ABSTRACT

NAG, ARUNAVA. A Vision-Based Odometry Model for Adaptive Human-Robot Systems. (Under the direction of Dr. Edgar Lobaton.)

Advances in automation, particularly those in industrial robotics, have brought significant production efficiencies to mid and large-scale manufacturers across the globe. Speed, advanced sensing and tighter integration with manufacturing execution systems have brought many benefits to modern production systems. Advances aside, industrial robotics still lack many safety attributes that do not allow them to operate unguarded and under conditions where true human-robot interaction can exist. Compounding the issue, programming methods and the overhead related to testing and validation run counter to real-time, rapid response ideologies. The dawn of easy-to-program, safe and cost efficient automation now drives the development of the next-generation of industrial robotics; systems that use smart sensing technologies, including force and torque monitoring that support "safe", incidental human operator contact. Now, because of advances on this front, it is possible for human and machine to work in concert with one another. Human-robot collaboration is “human-in-the-loop” automation; automation that simplifies tasks that would be too cumbersome and expensive to implement with rigid, engineered solutions. In this thesis a step towards “human-in-the-loop” automation has been taken, where an object is placed within the robot’s workspace and the robot is able to interact with the object without any pre-training in real time. The collaborative robot has been guided with a very cheap 3D vision solution which is generally not seen in the industry. The vision guidance helps to simplify the manufacturing assembly tasks using commercial hardware. Real-time streaming point clouds have been taken as input and compared with mesh CAD data using correspondence grouping algorithm to perform a fast as well as robust object
recognition solution has been demonstrated. Also using 3D geometry the position of the object is calculated and using the position of the object an optimized trajectory for the robot arms to reach the object has been planned. A real-time vision-guided human-robot collaborative system is discussed and also path planning optimization has been analyzed in this thesis. The case study demonstrates usage of Robot Operating System (ROS) and implementation of the entire recognition-manipulation phase in real time. It introduces the adaptive nature of the system for both safe product handling and safe H-R collaboration in real-time.
A Vision-Based Odometry Model for Adaptive Human-Robot Systems

by
Arunava Nag

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina

2016

APPROVED BY:

____________________  ______________________
Dr. Steven D Jackson    Dr. Cranos Williams

____________________  ______________________
Jason Low              Dr. Edgar Lobaton
                      Chair of Advisory Committee
DEDICATION

My work is dedicated to my father Mr. Ashish Nag and my mother Mrs. Suparna Nag. Thanks for your unconditional love and support. This would not have been possible without you.
BIOGRAPHY

The author was born in a small town named Durgapur in the state of West Bengal, India. He is the only son to Mr. Ashish Nag, and Mrs. Suparna Nag. He started his schooling in DAV Model School, and completed his primary to high school education in the same school. He pursued his undergraduate at Visvesvaraya Technological University with a major in Electronics and Communication Engineering. After his undergraduate he worked corporate IT sector and then left his job to pursue his higher education, a Master of Science in Electrical Engineering at NCSU, Raleigh, USA. The author loves and believes in sharing a strong bond with machines and enjoys animating in-animated bodies also called robots. The author also takes great interest in sports and plays different sports at competitive level. He has been a district level basket ball and a soccer player. He presently takes interest and pleasure in Badminton. He has represented his undergraduate and graduate university at various badminton competitions. He is also a bronze medal holder in the regional Badminton tournaments. He enjoys swimming and rock climbing as well. The author appreciates art, and loves to draw portraits. Apart from that he has a hobby and knack for photography and also enjoy learning new languages and art such as Kungfu-TaiChi.
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Edgar Lobaton for his advice and constant support. I also thank Dr. Steven Jackson and Mr. Jason Low for their support and permission to work at the Automation Lab in Fitts Industrial & Systems Engineering for providing me with all the necessary resources required for this research. I thank specially Mr. Jason low for his constant assistance, through his industrial experience and knowledge along with his adept technical skills during this research. Without his help and support this work would not have been complete.
# TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................ vii

LIST OF FIGURES ....................................................................................................... viii

Chapter 1 Introduction ................................................................................................. 1
  1.1 Motivation ............................................................................................................. 1
  1.2 Background ......................................................................................................... 6

Chapter 2 Approach Outline ......................................................................................... 13
  2.1 Introduction ........................................................................................................ 13
  2.2 Contributions .................................................................................................... 15

Chapter 3 Object Detection ......................................................................................... 18
  3.1 Introduction ........................................................................................................ 18
  3.2 Kinect - An Introduction .................................................................................... 19
  3.3 Object Detection-Methodology ......................................................................... 21
  3.4 Algorithm ........................................................................................................... 22
  3.5 Software Pipeline ............................................................................................... 23
  3.6 Results ................................................................................................................ 25

