeled as positive by the system. “False Positives” are those negative samples that are labeled as positive by the system. And “True Positives” are those negative samples that are labeled as negative by the system.

So, in Figure 2-16, we can see that “Precision” is defined as the ratio of “True positives” to the sum of “True Positives” and “False Positives,” which is the sum of all the samples that are labeled “Positive” by the system. Moreover, “Recall” is defined as the ratio of “True Positives” to the sum of “False Negatives” and “True Positives,” which is the total

number of positive samples. Now by setting different thresholds, the system will be able to label different number of “Positives” in a data set. And the precision and recall parameters will change accordingly.

Figure 2-17 shows an example of a PR curve comparison between two algorithms. There are several ways to evaluate an algorithm by its PR curve. We can calculate the area under curve (AUC) for the PR curve and the one that has a larger area is considered better. We can also observe and estimate the distance from point (1, 1) to its nearest point on the curve. Point (1, 1) here means the system labeled all the real positive samples as “Positive” and all the real negative samples are labeled as “Negative/” So we can say that algorithm 2 is better than algorithm 1 in Figure 2-17 below.

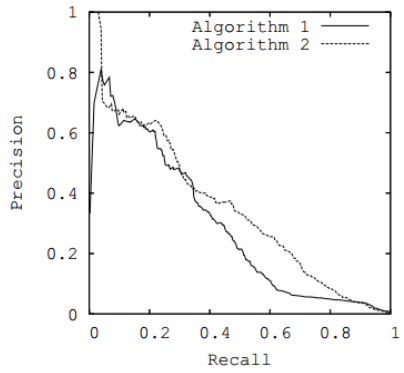


Figure 2-17 Example for Comparing PR Curve

CHAPTER 3 METHOD

The ultimate goal of this research was to develop a system that would control the robot to find and grasp the desired object using this proposed recognition and grasping system. To achieve this goal, the system first obtained both RGB and registered depth information of the current scene from the Microsoft Kinect. Then, an object detection system was implemented using RGB images to find the target object and return the coordinate of the object center in the image coordinate. Then, using the registered depth image, we had to find the corresponding 3D coordinates of the object center with respect to the RGB camera coordinate system of Kinect. Next, using a fixed coordinate transfer matrix, we compute the coordinate of the grasping point for the object with respect to the base of the robotic arm. Using robot kinematics, we obtained the angle parameter for each arm joint and grasped the object after deciding the best path. In this chapter, we first discuss how we acquired Microsoft Kinect data and transferred it into an ROS topic to enable it to be shared within the ROS system. Then, we described how the Faster-R-CNN handled the Kinect data to find out the 3D location of the object. Finally, we determined how to control the robotic arm to grab an object.

3.1 Acquiring RGB and Depth Data and Publishing to ROS Topic

The Microsoft Kinect currently has one of the most popular 3D sensors. It can return both RGB and depth images. Since we are using ROS as the main system for the software platform, we had to transfer the data into ROS topics.

As the name Kinect for Windows suggests, there is no official API or drivers from Microsoft released for a Linux platform. So, we had to choose one from three open source

Kinect APIs: CL NUI Platform, OpenKinect/LibFreenet and OpenNI. After testing both the OpenKinect and OpenNI+Primesense driver, I found that OpenNI was the most appropriate API for using Kinect on a ROS system since it has a related ROS package and tutorials.

There are many topics published by the OpenNI launch file shown in Figure 3-1. However, we only used two of them, the first one is "/rgb/image_color," and the second one is "depth_registered/points."

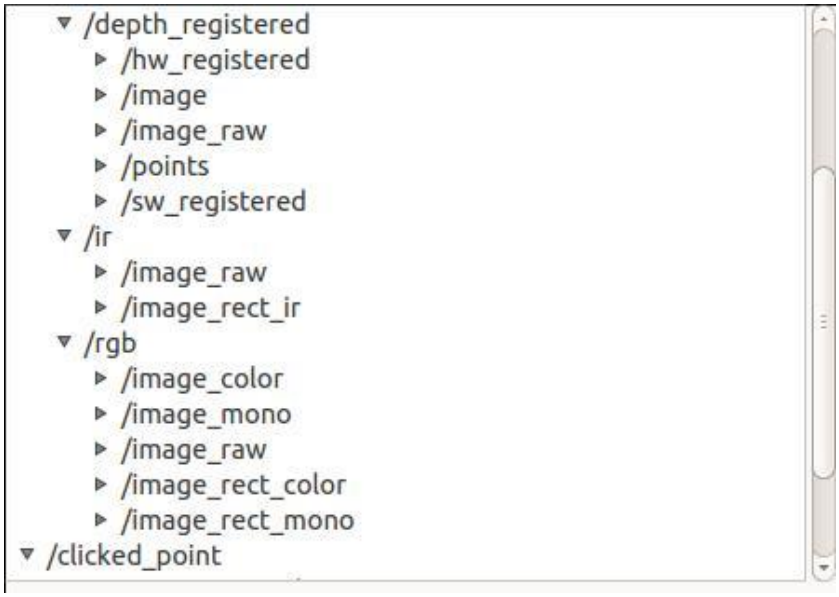


Figure 3-1 Topics published

The first topic is quite self-explanatory, while the second one is actually a registered depth image from an original depth camera coordinate system to RGB camera coordinate system. As introduced in the introductory background material at the first of this chapter, the coordinate systems of a depth camera and RGB camera are not the same since they are at different positions. So a registration process is needed in order to align the depth image to a corresponding RGB image so that we can find an object in the RGB image and find its

corresponding distance in a registered depth image. Luckily, the ROS OpenNI driver already provided the easy setup function [20] to only publish the depth image topics that are already registered to RGB images. The dynamic reconfigured GUI is shown in Figure 3-2. By enabling the depth_registration check box and setting depth_ir_offset_x and depth_ir_offset_y value, ROS driver will then receive registered depth images from the underlying OpenNI freamwork and will no longer publish unregistered data.

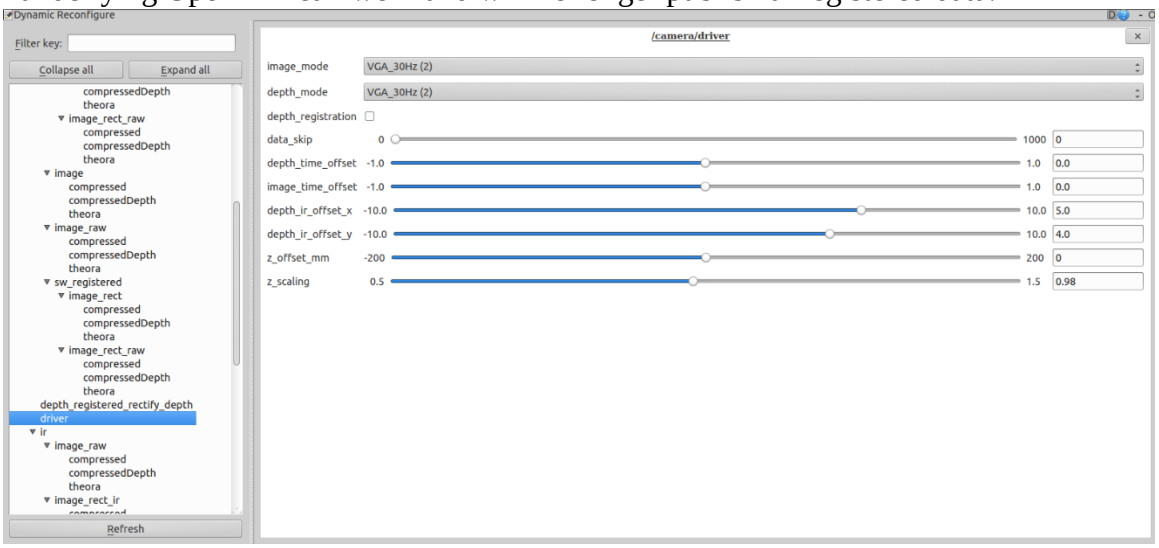


Figure 3-2 Reconfigured GUI

Now using a disparity image, we can see the differences between before and after a depth image was mapped to a RGB image in the RViz 3D visualization tool provided by ROS shown in Figure 3-3, and the depth image mapped with RGB image in Figure 3-4.

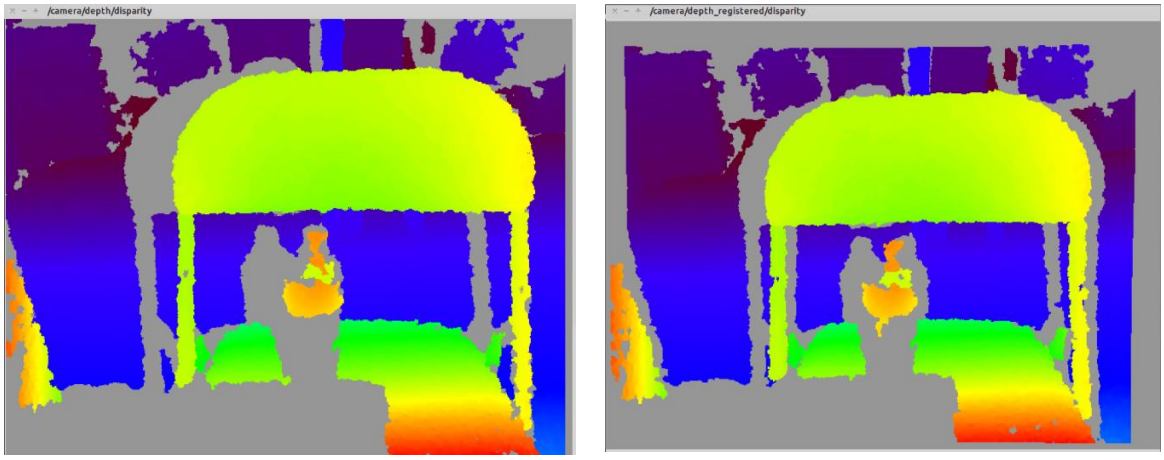


Figure 3-3 Unregistered and registered depth images in using a disparity image

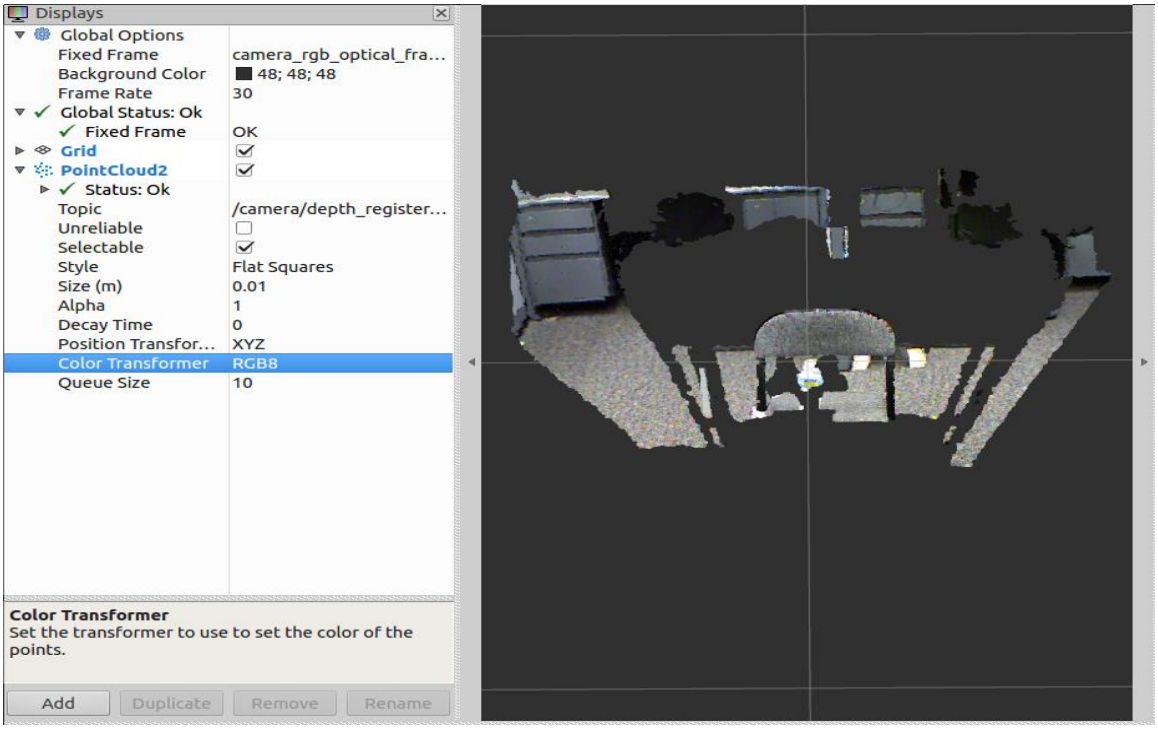


Figure 3-4 Registered depth image: The depth image is shown as a point cloud in a 3D view. Each point in the point cloud is given an RGB value, so we can see the point cloud as a 3D scene.

3.2 Transfer Data Between Server and Laptop

All the RGB and depth images were acquired by a designated laptop. But the laptop was much slower in computing than the server desktop, so we had to transfer the data to the server to finish computation and then send some simple results back to the laptop.

Since ROS is designed to be a distributed computing system, only one ROS master is needed for the system, which enabled us to run multiple tasks on several computers at the same time to increase the processing performance. We chose to achieve this distributed computing feature by using SSH communication [21].

Since we set the laptop as the ROS master server, suppose its IP address is 192.168.1.0, and the desktop's address is 192.168.1.1. So, on the desktop, we need to first ssh into the laptop. If the ssh was successful, the terminal's name changes from current computer to the laptop user name shown in Figure 3-5. At this remote terminal, we need to export the ROS material as the master IP address 192.168.1.0 and also export the ROS IP as 192.168.1.0 so that ROS knows that we cannot use host name directly.

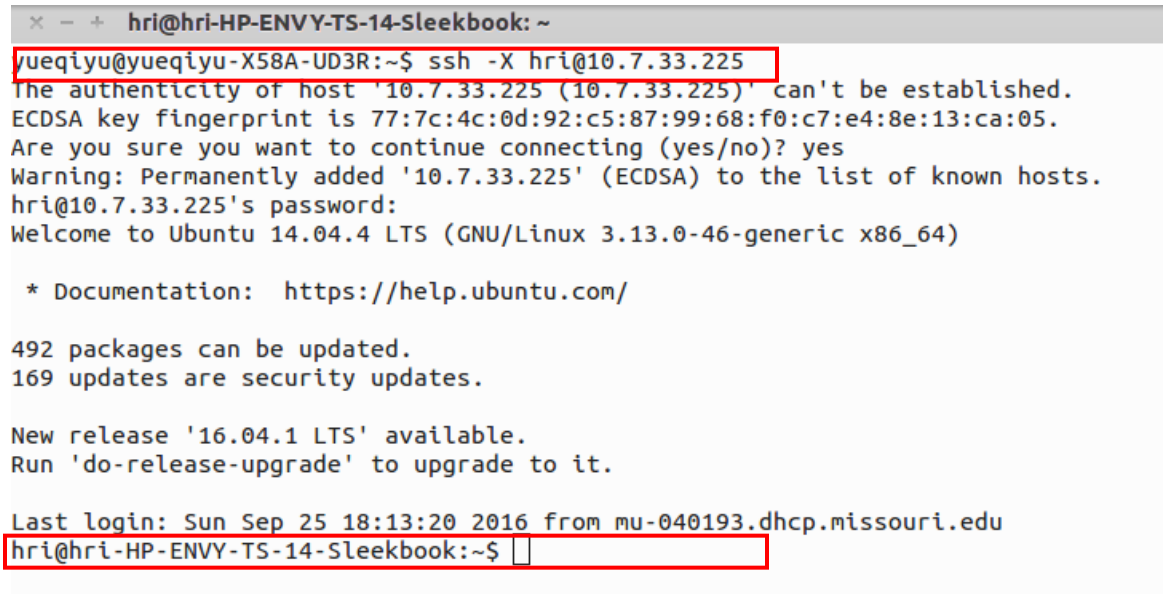
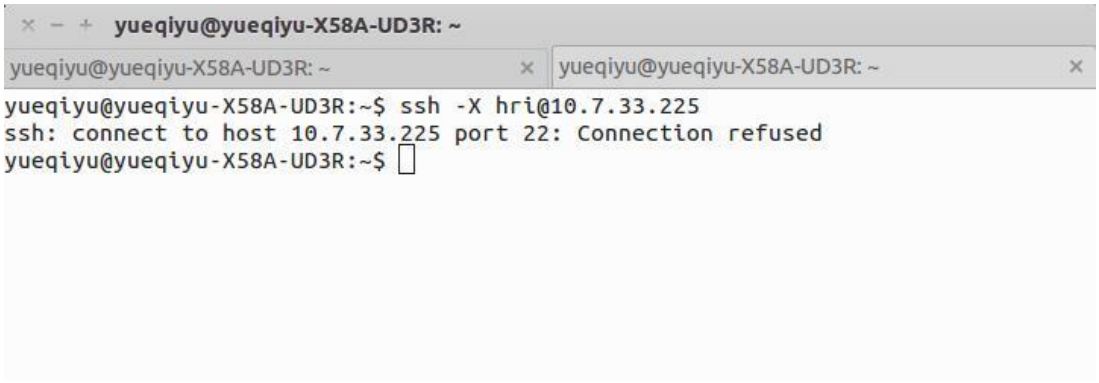


Figure 3-5 SSH results

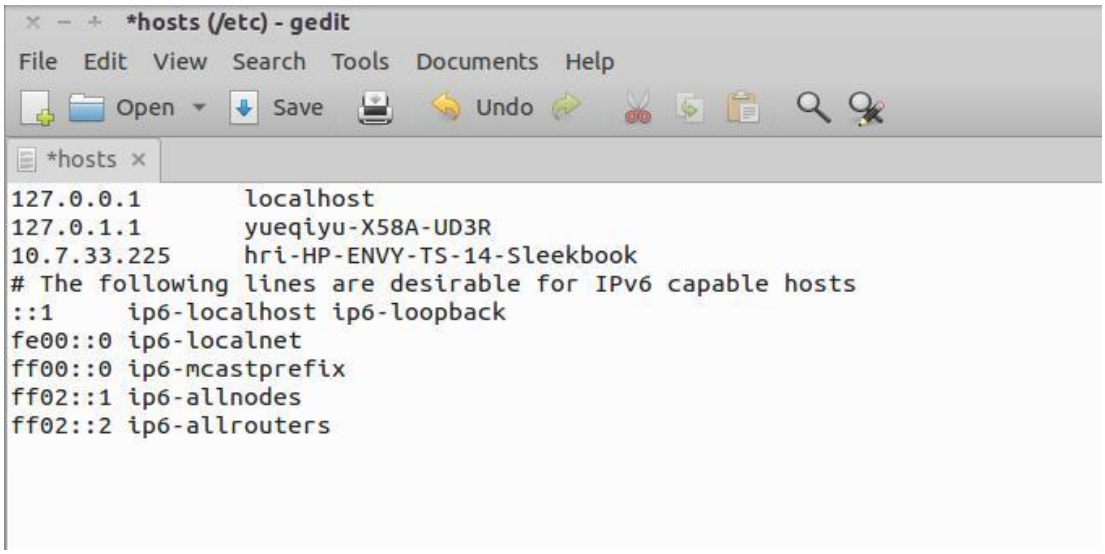
Then, on the laptop terminal, we exported the same ROS master URI, but the desktop ROS IP address was 192.168.1.1. However, after finishing all the steps, we found two problems:

1. Since the laptop was using WIFI to connect to the internet, it had a floating IP address, which made us realize that we needed to repeat the setup steps each time we ran the system.
2. When we were trying to communicate with the ROS master on desktop, an error appeared as shown in Figure 3-6(a). This was due to a firewall issue, requiring the

addition of this line (Figure 3-6(b)) to the etc/hosts file so that the firewall would not block data from this address.



(a) Problems when ssh is not properly configured



(b) Fixing the issue

Figure 3-6 Communication error and fix

3.3 Robotic Arm

3.3.1 Robotic Arm Control Using ROS

As noted in Section 2.6, we used AX-12A as the joint servo motor of the robotic arm used in the project. The robotic arm structure is shown in Figure 3-7. And to better communicate with and control the motor, we assigned an ID to each servo, from the first joint to the last one. Servo 1 is at the bottom, servo 2 and 3 are at the second joint, servo 4 and 5 are at the third joint, and servo 6 is at joint 4. Servo 7 is used for gripper control. For joint 2 and 3 with servo pairs, each pair of servos was placed at an opposite position, so that all the communication wires could hide inside the robotic arm. However, this setting indicates that when controlling the servo pairs, the pair of servos have to rotate in opposite directions to make the arm move toward one direction. We used a 12v 2.67A power adapter as the power input for the servos, which was sufficient.

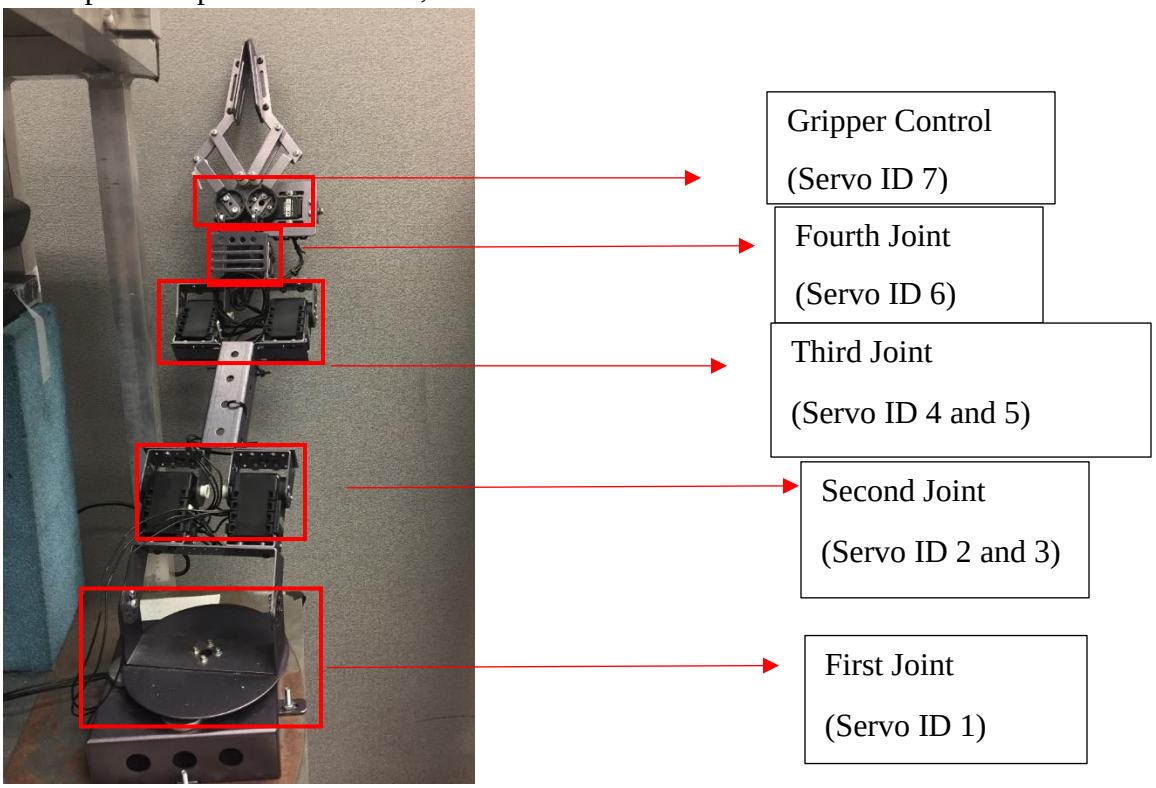


Figure 3-7 Structure of the robotic arm

To control the motors over ROS, we first need to install a related package which will enable us to interface with the Robotis Dynamixel line of servo motors. The package is called dynamixel_motor, and it supports multiple kinds of Dynamixel motors.

The Dynamixel servos are all connected to the computer through the USB port. If they are in a network, only one USB port will be used. Under the Ubuntu system, we assumed the servos were connected to the /dev/ttyUSB0 serial port. We had to create a launch file as a controller manager that would connect to the motors, find out the motor number and publish topics to the ROS system containing the motors' current raw feedback data such as current position (in angle), goal position, and error at a specific rate. Figure 3-8 shows key lines in the launch file that were used to configure the controller manager.

```
---------
    namespace: dxl_manager
    serial_ports:
        pan_tilt_port:
            port_name: "/dev/ttyUSB0"
            baud_rate: 1000000
            min_motor_id: 1
            max_motor_id: 25
            update_rate: 20
```

Figure 3-8 Configuration line

As shown in Figure 3-8, we found the communication port named /dev/ttyUSB0. We also set the communication rate at 1000000 (baud rate may vary based on different servos. We set the correct baud rate at AX-12A). Then, since we had seven motors with IDs from 1-7, we commanded the controller manager to search motors with IDs from 1–25. Finally, we set the communication update rate at 20 times per second.

When first using the controller manager, there was one problem shown in Figure 3-9. We received this error message from the terminal. After searching for the reason, we found that we needed to open the write and read permission of the USB port to the current user. We simply typed “sudo chmod 777 /dev/tty/USB0” to solve the problem.

```
serial.serialutil.SerialException: could not open port /dev/ttyUSB0: [Errno 13]
Permission denied: '/dev/ttyUSB0'
================================================================================
REQUIRED process [dynamixel_manager-1] has died!
process has died [pid 3062, exit code 1, cmd /opt/ros/indigo/lib/dynamixel_contr
ollers/controller_manager.py __name:=dynamixel_manager __log:=/home/yueqiyu/.ros
/log/6e87b89c-8655-11e6-8996-1c6f653cce4a/dynamixel_manager-1.log].
log file: /home/yueqiyu/.ros/log/6e87b89c-8655-11e6-8996-1c6f653cce4a/dynamixel_
manager-1*.log
Initiating shutdown!
--------------------------------------------------------------------------------
```

Figure 3-9 Configuration line

After successfully launching the controller manager, we could see the output from the terminal in Figure 3-10, and we could put up the first three of the seven servo’s feedback shown in Figure 3-11.

```
[INFO] [WallTime: 1475161467.472879] pan_tilt_port: Pinging motor IDs 1 through
25...
[INFO] [WallTime: 1475161469.289471] pan_tilt_port: Found 7 motors - 7 AX-12 [1,
 2, 3, 4, 5, 6, 7], initialization complete.
```

Figure 3-10 Successfully found seven motors

In Figure 3-11 we can see that the returned information gives detail about each servo’s ID, goal position and current position, error rate, current speed and load etc. By monitoring these information, we can control the robotic arm accurately to reach the goal position,

```
---
motor_states:
  -
    timestamp: 1475161644.29
    id: 1
    goal: 502
    position: 502
    error: 0
    speed: 0
    load: 0.0
    voltage: 12.1
    temperature: 39
    moving: False
  -
    timestamp: 1475161644.29
    id: 2
    goal: 128
    position: 128
    error: 0
    speed: 0
    load: 0.0
    voltage: 12.1
    temperature: 35
    moving: False
  -
    timestamp: 1475161644.3
    id: 3
    goal: 886
    position: 886
    error: 0
    speed: 0
    load: 0.0
    voltage: 11.8
    temperature: 36
    moving: False
  -
```

Figure 3-11 Controller management of published motor s' feedback

Now that we can freely communicate with the servos, we had to create a launch file so that we could control the servos. First, we had to create a configuration file that had all the parameters necessary for our controller. For example, for servo 1 we set the parameters as shown in Figure 3-12.

```
joint1_controller:
    controller:
        package: dynamixel_controllers
        module: joint_position_controller
        type: JointPositionController
    joint_name: tilt_joint1
    joint_speed: 0.6
    motor:
        id: 1
        init: 512
        min: 0
        max: 1023
```

Figure 3-12 Servo 1 setting

Since servo 1 is the base joint of the arm, we had to limit its maximum speed, so there would not be a great shaking when the first joint rotated to the designated position. Then, we need to create another launch file to enable us to control the servos. We assigned the name of the topics for each servo. The launch file assigned the names and published the topics to the ROS system. Figure 3-13 shows the result after the control launch file for each servo was successfully started. Figure 3-14 shows the published topics of this launch file.

```
[INFO] [WallTime: 1475161972.582598] pan_tilt_port controller_spawner: waiting f
or controller_manager dxl_manager to startup in global namespace...
[INFO] [WallTime: 1475161972.587039] pan_tilt_port controller_spawner: All servi
ces are up, spawning controllers...
[INFO] [WallTime: 1475161972.635419] Controller joint1_controller successfully s
tarted.
[INFO] [WallTime: 1475161972.673586] Controller joint2_controller successfully s
tarted.
[INFO] [WallTime: 1475161972.715016] Controller joint3_controller successfully s
tarted.
[INFO] [WallTime: 1475161972.751301] Controller joint4_controller successfully s
tarted.
[INFO] [WallTime: 1475161972.786687] Controller joint5_controller successfully s
tarted.
[INFO] [WallTime: 1475161972.819391] Controller joint6_controller successfully s
tarted.
[INFO] [WallTime: 1475161972.859177] Controller joint7_controller successfully s
tarted.
```

Figure 3-13 Successful start of the controller

```
yueqiyu@yueqiyu-X58A-UD3R:~/catkin_ws$ rostopic list
/diagnostics
/joint1_controller/command
/joint1_controller/state
/joint2_controller/command
/joint2_controller/state
/joint3_controller/command
/joint3_controller/state
/joint4_controller/command
/joint4_controller/state
/joint5_controller/command
/joint5_controller/state
/joint6_controller/command
/joint6_controller/state
/joint7_controller/command
/joint7_controller/state
/motor_states/pan_tilt_port
/rosout
/rosout_agg
```

Figure 3-14 Published topics of the launch file

There are 2 main topics we used to control the servo and to monitor its status: /jointx_controller/command and jointx_controller/state. The "command" topic was used to control the corresponding servo, and the "state" topic was used to get the current status of the corresponding servo. Normally, if we wanted to control the first servo, we needed to publish the rotation angle of 1.0 in the radian system message to the joint1 command topic. The format would be:

```
rostopic pub -1 /joint1_controller/command std_msgs/Float64 -- 1.0
```

For now, the basic preparation work to control the robotic arm was finished. In a later process, we had to write programs that would publish calculated angles to each joint, so the robotic arm could move to a specific point. That program is called a client in the ROS system.

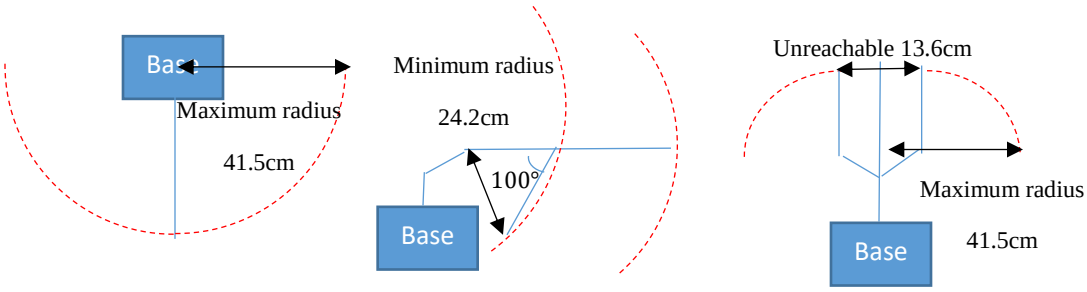
3.3.2 Gazebo Simulation

At the beginning, I debugged the control program directly on the robotic arm. Due to a mistake in the program that made servo pairs in joints 2 and 3 rotate towards the same direction, the 4 servos were damaged even with the overload protection mechanism. To prevent problems like this from happening again, I decided to use Gazebo to simulate the robotic arm and debug my control program on it first, and then control the real-world robotic arm.

Starting from the base, I used an $11 \times 11 \times 3.9$ cm box element sitting at the origin of the Gazebo world coordinate system as the base. In the real-world implementation, the arm base is settled at a fixed position and is connected to the robot tightly; so, I set the base model weight at 300 Kg so that it would not shake a lot while picking the object up. Then the first joint would be represented by a cylinder that has a radius of 5 cm and a thickness of 1cm. Then we attached two boxes to the first joint. The gripper was then built on the second joint. After building the physical model of the robotic arm, we set each joint's parameters. The first joint was able to rotate from -90 to 90 degrees, the second and third joints had a wider range of -120 to 120 degrees. The fourth joint, however, could rotate at 360 degrees. Also I set the fingers at two joints since there are two fingers. They can both rotate from -90 to 0 degree. Figure 3-16 shows the basic parameters of the robotic arm structure.

However, the robotic arm has some range limitations due the limited angle each joint can reach. So Figure 3-15(a) (b) (c) shows the simplified robotic arm from three viewpoints.

It can give us a general understanding of the spherical space range that the gripper can reach.



(a) Top View (b) Side View (c) Front View

Figure 3-15 Three Views of Robotic Arm and Its Range

Another important thing was to set up the appropriate coordinate for each joint. Thus, based on the robotic kinematics, we set each joint's z axis to align with joint segments, as shown in Figure 3-17.

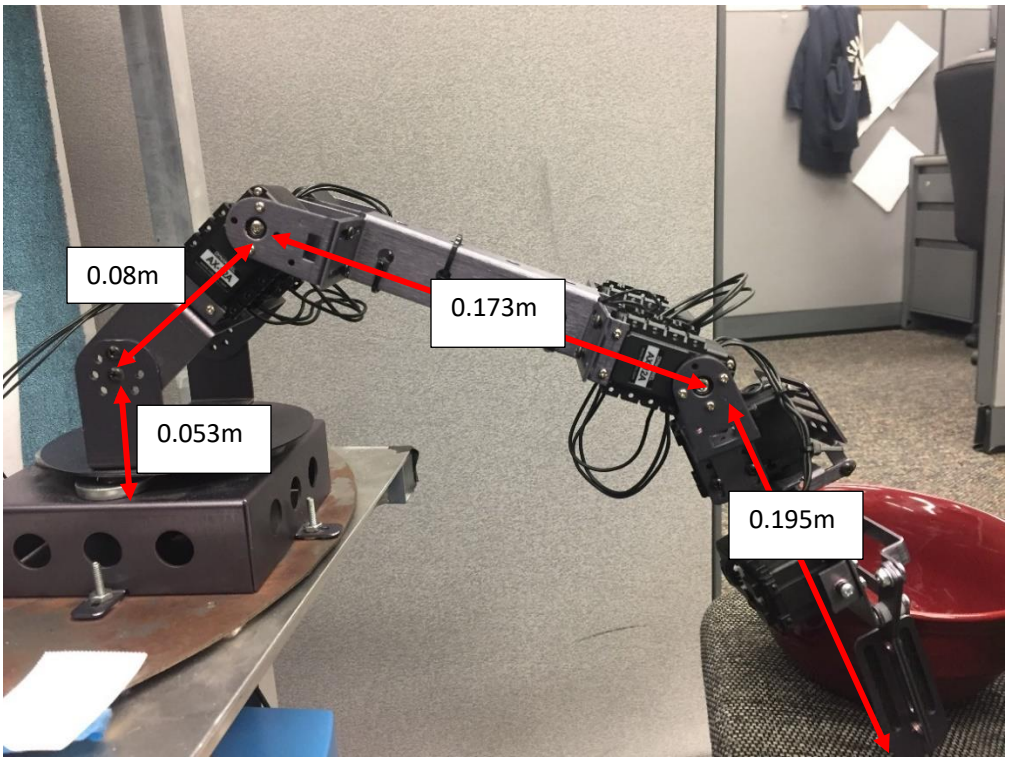


Figure 3-16 Robotic arm dimension

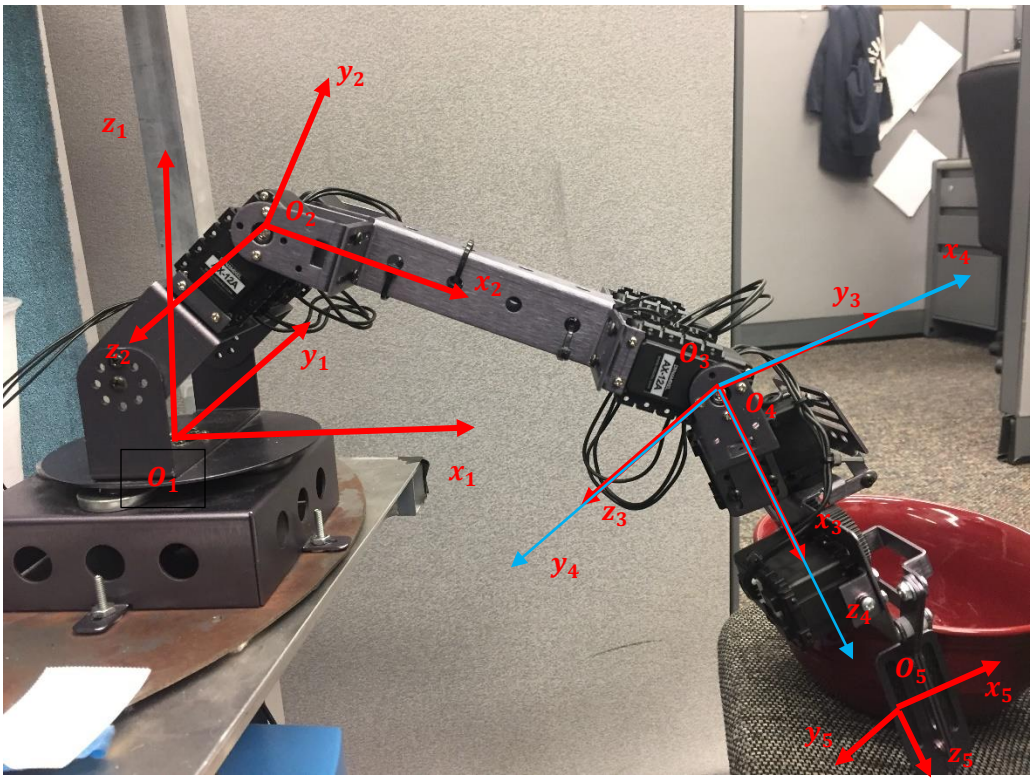


Figure 3-17 Robotic arm coordinate system

After the model was successfully built, we had to write plugins to control the model and make the model able to communicate with the ROS master.

There is a C++ library for plugins that is loaded by Gazebo at runtime. The plugin can access Gazebo's API to perform a wide range of tasks such as moving joints, applying force to an object, and publishing sensor data to the ROS master. The first plugin we had to create was for the robotic arm itself. All we had to control were the robotic arm's joints. So, we need to make the joints publish their status and subscribe control command messages to the ROS master. So, the plugin program should be linked to each of the joints and corresponding ROS topics.

3.4 Object Detection

Considering the size and capability of the gripper, I built the network to recognize a particular flashlight, pill bottle, coffee mug with handle, disposable cup and bowl. I divided the process to following steps:

1. Data Preparation
2. Training
3. Testing

3.4.1 Data Preparation

The data set I used here came from ImageNet and self-annotated images and some rotated images.

The rotated images were created to make the performance better since there is a slight perspective distortion in RGB images at the edge of the images, and the table surface was uneven, which made the object tilted with respect to the ground plane. So, I rotated the image along with the object and bounding box, using the built-in function of MATLAB to rotate the image around the center. Suppose we have $\mathrm{P}(x_i, y_i)$ as the current points' position matrix where i=1, 2, 3, 4, and $\mathrm{P}(x_i, y_i)$ represents the corner of the current bounding box. Then we transferred the image coordinate origin to the image center (T1 below) and then rotated around the z axis for α degrees as required (T2 below) and then transferred back to the first point of origin (T3 below).

$$\mathrm{T1} = \begin{bmatrix} 1 & 0 & -w/2 \\ 0 & 1 & -h/2 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathrm{T2} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathrm{T3} = \begin{bmatrix} 1 & 0 & w/2 \\ 0 & 1 & h/2 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

So the transfer matrix T is as follows:

We have the transferred points $P(x_i', y_i')$ where i=1, 2, 3, and 4:

and $$P(x_i', y_i') = P(x_i, y_i)T \quad (6)$$

However, to determine the new bounding box and ensure that it can cover the whole object after rotation, I used the following method to find the diagonal points.

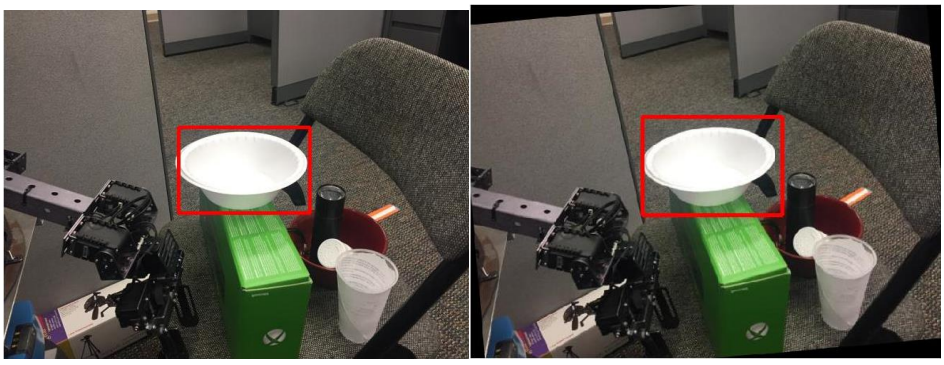
$$x_{d1} = Min(X) \; where \; X = [x_1', x_2', x_3', x_4'] \quad (7)$$

$$x_{d2} = Max(X) \; where \; X = [x_1', x_2', x_3', x_4'] \quad (8)$$

$$y_{d1} = Min(Y) \; where \; Y = [y_1', y_2', y_3', y_4'] \quad (9)$$

$$y_{d2} = Max(Y) \; where \; Y = [y_1', y_2', y_3', y_4'] \quad (10)$$

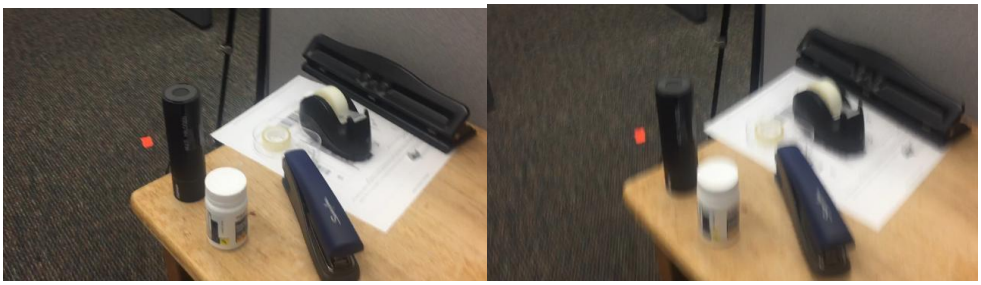
So we have the diagonal points $P_1 = (x_{d1}, y_{d1})$ and $P_2 = (x_{d2}, y_{d2})$ where P_1 is the left top corner point and P_2 is the lower right corner point. The comparison between rotated and original image is shown in Figure 3-18 as (a) and (b) below:



(a) Original picture with label (b) Rotated with label

Figure 3-18 (a) Original picture with label and (b) the rotated one

To get all the data for training and testing, I took ten videos for the flashlight and pill bottle in ten different backgrounds, each with a view of 360 degrees around them. After taking each video, I used MATLAB to extract all the frames from the video and saved as a JPEG format. Since the video frames were rated at 30 frames/second, a lot of the frames that were extracted from the video are quite out of focus. The comparison between the clear frame and the unfocused frame is shown in Figure 3-19.



(a) Clear Image (b) Unfocused image

Figure 3-19 Comparison between clear and unfocused images

We can see the one on the right is too vague to be used as a training picture. However, some frames that are not too vague can be used as training data and can increase the performance when dealing with objects that are slightly different. So, from 5000 frames extracted totally from videos, I chose 612 of them.

The data set has a total of 5018 images with the following formation:

1. 912 images with bounding box for bowl from ImageNet.
2. 187 images of bounding box for bowl I took myself, and rotated for -5, -4, -3, 3, 4, 5 degrees.

3. 124 images with bounding box for paper cup and disposable cup, and rotated for $-5, -4, -3, -2, -1, 1, 2, 3, 4, 5$ degrees to handle the perspective distortion of the camera.
4. A total of 624 images are for flashlight and pill bottle since they are quite specific and do not require a lot of images compared to the other three classes.

The Faster R-CNN network was trained with a VOC2007 data set [42]. So all the programs that were used to load the data were written specifically for the VOC2007 data format. The data was generated by picture in a folder designated as "JPEGImages,", and tags in XML format for each picture were filed in a folder called "Annotations." Pictures in the digital training data, training validation data, test data and test validation data without ".jpg" also required separate folders.

The basic procedure follows:

1. To name the pictures correctly, the name could only contain 6 digits; for example, picture 1 was named "000001.jpg" and picture 111 was named "000111.jpg".
2. A bounding box had to be drawn for targets in each picture and the information saved in the TXT file. An example of the saved information is shown in Figure 3-20, and the bounding box is shown in Figure 3-21. One line consists of picture's name, category, top left corner's x and y coordinates of the bounding box and lower right corner's x and y coordinates of the bounding box.

```
009979.jpg Torch 304 81 360 180 Pill 411 182 464 257
009980.jpg Torch 306 82 361 182 Pill 411 180 466 260
009981.jpg Torch 308 84 363 182 Pill 412 182 464 263
009982.jpg Torch 307 91 369 187 Pill 408 191 476 265
009983.jpg Torch 310 92 372 187 Pill 413 190 474 270
009984.jpg Torch 313 120 371 209 Pill 415 215 470 296
009985.jpg Torch 298 123 349 216 Pill 393 220 447 300
009986.jpg Torch 264 155 318 268
009987.jpg Torch 259 155 320 272
009988.jpg Torch 258 156 320 270
009989.jpg Torch 242 156 300 271
009990.jpg Torch 242 156 294 267
```

Figure 3-20 TXT file information

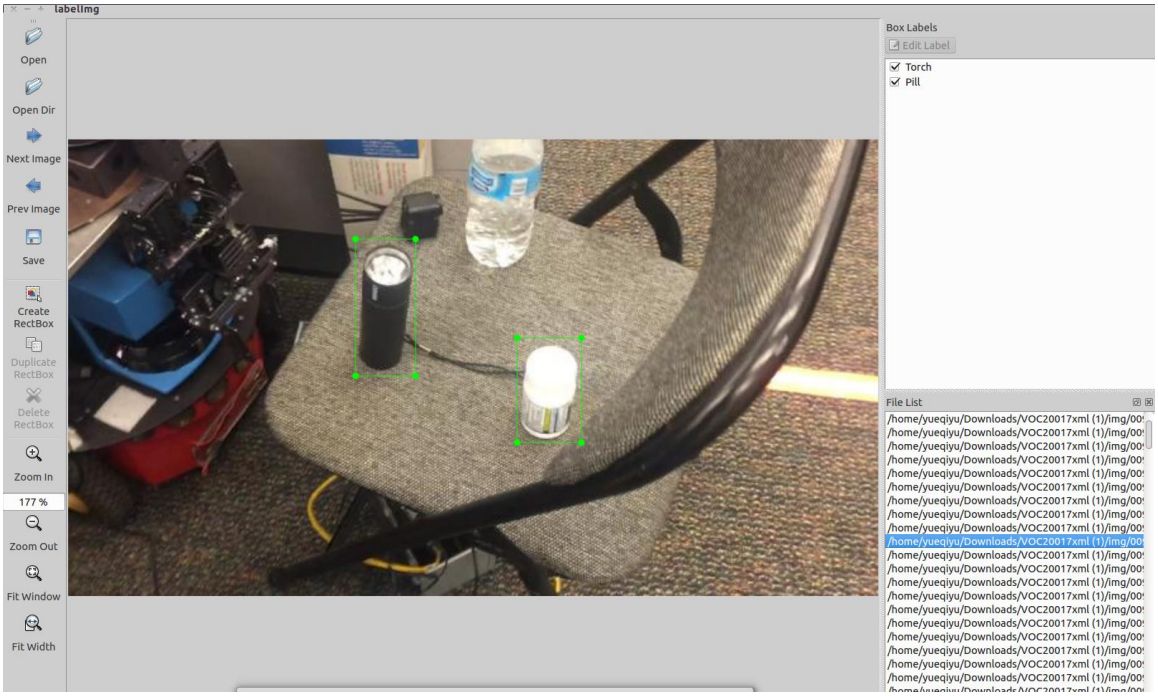


Figure 3-21 Drawings of bounding box for selected objects

3. Transfer the TXT file to several XML files for every picture. The XML file for each picture would look like Figure 3-23. However, sometimes, the XML file generated

by the labeling tool would contain the version information on first line as shown in Figure 3-22 below. This causes failure in training. So, I had to write a program to check these XML files and make sure no such version information exists in the annotation XML files.

```
<?xml version="1.0" encoding="utf-8"?>
<annotation>
    <folder>VOC2007</folder>
    <filename>000001.jpg</filename>
    <source>
        <database>My Database</database>
        <annotation>VOC2007</annotation>
```

Figure 3-22 Version information

```
<annotation>
        <folder>VOC2007</folder>
        <filename>010431.jpg</filename>
        <source>
                <database>My Database</database>
                <annotation>VOC2007</annotation>
                <image>flickr</image>
                <flickrid>NULL</flickrid>
        </source>
        <owner>
                <flickrid>NULL</flickrid>
                <name>xiaoxianyu</name>
        </owner>
        <size>
                <width>640</width>
                <height>360</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>Torch</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>302</xmin>
                        <ymin>146</ymin>
                        <xmax>346</xmax>
                        <ymax>286</ymax>
                </bndbox>
        </object>
</annotation>
```

Figure 3-23 XML Annotation file example

4. Then, we saved the XML files in the “Annotation” folder and put all the pictures in the “JPEGImages” folder.
5. The next step was to divide all the pictures into text files called “trainval.txt” (includes picture names that will be used for training and validation),”test.txt” (includes picture names for testing), “train.txt” (includes picture names for training) data and “val.txt” (includes picture names for validation). The pictures were all randomly picked without repetition to put into the four files. 2225 pictures were

used for testing. 1423 pictures were used for training, and the remaining 1370 pictures were for used for validation. The four files were put into path ImageSets\Main, which completed the data preparation process.

3.4.2 Training Process

Before starting the training process, we had to modify several scripts related to training since the original code was based on VOC2007 data sets.

To summarize this process：

1. I made all the file and folder names exactly the same as those in the VOC2007 data sets, and replaced the original files and folders.
2. I changed the tags in the script called “VOCinit” from the VOCdevkit package since there were only 20 classes in VOC2007 data sets and five classes to train.
3. The validation iteration usually pertains to 20% of the pictures used for validation. So I set this value at 300.
4. Then I changed the network model based on number of classes. Specifically, I changed the Fast R-CNN train model since its parameters are related to the number of classes of objects. The first change was made on the “train_val.prototxt” file at path “models\fast_rcnn\prototxts\ZF” and “models \fast_rcnn \ prototxts \ ZF_fc6 ”. I changed the input dim of “bbox_targets” and “bbox_loss_weights” to 24 which come from the equation:

$$\text{input dim} = (\text{number of class}(5) + \text{background}(1)) * 4 \quad (11)$$

where “4” represents the top left and lower right corner’s coordinates of the bounding box. Figure 3-24 below shows the changes in the train_val.prototext file:

```
input: "bbox_targets"
input_dim: 1  # to be changed on-the-fly to match num ROIs
input_dim: 24 # 4 * (K+1) (=5) classes
input_dim: 1
input_dim: 1

input: "bbox_loss_weights"
input_dim: 1  # to be changed on-the-fly to match num ROIs
input_dim: 24 # 4 * (K+1) (=5) classes
input_dim: 1
input_dim: 1
```

Figure 3-24 Changes for input dimension

5. The number of outputs had to be changed for the 7th full connection layer (fc7, as indicated in the prototext file) in the same prototext file of the Fast RCNN structure as shown in Figure 3-25.

```
layer {
        bottom: "fc7"
        top: "cls_score"
        name: "cls_score"
        param {
                lr_mult: 1.0
        }
        param {
                lr_mult: 2.0
        }
        type: "InnerProduct"
        inner_product_param {
                num_output: 6
```

```
layer {
        bottom: "fc7"
        top: "bbox_pred"
        name: "bbox_pred"
        type: "InnerProduct"
        param {
                lr_mult: 1.0
        }
        param {
                lr_mult: 2.0
        }
        inner_product_param {
                num_output: 24
```

Figure 3-25 Changes for output layer

6. The same area for the test.prototext file had to be changed for the fc7 output parameter numbers.

The training procedure is started by running the corresponding MATLAB script named “script_faster_rcnn_VOC2007_ZF.m.”

3.5 Object Picking

Based on the location data we received from the ROS topic, we first used direct kinematics to calculate the transformation matrices that we were going to use.

3.5.1 Object Location for Robotic Arm's Base

Based on the actual structure parameters here, we first had to calculate the object location based on the robotic arm's base. Suppose we had a world coordinate system XYZ with the Origin at (0, 0, 0) set for the base of the robotic arm, and suppose the camera coordinate system where its 3 axis are called noa with the Origin at (−0.13, 0, 0.31) with respect to the world coordinate system. We can see the two coordinate systems in Figure 3-26.

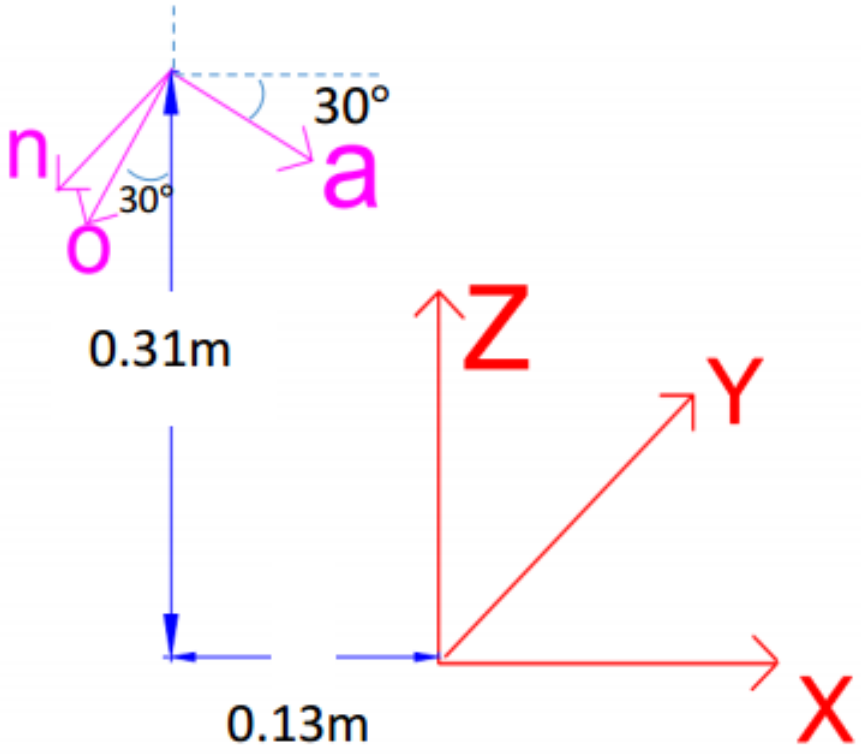


Figure 3-26 Coordinate system of robotic arm base and Kinect

If we had a point (a, b, c) in the space with respect to the camera coordinate system, we would have to find the point's coordinate under the world coordinate system. So, the transformation from world coordinate system to camera coordinate system would follow the following steps as shown in Figure 3-27 (a) (b) (c) (d):

2. A rotation of −90 degrees about the Z axis.
3. A rotation of 120 degrees about Y axis.
4. A translation of [−0.13, 0, 0.31] (relative to the x, y, and z axes, respectively).

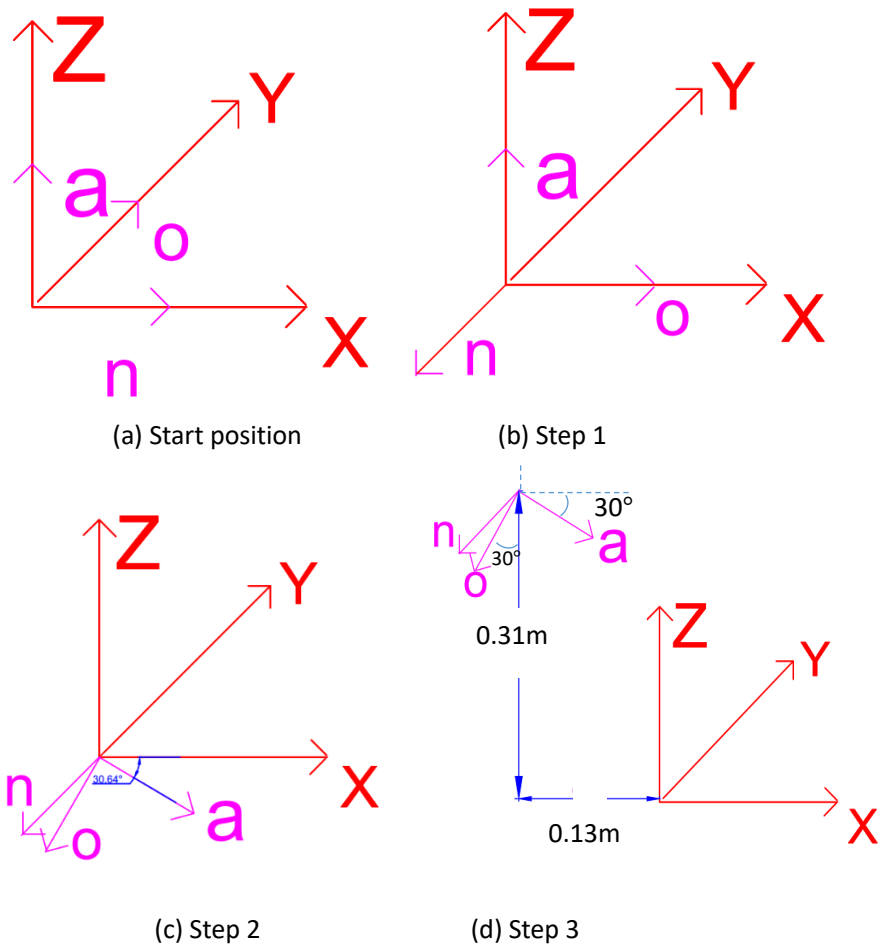


Figure 3-27 Coordinate transfer procedure

$$\mathrm{Rot}(z, 90°) = \begin{bmatrix} \cos(-90°) & -\sin(-90°) & 0 & 0 \\ \sin(-90°) & \cos(-90°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (12)$$

$$\mathrm{Rot}(y, 120°) = \begin{bmatrix} \cos(120°) & 0 & \sin(120°) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(120°) & 0 & \cos(120°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

$$\mathrm{Trans}(-0.13,0,0.31) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -0.13 & 0 & 0.31 & 1 \end{bmatrix} \quad (14)$$

So the final matrix equation representing the transformation is:

$P_{xyz} = \mathrm{Trans}(-0.13,0,0.31)\mathrm{Rot}(y, 120°)\mathrm{Rot}(z, 90°)P_{noa}=$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -0.13 & 0 & 0.31 & 1 \end{bmatrix} \times \begin{bmatrix} -0.5 & 0 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{3}}{2} & 0 & -0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix} \quad (15)$$

Finally we have

$$P_{xyz}^{T} = [\begin{bmatrix} \frac{\sqrt{3}}{2}c - \frac{b}{2} - 0.125 \\ 0.02 - \mathrm{a} \\ 0.27 - \frac{\sqrt{3}}{2}b - \frac{c}{2} \\ 1 \end{bmatrix}. \quad (16)$$

This would be the object's center location with respect to the robotic arm base. The second step uses robotic kinematics to calculate each joint's angle and is described in the following section 3.5.2.

3.5.2 Robotic Kinematics to Decide Each Arm Angle

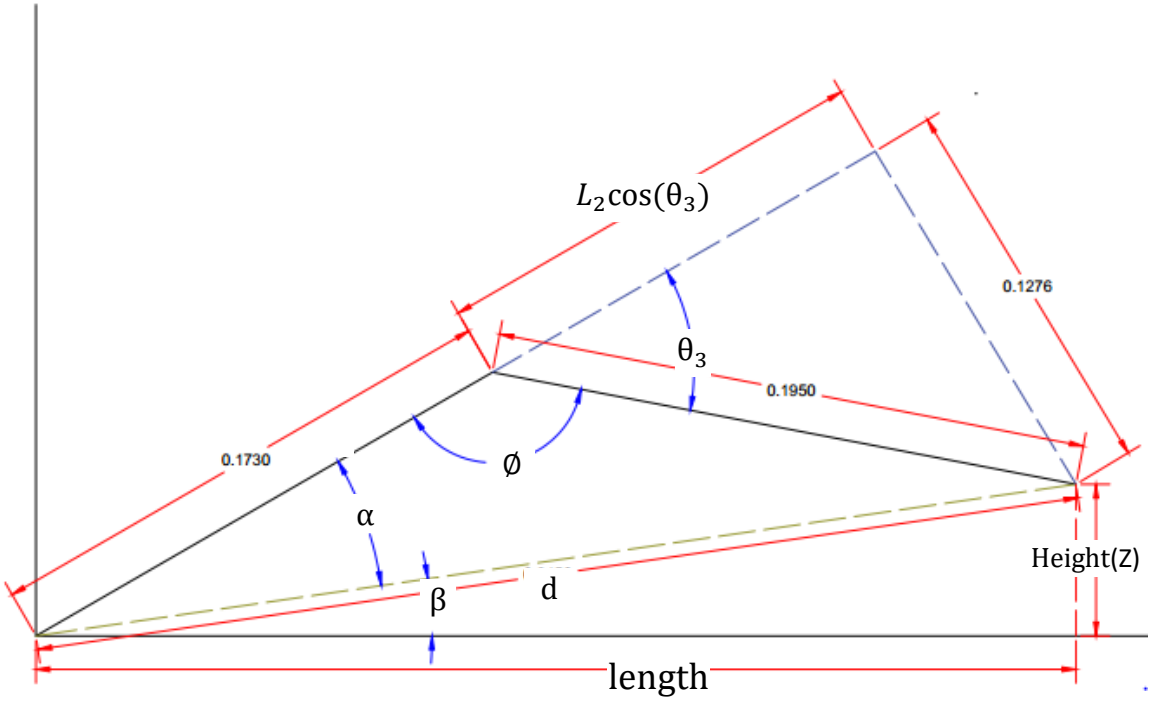
First of all, since the robotic arm only has four degrees of freedom, we can easily calculate the first joint's angle as

$$\theta_1 \arctan(x). \tag{17}$$

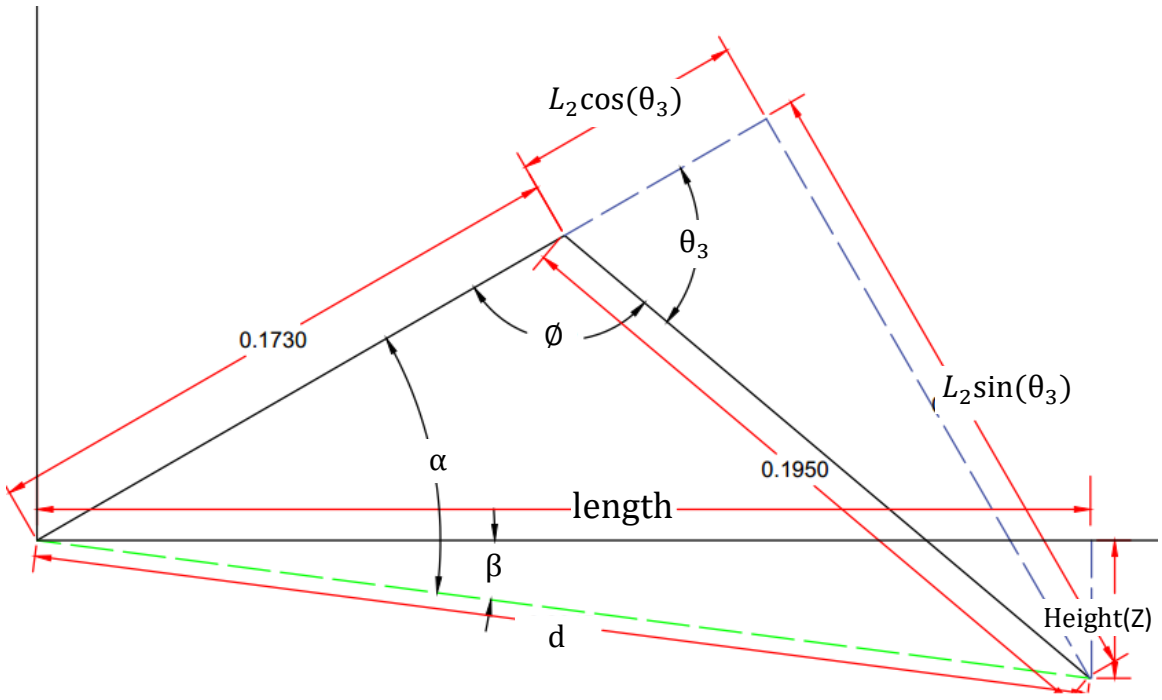
and θ_1 should satisfy:

$$-90° < \theta_1 < 90° \tag{18}$$

However，instead of using the coordinate transform method, solutions for θ_2 and θ_3 can be considered as a simple 2D mathematical problem as shown in Figure 3-28 (a) and (b) below.



(a) Situation 1



(b) Situation 2

Figure 3-28 Situations of the robotic arm position

Starting from Ø in the picture, the arm length between the 2nd and 3rd joint is L_1=0.173m, and the arm length between the 3rd and 4th joint is L_2 = 0.19 m. For "length" and "height," as indicated in Figure 3-28:

$$Length = \sqrt{x^2 + y^2} \quad Height = z \tag{19}$$

So we have:

$$\mathrm{d} = \sqrt{x^2 + y^2 + z^2} \tag{20}$$

Thus, we can calculate Ø as:

$$\cos(\emptyset) = \frac{{L_1}^2 + {L_2}^2 - d^2}{2L_1L_2} \tag{21}$$

which gives us:

$$\emptyset = \arccos\left(\frac{{L_1}^2+{L_2}^2-d^2}{2L_1L_2}\right). \quad (22)$$

Then we know:

$$\theta_3 = 180 - \emptyset. \quad (23)$$

Based on the obtained θ_3, we have the equation for α and β in Figure 3-28 as:

$$\tan(\alpha)\frac{L_1+L_2\cos(\theta_3)}{L_2\sin(\theta_3)} \quad (24)$$

and

$$\tan(\beta)\frac{z}{d} \quad (25)$$

which yields

$$\theta_2 = \alpha + \beta = \arctan\frac{L_1+L_2\cos(\theta_3)}{L_2\sin(\theta_3)} + \frac{z}{d} \quad (26)$$

However, in actuality, the servos for joint 2 have a default origin angle which is 30 degrees to x axis of the world coordinate. So, we have to consider this situation as Figure 3-28 (b) as shown here, so we get ${\theta_2}' = \theta_2 - 30°$.

And finally θ_4 will be determined according to the object's pose.

3.5.3 Finding the Grasping point

For objects like the flashlight, pill bottle and cup2, they are all cylinder-shaped objects, so the grasping point is easy to find and define because their center point has the largest z value (to prevent mistaking the bounding box's center for the object center) as shown in Figure 3-29 below:

Figure 3-29 Detection and finding grasping point for cylinder-shaped objects

For objects like a mug, it would be a little bit more difficult to grasp since the best grasping point would be on the handle and it is not easy to recognize. However, instead of training the network to recognize the handle which would be really time consuming, I just used the depth image to find the grasping point. Figure 3-30 below shows the depth image for a scene with a mug in it.

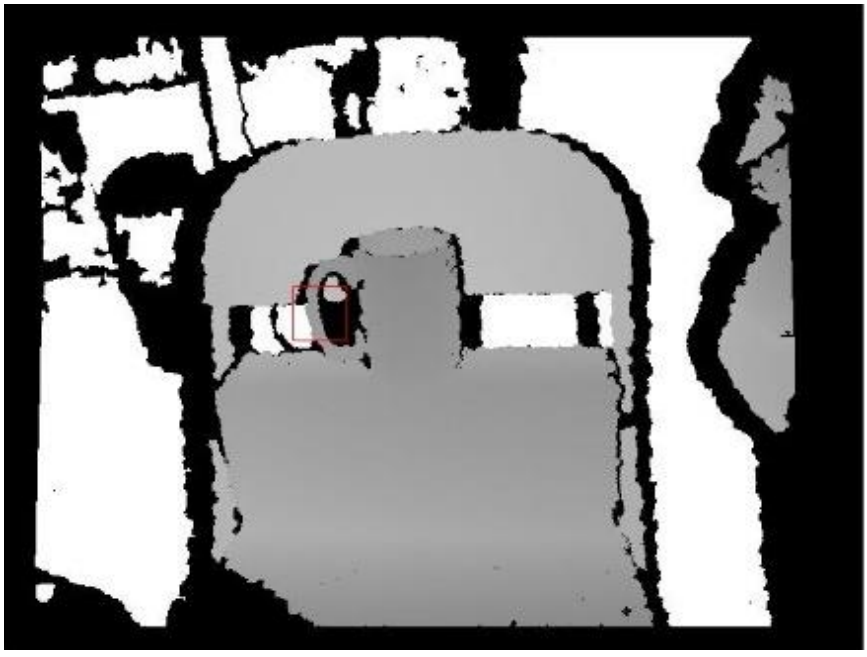


Figure 3-30 Depth scene with mug in it

Since Microsoft Kinect can only generate a depth image within a limited range (typically 40 cm–300 cm), and is not able to find the depth information for some shadowed areas, it will give an NaN value at those areas in the depth image (shown as black areas in Figure 3-31). The way to find the grasping point for a mug like this is to follow these steps:

1. Find out whether the handle is on the left or on the right: Due to the arm's reachable range, objects cannot be placed over 100 cm away from the Kinect. Suppose we have the center point (x, y) found by Faster R-CNN from previous steps. The bounding box has a width of "L". So, I created two vectors where

$$V1 = depth\left(\left(x - \frac{L}{2}\right) : x, y\right) \tag{28}$$

$$V2 = depth\left(x : \left(x + \frac{L}{2}\right), y\right) \tag{29}$$

   Since the depth value in the line vector that has a handle on that side would have a larger variance than the vector representing the side of the main body of the cup, I calculate the variance value of V1 and V2 without elements that have values larger than 300 cm. If the variance of V1 is larger than V2, the handle is usually on the left.

2. Suppose I found the variance of V1 to be smaller than V2, the handle would be on the right. Then I search for the minimum value in V3 where:

$$V3 = V2\left(\frac{L}{4} : end\right) = depth\left(\left(x + \frac{L}{4}\right) : \left(x + \frac{L}{2}\right), y\right) \tag{30}$$

   The minimum value represents where the handle is and gives the position of grasping point.

For an object like a bowl, I first search for the edge of the bowl at the area shown in Figure 3-31.

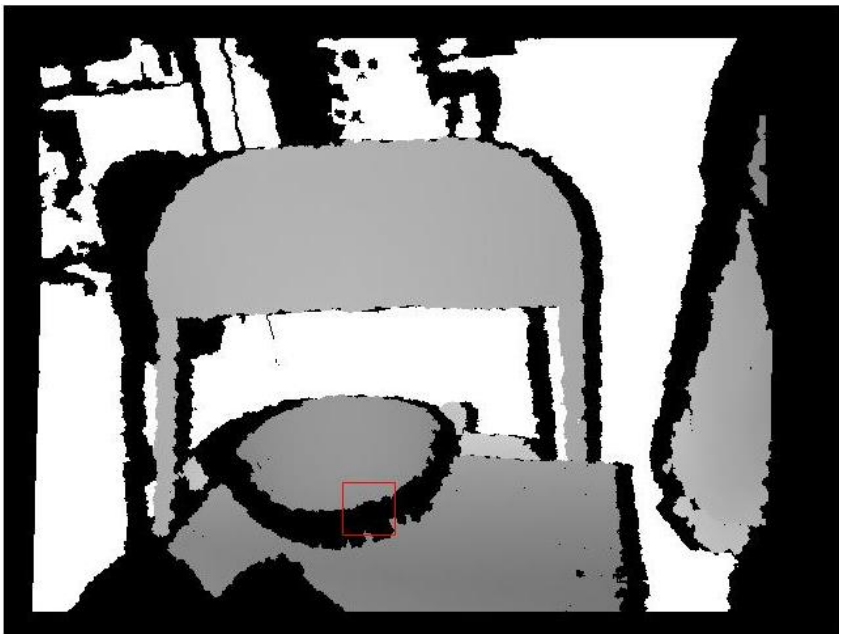


Figure 3-31 Depth scene with bowl in it

Using the depth image, suppose we have the bounding box height “H” and length “L”; we would then create a vector as

$$V4 = depth\left(\left(x - \frac{L}{2}\right) : \left(x + \frac{L}{2}\right), \left(y - \frac{H}{2}\right) : (y + \frac{H}{2})\right) \tag{31}$$

The next step is to search for the minimum value for each column in the vector and their location. This indicates the edge of the bowl. Since we already have the joint1 angle and horizontal distances, we can calculate the nearest point to the robotic arm base and set it as the grasping point.

3.5.4 Robotic Arm Path Planning

Since we've found the solution to calculate angles for the arm joints to make the gripper move to the desired place, we now need to consider the path planning problem.

The first problem we meet is related to the gripper itself. From Figure 3-32, we can see that the gripper is grasping a flash light and has already reached the position for grasping. However, the problem is how it can reach this position. Simply going straight from one position to this position would not be an option since the arm might sweep not only the flash light but also everything in its path. So, the first problem is how we can reach the desired position without touching any other objects on the table.

Figure 3-32 Robotic arm in grabbing position

To do this, we first set up an initial position as shown in Figure 3-33.

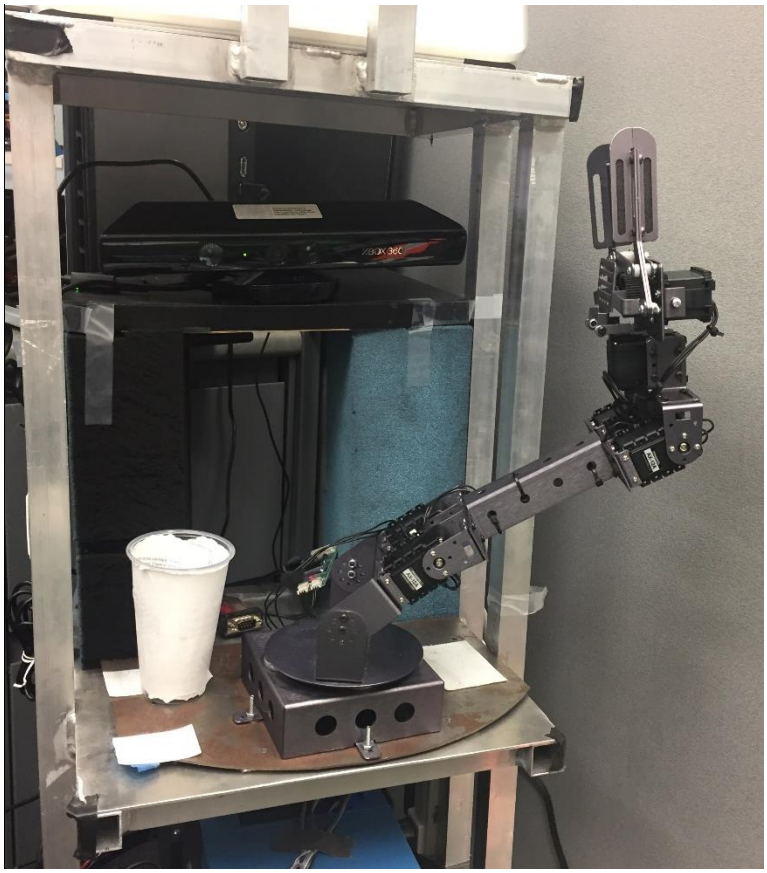


Figure 3-33 Robotic arm in initial position

There are two considerations for this initial position.

1. The arm would be easier to get back to this position, which does not make it difficult for the robot itself to move around: I've considered other poses as shown in figure 3-34. However, it would be more complicated and take more time for the arm to get to these positions after finishing grasping when tables or other obstacles are around the robot. Also, it would make it hard for the robot to move around since the arm might crash into other furniture. Thus, making the arm point up and stay in this area annotated by a red rectangle as shown in Figure 3-35 would be the best choice of all.

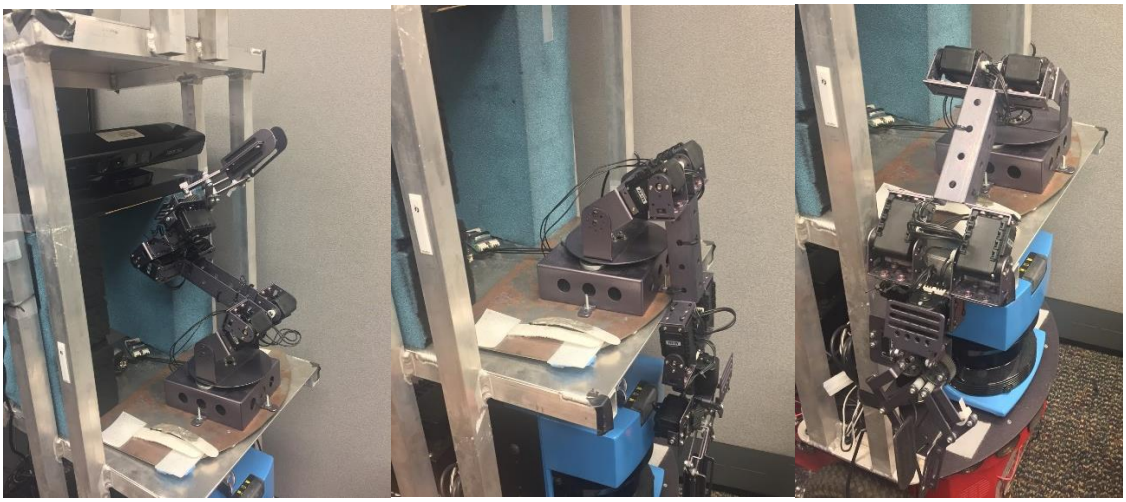


Figure 3-34 Different possible initial positions

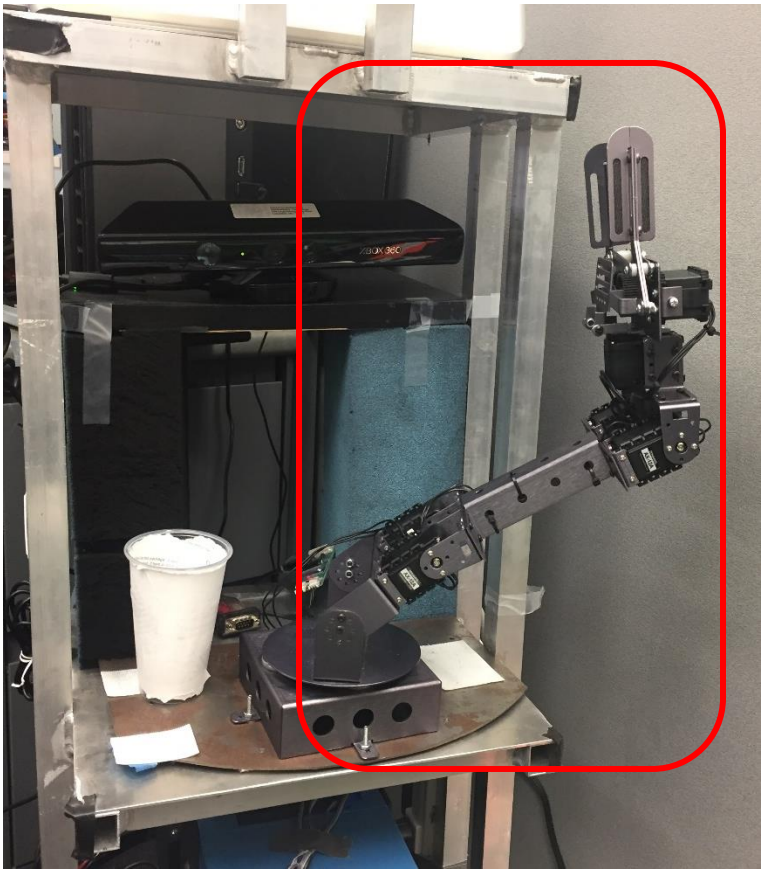


Figure 3-35 Initial pose and possible area

2. At this position, the arm cannot block the view of the Kinect: Among all the possibilities, only when the arm was at this area as shown in Figure 3-35 could the Kinect view not be blocked (The IR camera would be blocked if the arm is on the other side). So I just chose right side.

The next problem was determining how the arm would approach the object since a direct path might cause other items to get knocked off. Going from the top would solve this problem. Figure 3-36 shows a situation where this kind of robot would mostly be used to grasp objects.

Figure 3-36 Common situation with several objects on a table

Since it is only going to grasp objects from tables or similar places (shelves not included), there are no obstacles on top that will block the way of the arm. We tried to design the arm into a ready position and then lower it down directly to the grabbing position.

Another problem occurred when the robotic hand would sometimes get stuck due to obstruction of object(s) as shown in Figure 3-37.

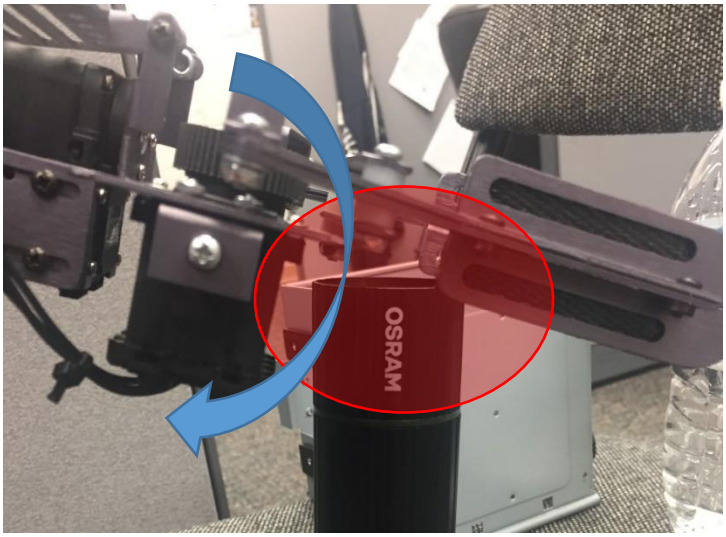


Figure 3-37 Problem when moving arm to the grasping position

In Figure 3-37, we can see that the red area highlighting the robotic hand status as it gets stuck on its way to the grasping position, and the blue arrow is indicating the direction in which the robotic gripper is moving. So lowering the arm directly from above might cause this area to crash on the top of the object especially with an object shape like that shown in Figure 3-38.

Figure 3-38 Cups that have a larger diameter at top than the bottom

So the arm cannot be lowered down directly from up above; thus, it needs to go somewhere first and then go to the grasping position. Figure 3-39 shows the position where we determined the arm should go to before going to the grasping position. This position would greatly resolve the obstruction problem. The arm goes from point A to B as indicated in Figure 3-339. Point A would be called the "Ready" position. In this position the horizontal distance between A and B is 5 cm, and the vertical distance between A and B is 3 cm.

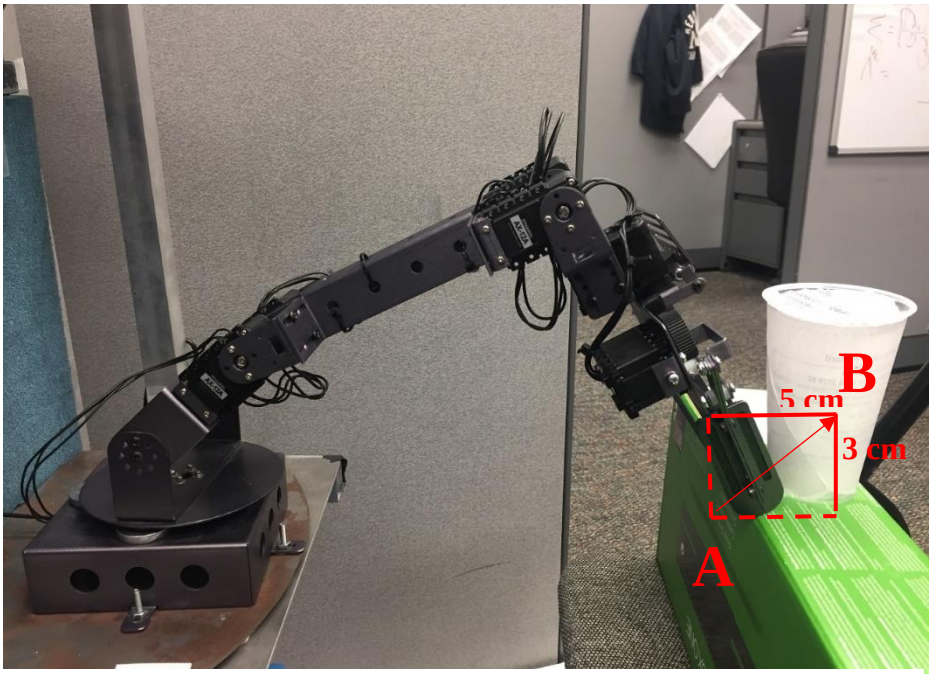


Figure 3-39 "Ready" position

However, there are some situations where the arm cannot reach the "Ready" position first. For example, the grasping position may be at the edge or out of the range that the robotic arm can reach. At this point, a decision must be made on whether it is possible to go directly to the grasping position by scanning the rough top of the object's 3D shape. If the shape has a diameter that is smaller than the robotic gripper's open width, then the

gripper can go directly to the grasping position. Otherwise we must give up trying to get the object and return to the initial position.

The ready position of a bowl would be different, since they are not reachable from below, I designed the robotic arm to first reach 2 cm right above the grasping point as shown in Figure 3-40 below and then drop right down to the grasping point and grab it.

However, since the robotic arm only has limited power, it could neither lift up the ceramic bowls used here nor the coffee mug. We evaluated if the grasp was a success by judging if the grabber closed tightly at the edge of the bowl or the handle of the coffee mug.

Figure 3-40 Ready position for grasping bowl

CHAPTER 4 RESULTS AND DISCUSSION

The performance of the system during training is our first topic in this chapter. The results from grasping actual objects in as many as 20 grasping tests for each class of objects are presented based on experimental data. The results are recorded as part of the grasping point recognition performance assessment. We will also look at time spent on detecting each object and determine the grasping success rate.

4.1 Faster R-CNN Performance for Object Detection in Data Sets

4.1.1 Training Performance

The training is separated into two stages as previously mentioned in Chapter 2. The first stage training involves RPN training followed by Fast RCNN training. I set the system to output the average foreground error rate, background error rate, cls layer loss function output and regression loss function output every 200 iterations. Figure 4-1, Figure 4-2, Figure 4-3, and Figure 4-4 below show the Training performance of a stage one RPN. We can see from Figure 4-1, during the first stage of RPN training that the network quickly reduced the foreground error rate from around 0.45 to around 0.15, which indicates that the network is using an appropriate cost function that can bring down the error rate quickly at the beginning and slow down when the error rate has dropped to a relatively low level [22]. This would save a lot of time when doing large data set training. We can also see that the validation error rate curve followed the

training error curve quite well, which indicates that the system is not overfitting [23] and is practical to general situations other than what is shown in the training data set.

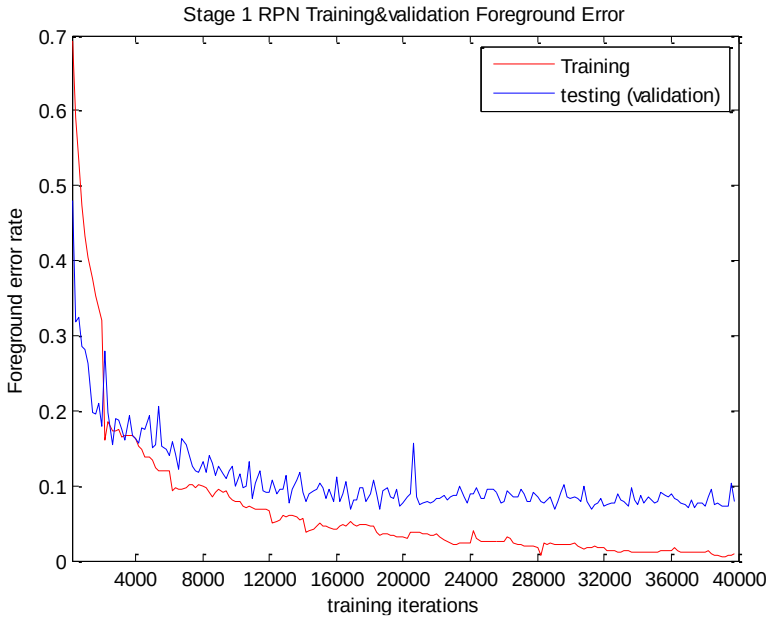


Figure 4-1 Stage 1 RPN training and validation foreground error curve

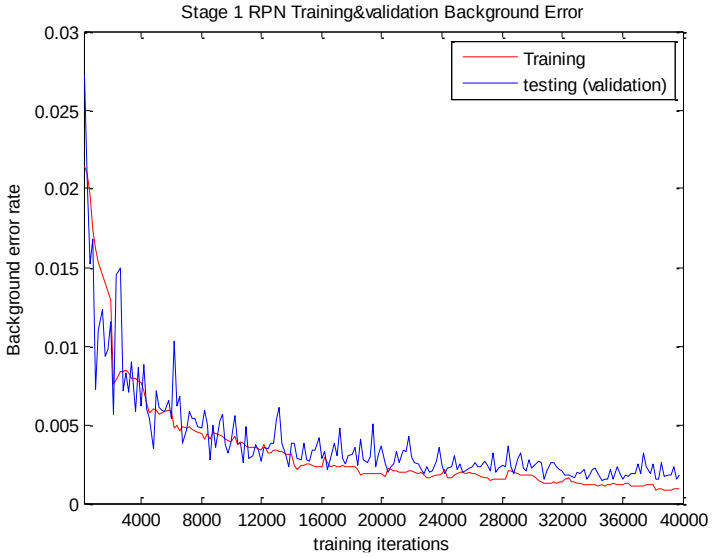


Figure 4-2 Stage 1 RPN training and validation background error curve

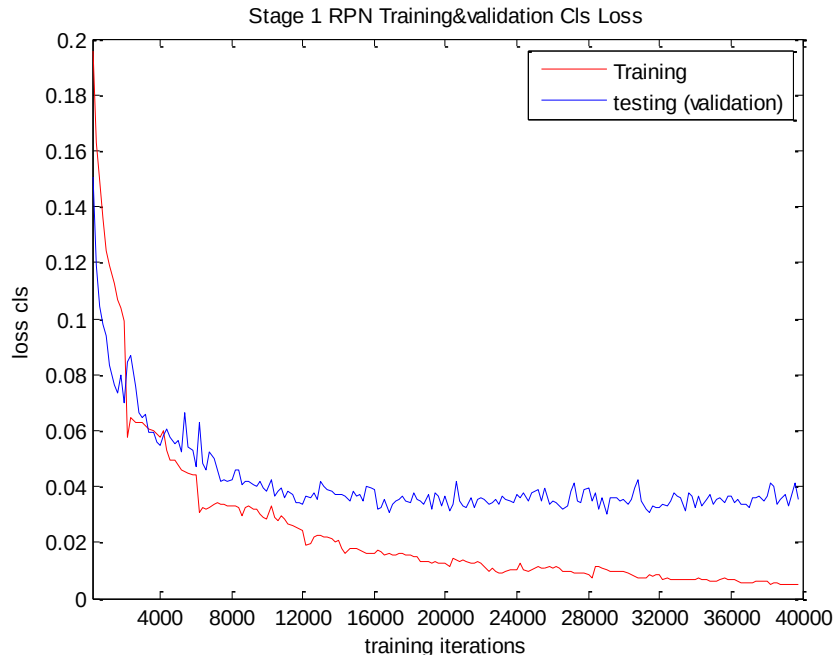


Figure 4-3 Stage 1 RPN training and validation cls loss curve

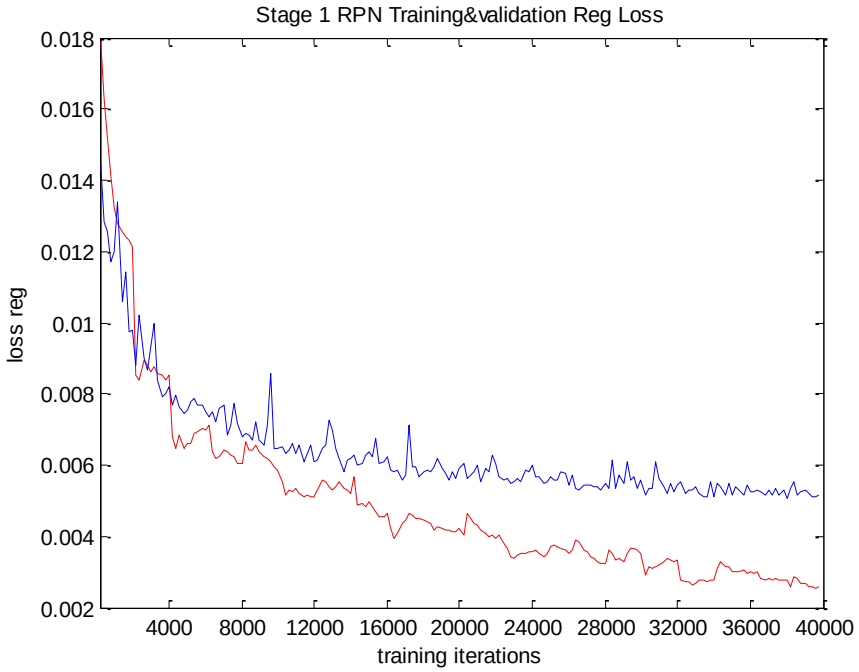


Figure 4-4 Stage 1 RPN training and validation regression error curve

The training and validation error, cls loss, and regression loss are shown in Figure 4-5, Figure 4-6 and Figure 4-7 below. From Figure 4-5, "Stage 1 Fast RCNN Training and validation error," we can see that the RPN network did a great job making proposals which makes the training errors of the Fast RCNN start at around 0.2, and the validation results indicate that there are no overfitting problems.

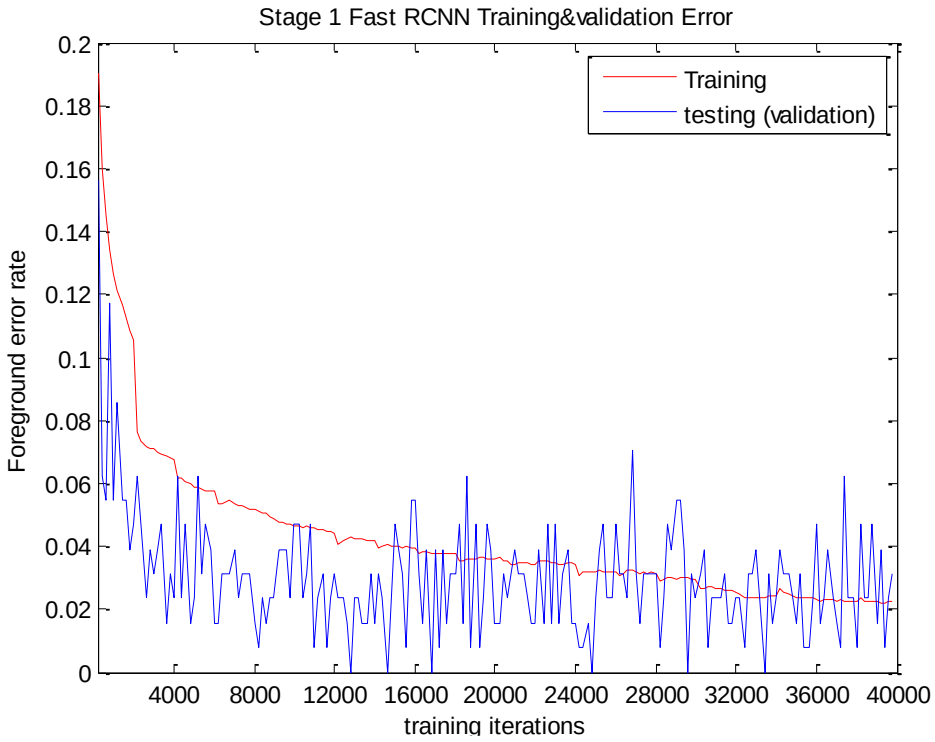


Figure 4-5 Stage 1 Fast RCNN training and validation error

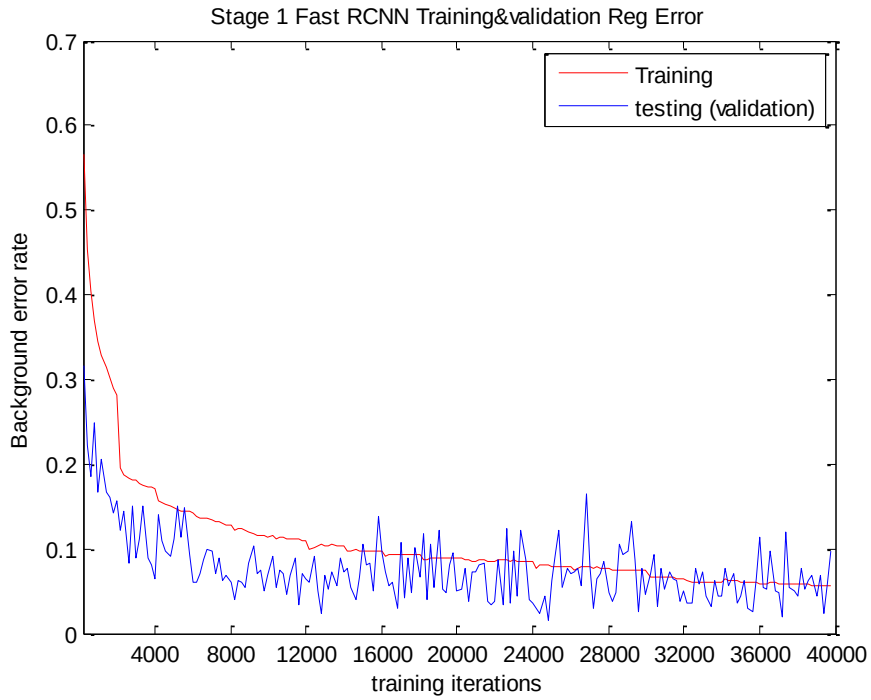


Figure 4-6 Stage 1 Fast RCNN training and validation regression error

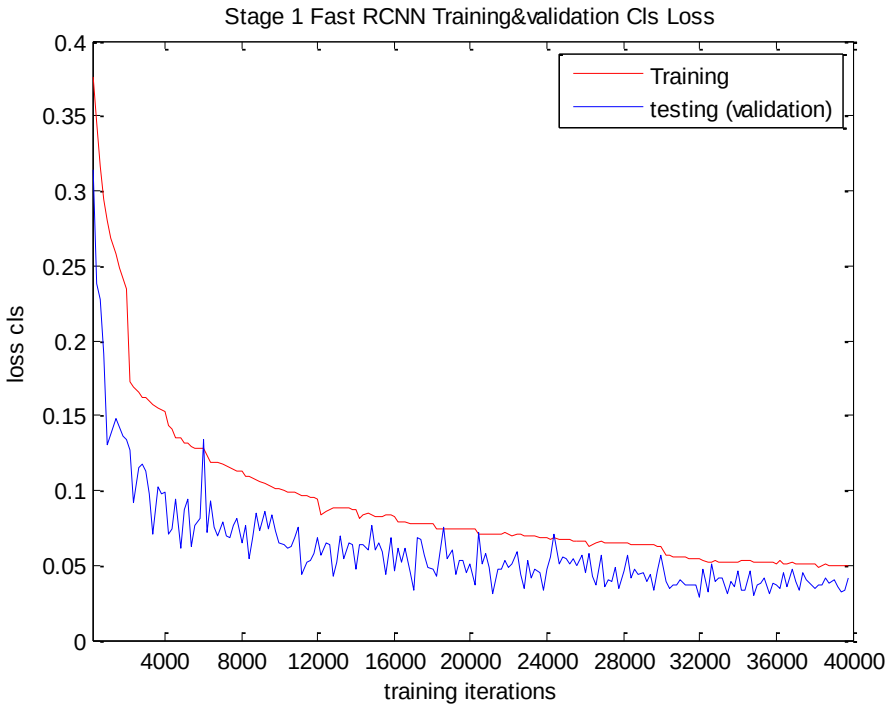


Figure 4-7 Stage 1 Fast RCNN cls loss

After stage 1 training on both RPN and the Fast RCNN network, the parameters for the whole system were updated accordingly. Then, stage 2 training started from RPN using these new parameters. We can see that the foreground error during training and validation started from a relatively low level compared to stage 1 RPN training, which indicates that the network did get better after stage 1 training.

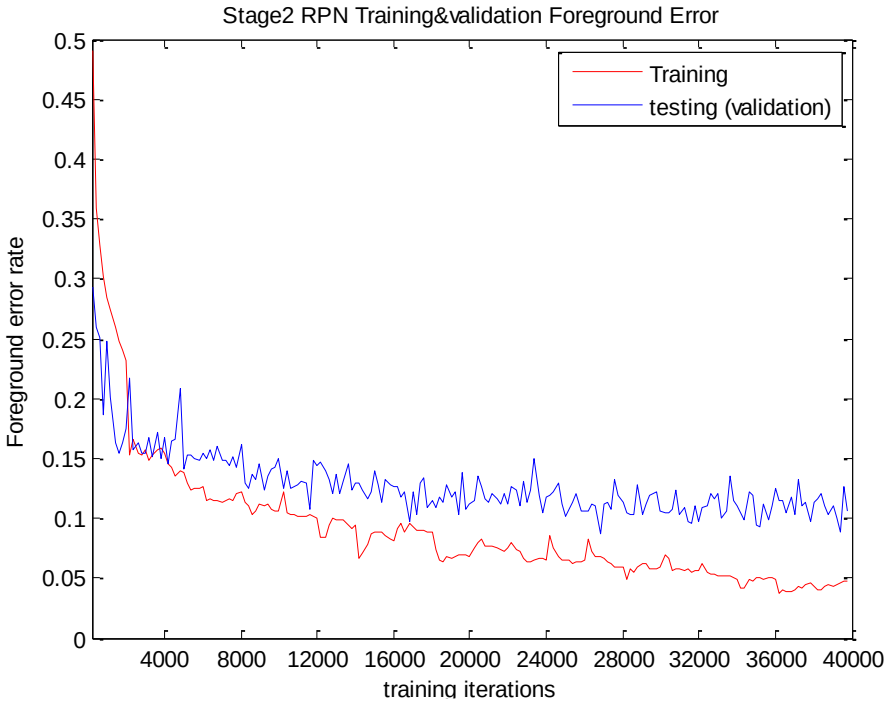


Figure 4-8 Stage 2 RPN training and validation error for foreground

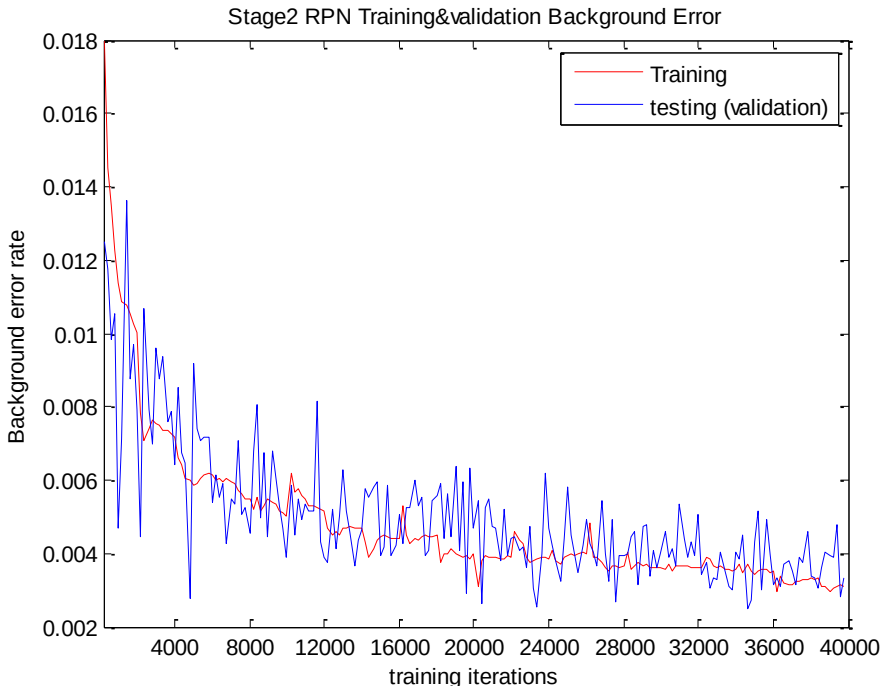


Figure 4-9 Stage 2 RPN training and validation curve for background error

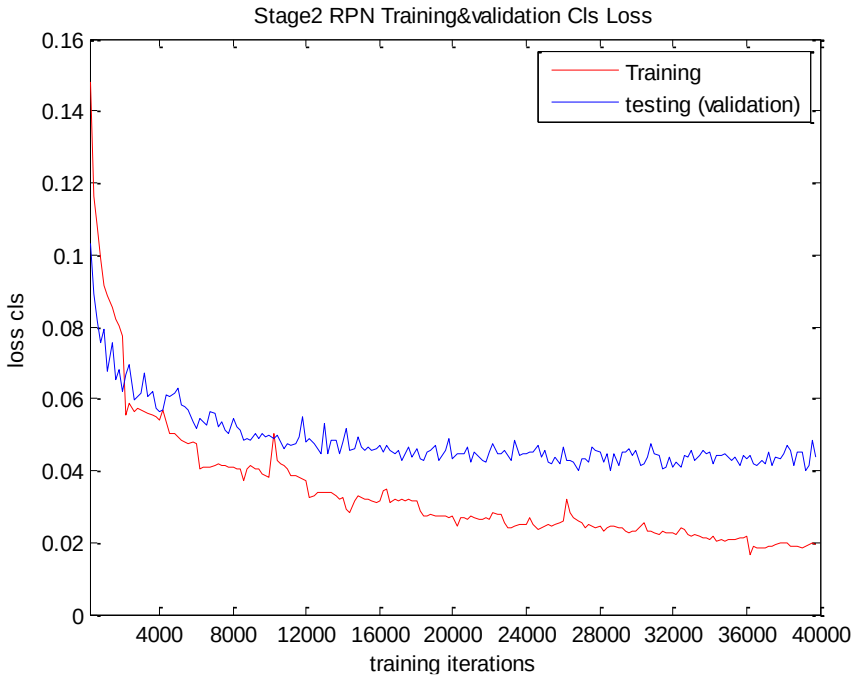


Figure 4-10 Stage 2 RPN training and validation cls loss curve

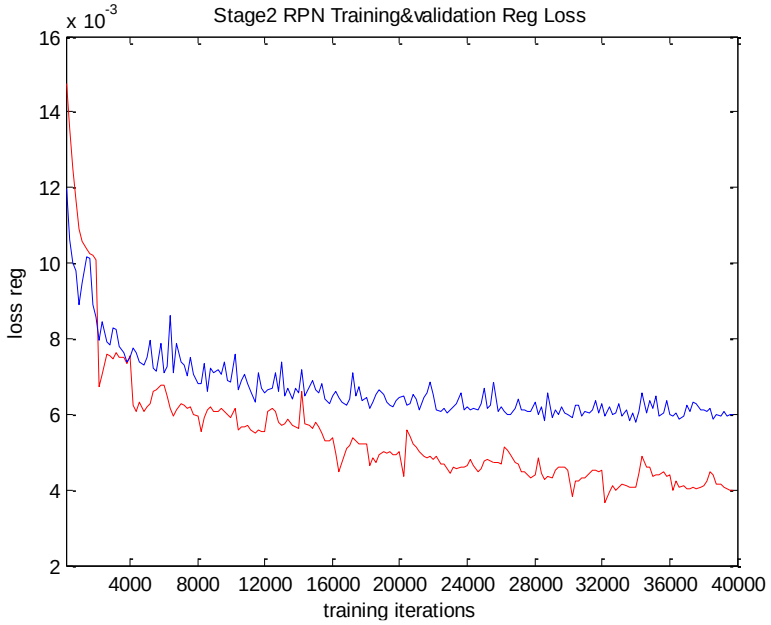


Figure 4-11 Stage 2 RPN training and validation reg. loss curve

Below are the results for stage 2 Fast RCNN training and validation:

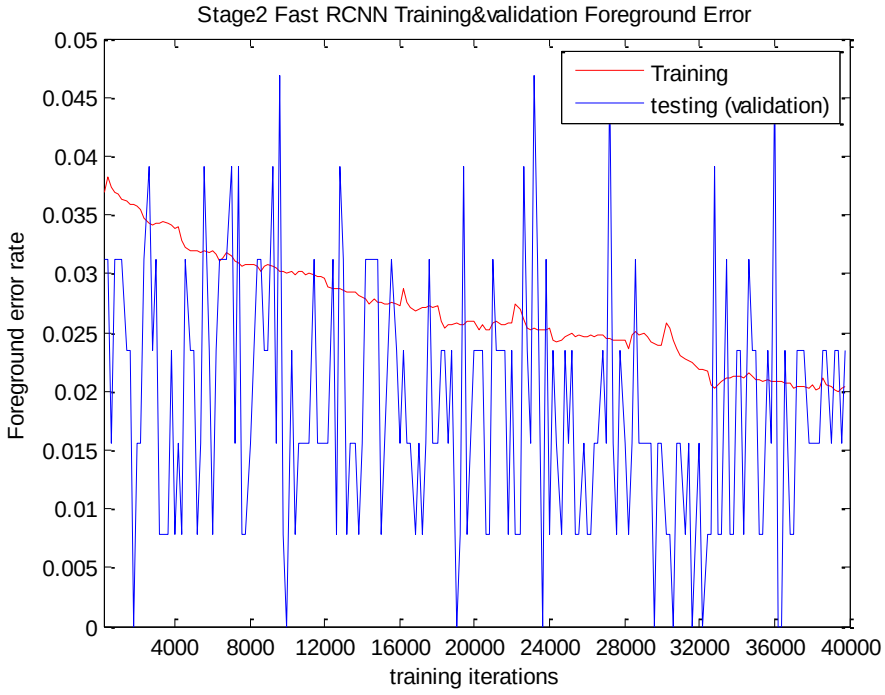


Figure 4-12 Stage 2 Fast RCNN training and validation error

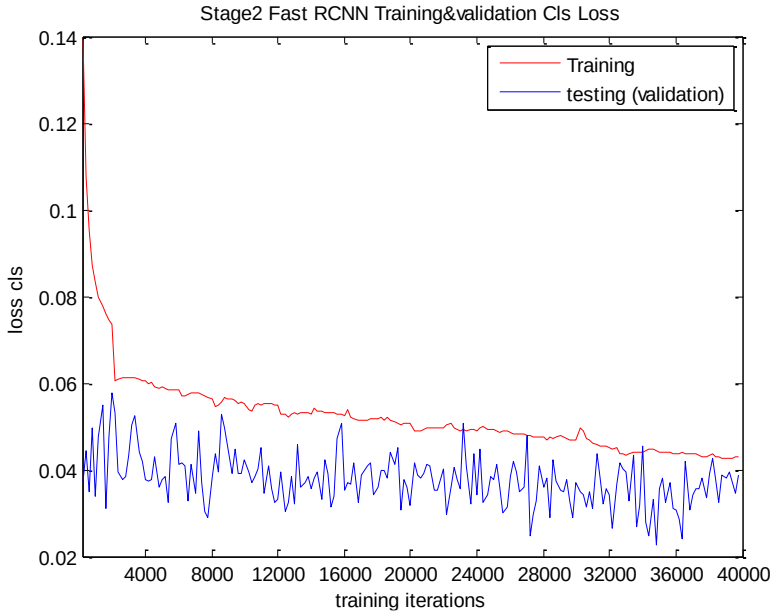


Figure 4-13 Stage 2 Fast RCNN training and validation loss

Figure 4-14 Stage 2 Fast RCNN training and validation reg. loss

4.1.2 Testing Performance

Since there are five classes of objects. I used the precision-recall curve to evaluate the test results for the first stage Fast RCNN detection testing and the stage 2 Fast RCNN detection testing. Since the results are surprisingly good, I set the figures to show at a range of [0, 1.01] on both x and y axis. The results are excellent as shown in Figures 4-15, 4-16, 4-17, and 4-18. We can see that the stage 1 training was very successful, and stage 2 training can only slightly adjust the whole model since the results are already good enough. In the figure, as the recall rate approaches 1.0, the precision value holds quite steady until it starts to decrease drastically somewhere around the 0.9 recall rate, which means the whole model after training is extremely work reliable.

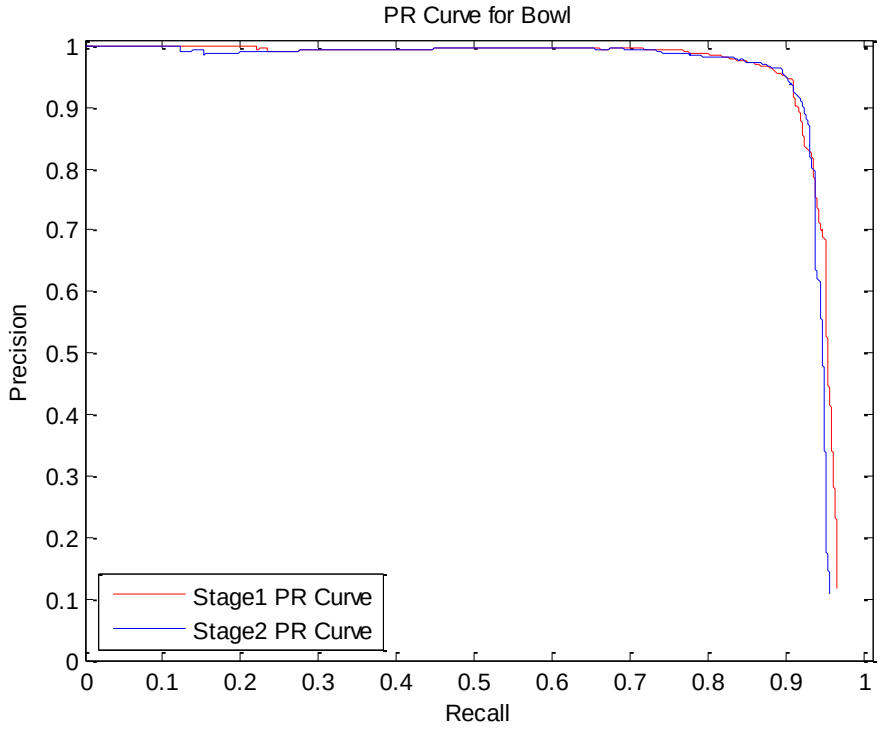


Figure 4-15 PR curve for bowl

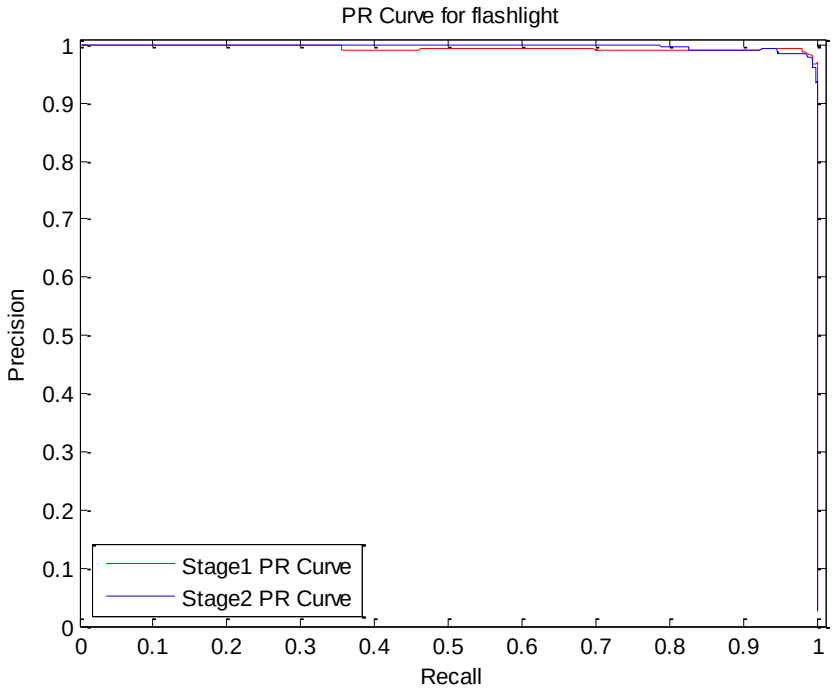


Figure 4-16 PR curve for flashlight

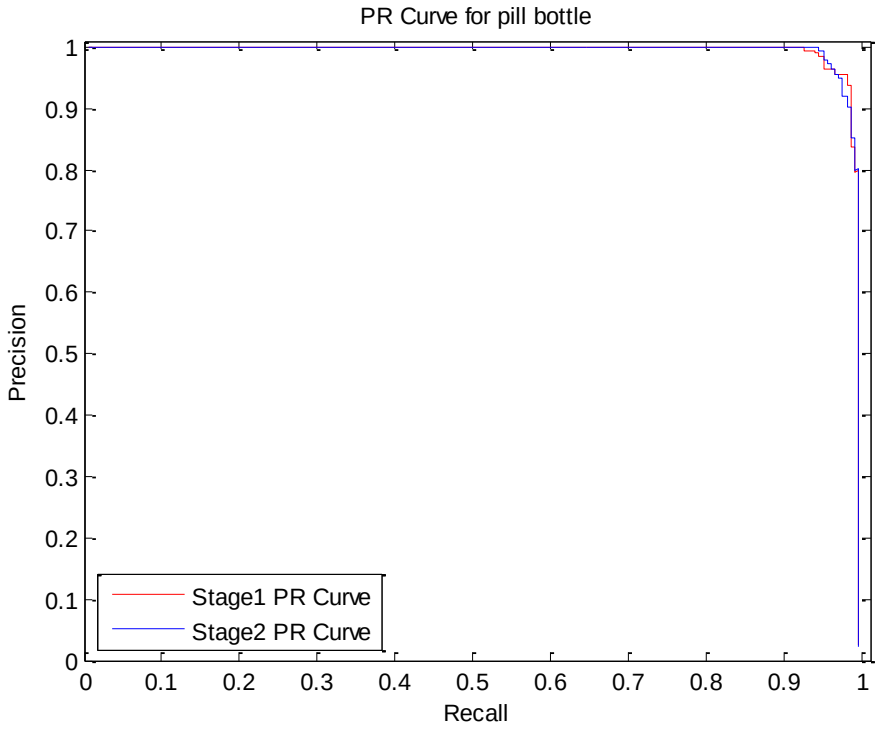


Figure 4-17 PR Curve for pill bottle

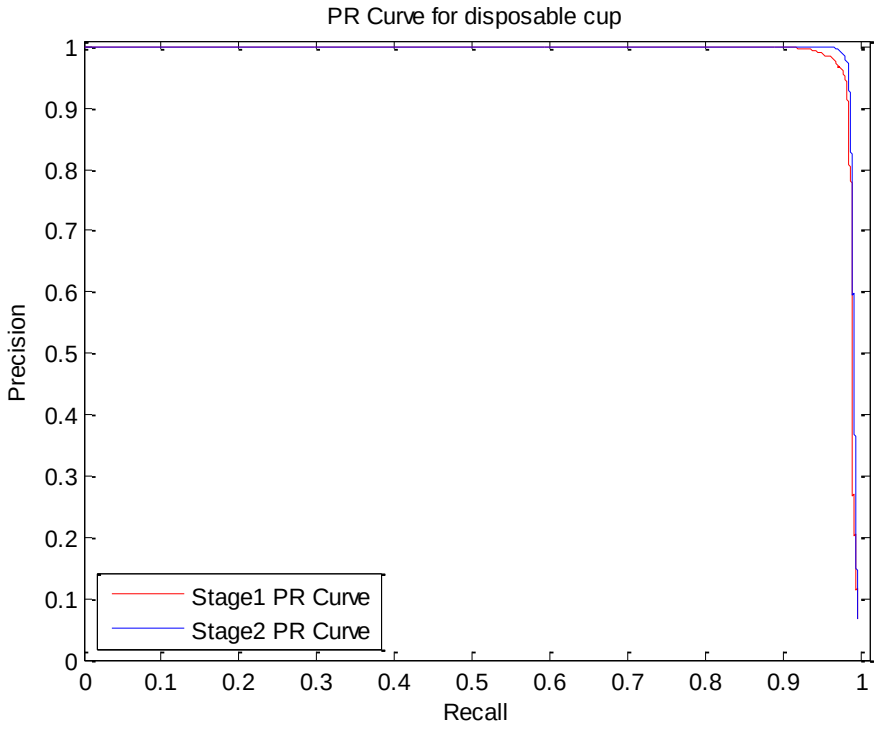


Figure 4-18 PR Curve for disposable cup

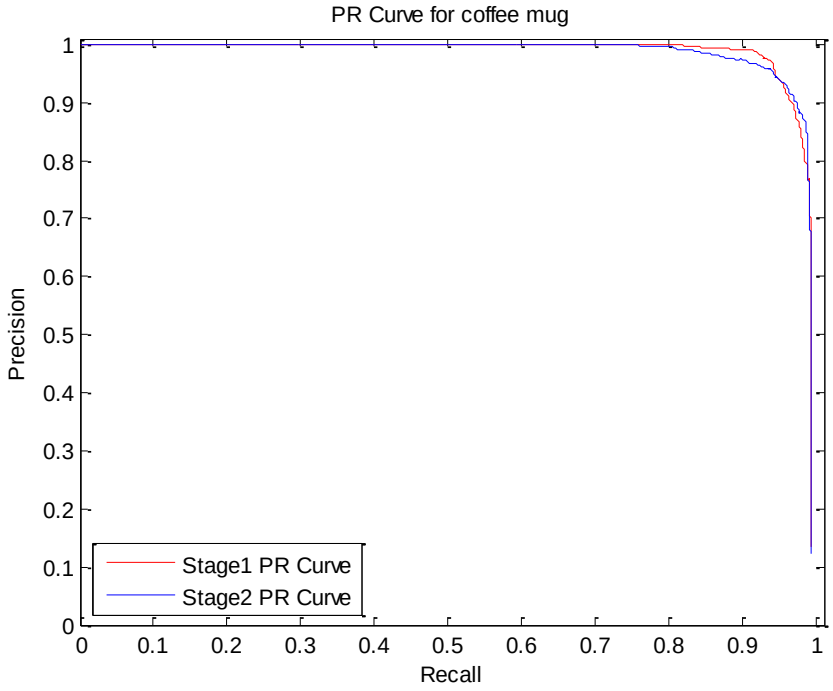


Figure 4-19 PR Curve for coffee mug

Sudeep Pillai and John J. Leonard also did an experiment in recognizing the bowls and cups using the deep learning method as mentioned in Chapter 2. Their results are shown in Table 4-1 [10], with comparison to my thesis research results shown in Table 4-2.

Table 4-1 Precision and Recall Comparison with Sudeep Pillai's work [10]

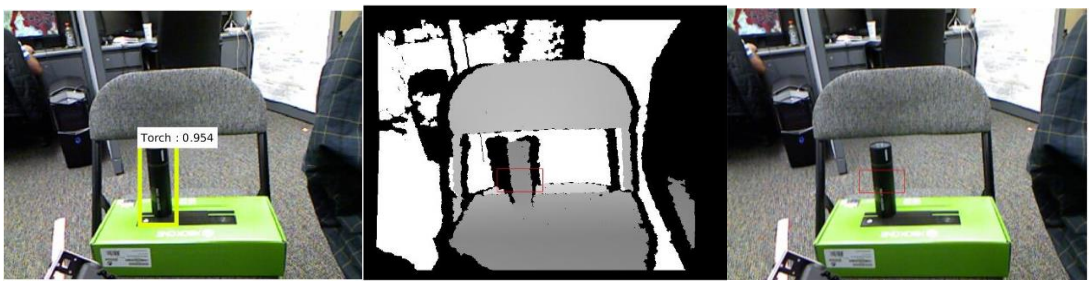
	Precision/Recall(Single View)		Precision/Recall(Multiple View)
	Sudeep	Mine	Sudeep
Coffee Mug	70.8/50.8	100/64.1	80.1/64.1
Bowl	88.6/71.6	99.55/71.6	88.7/70.2

Table 4-2 Precision and Recall from my Experiments

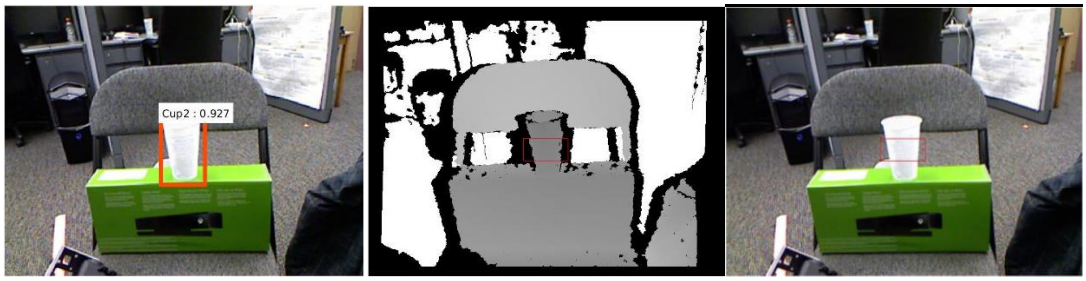
	Precision/Recall(Single View)	Average Precision(AP)
Coffee Mug	96.90/90.82	90.64%
Bowl	94.59/90.02	90.06%
Flashlight	97.86/98.57	99.31%
Pill Bottle	97.42/97.47	90.91%
Disposable Cup	96.60/98.08	90.91%
Mean Average Precision(mAP)		92.37%

4.1.3 Grasp Point Recognition Performance

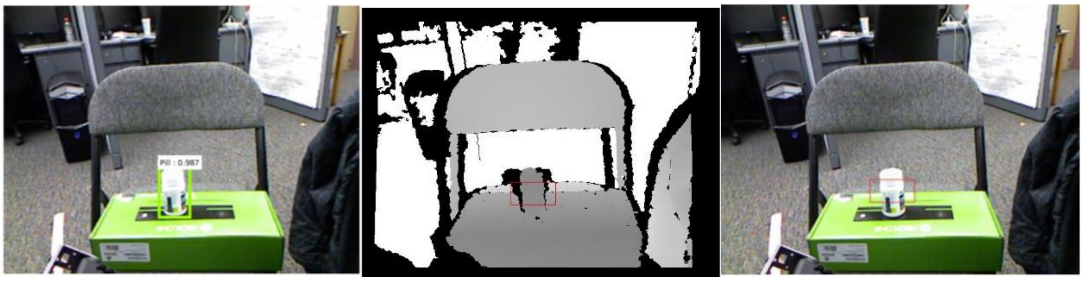
Since my algorithm first located the object location quite accurately, finding the grasping point has become easier and less time consuming. Figure 4-20 (a)–(e) below shows how the object was recognized, and then the grasping point was found (centered at the red box on the image). I also labeled the grasping point on the RGB image to make it easier to understand. I picked one example of grasping point recognition to show in Figure 4-20 for each class of object.



(a) Example of recognizing flashlight and corresponding grasp point

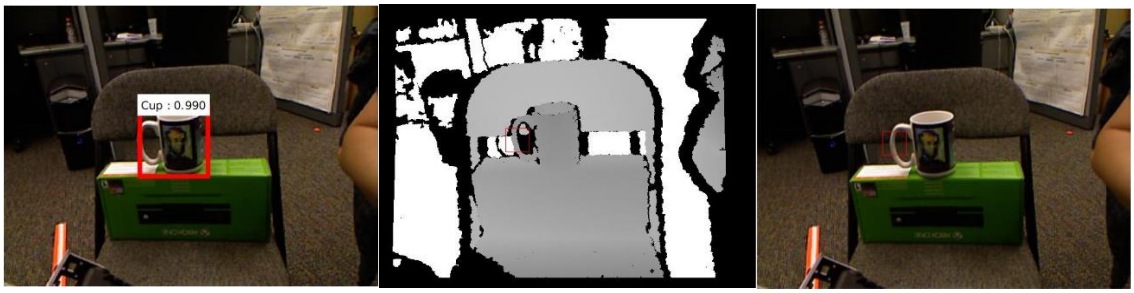


(b) Example of recognizing disposable cup and corresponding grasp point



(c) Example of recognizing pill bottle and corresponding grasp point

(d) Example of recognizing bowl and corresponding grasp point



(e) Example of recognizing coffee mug and corresponding grasp point

Figure 4-20 Grasp point recognition example for each class

Table 4-3 shows the grasp point recognition performance during 20 exercises wherein robotic grasp performance was reported for each object. We can see that the algorithm performs well when dealing with these objects, especially when trying to find the grasp point on the flashlight, pill bottle and disposable cup. Though it failed to recognize the edge of the bowl and handle of the coffee mug a few times due to the inaccuracy of Kinect, as a first-generation version, some wrong values can be reported at some points, which sometimes occurred during my experiments. Also, since Kinect generates depth image based on its IR camera, it may get some null values when the IR does not receive the reflection from desired points.

Table 4-3 Grasp Point Recognition Performance Based on 20 Experiments

Object Class	Success Rate (times)	Fail Rate (times)	Success Percent
Bowl	18	2	90%
Coffee Mug	18	2	90%
Disposable Cup	20	0	100%
Flashlight	20	0	100%
Pill Bottle	20	0	100%
Total	96	4	96%

4.1.4 Total Time Performance

The total time spent for detection and finding different grasp points is shown in Table 4-4 and Table 4-5, which indicate good system performance even though I used the NVidia graphic card GTX 970 with 4G memory, which can only be considered a fairly common graphic card for the algorithm compared to the K40 graphic card which has 12 G of memory.

The time spent on detection takes up most of the time spent totally. However, it is a great improvement on speed and stability compared to Ian Lenz's work which takes 13.5s/image [3] and Ben Kehoe's work [25] which takes from 1.4–10.6 seconds to recognize the grasp point and pose for the object, the unstable and slow performance of their system is due to the using of CPU that come with the robot. Instead, I used GPU that is capable of parallel computation to perform object detection which can be much faster than CPU. The total time for my system to detect the target object and estimate the grasp point was in the range of [0.074, 0.263] seconds.

Table 4-4 Time Performance for Object Detection

Object Class	Min Time(s)	Max Time(s)	Average Time(s)
Bowl	0.079	0.243	0.124
Coffee Mug	0.082	0.258	0.131
Disposable Cup	0.074	0.250	0.144
Flashlight	0.083	0.263	0.145
Pill Bottle	0.081	0.255	0.132
Total	0.074	0.263	0.135

Table 4-5 Time Performance for Grasping Point Estimation

Object Class	Min Time(ms)	Max Time(ms)	Average Time(ms)
Bowl	0.128	0.255	0.156
Coffee Mug	0.102	0.434	0.217
Disposable Cup	0.096	0.189	0.117
Flashlight	0.065	0.082	0.073
Pill Bottle	0.073	0.142	0.097
Total	0.065	0.434	0.134

4.1.5 Grasping Accuracy Performance

For each class of object, I performed 20 robot grasping experiments. The objects were randomly placed within a reachable range of the robotic arm, with dynamic backgrounds.

Since the grabber of the robotic arm can only expand 5.5 cm, it requires great accuracy when trying to pick up objects. For example, the flashlight I used as an object had a diameter of 4.7 cm, so the error rate should be less than (5.5–4.7)/2=0.4 cm=4 mm or the grasp might fail. The disposable cup is the hardest to pick up since it has the widest diameter. However, the bowl and coffee mug have a much thinner grasp area which may give more tolerance to the grasp error.

Table 4-6 Absolute Grasping Error

Object Class	Absolute Min Error Rate(mm)	Absolute Max Error Rate(mm)	Average Absolute Error Rate(mm)
Bowl	0	6	1.72
Coffee Mug	0	5	1.89
Disposable Cup	0	5	1.41
Flashlight	0	4	1.37
Pill Bottle	0	3	1.54
Total	0	6	1.59

The grasping success rate is shown in Table 4-7. Due to the limitation of the open width of the grabber, the robotic arm failed to pick up the object even though the grasping point was recognized correctly. It sometimes crashed onto the edge of the object which was considered a failure of grasping.

Compared to Ashutosh Saxena's similar work [6] in grasping similar objects, the system has made significant improvements in increasing the accuracy though it achieved lower grasp-success rate which is 87% compared to 90% from Saxena's experiment.

However, considering they used a wider open grabber, the grasp-success rate of my experiment might be increased if using the same gripper as theirs.

Table 4-7 Grasping Success Rate

Object Class	Success Rate (times)	Fail Rate (times)	Success Percent
Bowl	17	3	85%
Coffee Mug	18	2	90%
Disposable Cup	16	4	80%
Flashlight	18	2	90%
Pill Bottle	18	2	90%
Total	87	13	87%

Table 4-8 Ashutosh Saxena's Experiment Results [6]

Object	Mean Absolute Error(mm)	Grasp-Success-Rate
Mugs	24	75%
Pens	9	100%
Wine Glass	12	100%
Cell Phone	16	100%
Books	29	75%
Overall	18	90%

CHAPTER 5 CONCLUSIONS

I developed a fast and accurate robotic grasp method using the deep learning method of Faster RCNN. The whole system was built and tested on the robotic operation system (ROS). The special communication mechanism of ROS made it possible for our system to use multiple packages from different languages and developed environments. Since ROS is designed as a distributed system, we were able to assign a desktop with NVidia GTX970 and Intel i7-930 as the processing server and a laptop as the control server to receive image and depth information and to control the robot. By using a trained Faster RCNN which was trained on a combined data set of RGB images containing five classes of objects, I obtained fast and accurate object detection results. Then, using the accurate bounding box given by the system, I could quickly find the right grasp point for each object based on the depth image within the bounding box. This algorithm greatly reduced the computing time compared to previous work, which input the entire RGB-D image into the network to find the correct grasp point. To summarize, this graduate research has achieved the following improvement on robotic recognition and grasp.

1. The time spent on finding the right grasping point was reduced to an average of 0.135 second which is much faster than previous work which takes 1.4 to 10.6 seconds.

2. The system was tested with an inexpensive robotic arm with 4 degree of freedom (costing around 300 U.S.dollars) and a gripper which expands to 5.5 cm.

3. The average absolute error rate when grasping is quite low averaging 1.59 mm compared to 18 mm from previous work [6], which used a much more expensive robot with over 5 degrees of freedom and a larger gripper width.

4. The grasping performance is also quite good at 87% since several failures could be avoided by changing the gripper to a wider one such as that used in previous work.

Some improvements remain for future work. The data set I used was limited in size and variety. To make the robot capable of recognizing more objects and able to handle novel objects, a larger data set with many more classes of objects is needed to train the network.

The robotic arm used in this research lacked the degrees of freedom, which greatly restricted the capability of picking up objects where at least 6 degrees of freedom was required for a robotic arm so that it could handle more complicated tasks and act more like a human arm.

REFERENCES

[1] Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., & Abbeel, P. (2010). Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. 2010 IEEE International Conference on Robotics and Automation. doi:10.1109/robot.2010.5509439

[2] D. Kragic and H. I. Christensen. Robust visual servoing. IJRR, 2003.

[3] Lenz, I., Lee, H., & Saxena, A. (2013). Deep Learning for Detecting Robotic Grasps. Robotics: Science and Systems IX. doi:10.15607/rss.2013.ix.012

[4] Szedegy, Christian, Alexander Toshev, and Dumitru Erhan. "Deep Neural Networks for Object Detection." Https://pdfs.semanticscholar.org/. Google, Inc.

[5] Bezak, P., Bozek, P., & Nikitin, Y. (2014). Advanced Robotic Grasping System Using Deep Learning. Procedia Engineering, 96, 10-20. doi:10.1016/j.proeng.2014.12.092

[6] Saxena, A., Driemeyer, J., & Ng, A. Y. (2008). Robotic Grasping of Novel Objects using Vision. The International Journal of Robotics Research, 27(2), 157-173. doi:10.1177/0278364907087172

[7] Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1-1. doi:10.1109/tpami.2016.2577031

[8] Hosang, J., Benenson, R., Dollar, P., & Schiele, B. (2016). What Makes for Effective Detection Proposals? IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(4), 814-830. doi:10.1109/tpami.2015.2465908

[9] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.2014.81

[10] Pillai, S., & Leonard, J. (2015). Monocular SLAM Supported Object Recognition. Robotics: Science and Systems XI. doi:10.15607/rss.2015.xi.034

[11] Girshick, R. (2015). Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV). doi:10.1109/iccv.2015.169

[12] Uijlings, J. R., K. E. A. Van De Sande, Gevers, T., & Smeulders, A. W. (2013). Selective Search for Object Recognition. International Journal of Computer Vision, 104(2), 154-171. doi:10.1007/s11263-013-0620-5

[13] J. Ponce, D. Stam, and B. Faverjon. On computing two-finger force-closure grasps of curved 2D objects. IJRR, 12(3):263, 1993.

[14] Gu, Yue. "[置顶] ROS 探索总结（一）--ROS 简介." ROS 探索总结（一）--ROS 简介 - 古月居 - 博客频道 (a summary of ROS part 1- introduction of ROS) - CSDN.NET. Retrieved December 14, 2016, from http://blog.csdn.net/hcx25909/article/details/8795043

[15] ROS.org | Powering the world's robots. (n.d.). Retrieved December 06, 2016, from http://www.ros.org/

[16] DYNAMIXEL AX-12A. (n.d.). Retrieved December 06, 2016, from http://www.robotis.us/ax-12a/

[17] Sunshineatnoon. How to train fast rcnn on imagenet. Retrieved December 06, 2016, from http://sunshineatnoon.github.io/Train-fast-rcnn-model-on-imagenet-without-matlab/

[18] Tan, Yuanyuan. Faster-rcnn 之 RPN 网络的结构解析 (details of RPN network structure from Faster R-CNN). Retrieved December 06, 2016, from

http://blog.csdn.net/sloanqin/article/details/51545125

[19] Van, R. C. J. (1979). Information retrieval. London: Butterworths.

[20] "Wiki." Openni_launch/Tutorials/QuickStart - ROS Wiki.

[21] "Wiki." ROS/Tutorials/MultipleMachines - ROS Wiki.

[22] Nielsen, Michael. " Neural Networks and Deep Learning CHAPTER 3." Neural Networks and Deep Learning. Jan. 2016.

[23] Hawkins, D. M. (2004). The Problem of Overfitting. ChemInform, 35(19). doi: 10.1002/chin.200419274

[24] Shen, Xiaolu. "【目标检测】Faster RCNN 算法详解." (Object Detection- Details of Faster R-CNN algorithm) Shenxiaolu1984 的专栏 - 博客频道 - CSDN.NET. N.p. Retrieved December 14, 2016, from http://blog.csdn.net/shenxiaolu1984/article/details/51152614

[25] Kehoe, B., Matsukawa, A., Candido, S., Kuffner, J., & Goldberg, K. (2013). Cloud-based robot grasping with the google object recognition engine. 2013 IEEE International Conference on Robotics and Automation. doi:10.1109/icra.2013.6631180

[26] D. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 2004.

[27] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In CVPR, 2005.

[28] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. IJCV, 2010

[29] Forsyth, D. (2014). Object Detection with Discriminatively Trained Part-Based Models. Computer, 47(2), 6-7. doi:10.1109/mc.2014.42

[30] Zhang, Jianguo., Schmid, Cordelia., Lazebnik, Svetlana., Ponce, Jean. (2006). VOC2006 QMUL description of LSPCH in sec 2.16 of The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results.

[31] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.

[32] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and L. FeiFei. ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012). http://www.image-net.org/challenges/LSVRC/2012/

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. FeiFei. ImageNet: A large-scale hierarchical image database. In CVPR, 2009.

[34] Pelossof, R., Miller, A., Allen, P., & Jebara, T. (2004). An SVM learning approach to robotic grasping. IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004. doi:10.1109/robot.2004.1308797

[35] Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2007). LabelMe: A Database and Web-Based Tool for Image Annotation. International Journal of Computer Vision, 77(1-3), 157-173. doi:10.1007/s11263-007-0090-8

[36] Sorokin, A., & Forsyth, D. (2008). Utility data annotation with Amazon Mechanical Turk. 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. doi:10.1109/cvprw.2008.4562953

[37] Luis von Ahn. Human Computation. In Design Automation Conference, page 418, 2009

[38] Cloud Computing Framework - A Framework to Evaluate Cloud Computing for IT Service Providers. (2011). Proceedings of the 1st International Conference on Cloud Computing and Services Science. doi:10.5220/0003459501310137

[39] Ciocarlie, M., Pantofaru, C., Hsiao, K., Bradski, G., Brook, P., & Dreyfuss, E. (2011). A Side of Data With My Robot. IEEE Robotics & Automation Magazine, 18(2), 44-57. doi:10.1109/mra.2011.940990

[40] Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. doi:10.1109/5.726791

[41] J. H. Lemelson. (1966). U.S. Patent No.3272347. Washington, DC: U.S. Patent and Trademark Office.

[42] S. (2016). ShaoqingRen/faster_rcnn. Retrieved December 11, 2016, from https://github.com/ShaoqingRen/faster_rcnn

[43] Zhou, L. (2007). SLAM: Simultaneous localisation and mapping. Odense.

[44] L. Bo, X. Ren, and D. Fox. Hierarchical matching pursuit for image classification: Architecture and fast algorithms. Advances in Neural Information Processing Systems (NIPS), 2011.

[45] Castle, R., Klein, G., & Murray, D. (2010). Combining monoSLAM with object recognition for scene augmentation using a wearable camera. Image and Vision Computing, 28(11), 1548-1556. doi:10.1016/j.imavis.2010.03.009

[46] Civera, J., Galvez-Lopez, D., Riazuelo, L., Tardos, J. D., & Montiel, J. M. (2011). Towards semantic SLAM using a monocular camera. 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. doi:10.1109/iros.2011.6094648

[47] Gupta, S., Girshick, R., Arbeláez, P., & Malik, J. (2014). Learning Rich Features from RGB-D Images for Object Detection and Segmentation. Computer Vision – ECCV 2014 Lecture Notes in Computer Science, 345-360. doi:10.1007/978-3-319-10584-0_23

[48] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H., & Davison, A. J. (2013). SLAM : Simultaneous Localisation and Mapping at the Level of Objects. 2013 IEEE Conference on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.2013.178

[49] Lai, K., Bo, L., Ren, X., & Fox, D. (2011). A large-scale hierarchical multi-view RGB-D object dataset. 2011 IEEE International Conference on Robotics and Automation. doi:10.1109/icra.2011.5980382

[50] Lai, K., Bo, L., & Fox, D. (2014). Unsupervised feature learning for 3D scene labeling. 2014 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/icra.2014.6907298

[51] A. ten Pas and R. Platt, "Using Geometry to Detect Grasp Poses in 3D Point Clouds," in International Symposium on Robotics Research (ISRR), 2015.

[52] Jiang, Y., Moseson, S., & Saxena, A. (2011). Efficient grasping from RGBD images: Learning using a new rectangle representation. 2011 IEEE International Conference on Robotics and Automation. doi:10.1109/icra.2011.5980145

[53] Fischinger, D., & Vincze, M. (2012). Empty the basket - a shape based learning approach for grasping piles of unknown objects. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. doi:10.1109/iros.2012.6386137

[54] Fischinger, D., Vincze, M., & Jiang, Y. (2013). Learning grasps for unknown objects in cluttered scenes. 2013 IEEE International Conference on Robotics and Automation. doi:10.1109/icra.2013.6630636

[55] Klingbeil, E., Rao, D., Carpenter, B., Ganapathi, V., Ng, A. Y., & Khatib, O. (2011). Grasping with application to an autonomous checkout robot. 2011 IEEE International Conference on Robotics and Automation. doi:10.1109/icra.2011.5980287

APPENDIX

The code and data can be found on the Eldertech server (kronos) under username “yyx84”. They are separated into several folders.

1. The folder named A_Fast_and_Accurate_Robotic_Grasp_System_Using_Deep_Learning_Code contains all the code folders. Under this folder there are “Arm Init” folder which contains all code to initiate a robotic arm in ROS, “DataSetAnnotationCode” which contains code to make a dataset, “GazeboModel” which contains code for robotic arm simulation and the “Grasping” folder which contains all the code to perform object detection, grasp point estimation and grasping.

2. The folder named “Pre-Train-Model” contains the initial parameters to train the Faster R-CNN network.

3. The folder named “VOCdevkit2007” contains all the datasets used to train the Faster R-CNN network.

4. All the details about how to use these code sets are included in the “README” file under the main folder.

The code is also available at github; the link is:

https://github.com/YueqiYu/A_Fast_And_Accurate_Robotic_Grasping_System_Using_Deep_Learning