Chapter 4 Object Recognition and Pose Estimation ................................................ 27
  4.1 Introduction ........................................................................................................ 27
  4.2 Background ....................................................................................................... 29
  4.3 Approaches ....................................................................................................... 30
    4.3.1 Reference 3D Scan ..................................................................................... 34
    4.3.2 CAD Model alignment and augmentation .................................................. 35
    4.3.3 Results and Outcomes .............................................................................. 37
  4.4 Final Approach .................................................................................................. 38
    4.4.1 Recognition using CAD Model ................................................................. 39
    4.4.2 Pose Estimation ....................................................................................... 39
  4.5 Results ................................................................................................................ 43

Chapter 5 Robot Manipulation .................................................................................. 47
  5.1 Introduction ....................................................................................................... 47
  5.2 Baxter - An Introduction .................................................................................. 48
  5.3 Manipulation Libraries ..................................................................................... 53
    5.3.1 MoveIt ........................................................................................................ 54
    5.3.2 IK Service .................................................................................................. 54
    5.3.3 Object Finder ............................................................................................. 56
5.3.4 Baxter Arm Operation ............................................. 56
5.3.5 Baxter Gripper .................................................. 56
5.4 Method .............................................................. 57
5.5 Manipulation Issues ............................................... 59
5.6 Result .............................................................. 62

Chapter 6 Conclusion and Future Work ........................................ 63
  6.1 Conclusion ....................................................... 63
  6.2 Limitations ...................................................... 64
  6.3 Future Work .................................................... 65

Bibliography ........................................................................... 66

APPENDIX ............................................................................ 70
  Appendix A Source Code ................................................. 71
LIST OF TABLES

Table 3.1  Kinect Hardware Specifications ............................... 20
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>(a) Estimated Worldwide annual supply of industrial robots (b) Estimated worldwide annual supply of industrial robots at year end by industries 2012-2014 [1]</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Pose Estimation and Recognition, Baxter after picking up coke can (left to right)</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>PR2 robot performing (a) Detection (b) Recognition and (c) Grasp simulation [30]</td>
<td>11</td>
</tr>
<tr>
<td>2.1</td>
<td>Overview of the Process pipeline</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Schematic for Kinect V1 (Source Microsoft Library)</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Objects on the table are detected and segmented</td>
<td>24</td>
</tr>
<tr>
<td>3.3</td>
<td>RGB Camera view (left); Only clustered detected objects without the plane(right)</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>RGB Camera view (left); Only clustered and detected objects with axis colors(right)</td>
<td>26</td>
</tr>
<tr>
<td>3.5</td>
<td>RGB Camera view (left); Original point cloud merged with detected objects(right)</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Correspondence Grouping Algorithm using Hough 3D Scheme Computation Flow</td>
<td>31</td>
</tr>
<tr>
<td>4.2</td>
<td>Commands and parameter to run Correspondence Grouping Algorithm</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Identified Milk Carton in red (top left); Corresponding points between reference and scene model (top right); Recognized milk carton in red in gray-scale (bottom left); Command window output (bottom right) [40]</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Scanned OFI with highly dense of point cloud;NCSU USB block and a Metal fence Bracket (left to right)</td>
<td>35</td>
</tr>
<tr>
<td>4.5</td>
<td>Scanned Point Cloud alignment with its CAD model</td>
<td>36</td>
</tr>
<tr>
<td>4.6</td>
<td>(a) Metal Fence bracket (b) CAD model of the bracket with blue fiducial patch</td>
<td>37</td>
</tr>
<tr>
<td>4.7</td>
<td>(a) 3D scanned dense point cloud of the metal bracket (b) Top view of Kinect captured 3D point cloud of the metal bracket</td>
<td>38</td>
</tr>
<tr>
<td>4.8</td>
<td>Tf Tree showing link between Robot base and Kinect Frames to calculate Tf frame transformations</td>
<td>41</td>
</tr>
<tr>
<td>4.9</td>
<td>Flow diagram representation for the pose estimation process</td>
<td>42</td>
</tr>
</tbody>
</table>
Figure 4.10  Detected table with normal indicated with blue arrow (top left); Detected object with embedded unique ID (top right); Recognized object published as rostopic in command window 

Figure 4.11  (a) Bar Chart showing the confidence match percentage, (b) Kinect Position on Baxter

Figure 4.12  Output window showing calculated object and plane position

Figure 5.1  Baxter Specification [49]

Figure 5.2  7DOF Baxter arm and its various joints [50]

Figure 5.3  Baxter arms equipped with Series Elastic Actuator [50]

Figure 5.4  Baxter Arm equipped with camera [50]

Figure 5.5  MoveIt simulation window

Figure 5.6  Software flow for robot Arm path planning and object manipulation

Figure 5.7  (a) Warning generated due to kinect baxter frame mismatch (b) Calibration Markers

Figure 5.8  Time mismatch error leading to operation failure

Figure 5.9  Unsupported parameter error window between ROS KDL and Baxter URDF

Figure 5.10  MoveIt planning command window output

Figure 5.11  The robot grasps and picks up the can
CHAPTER 1

INTRODUCTION

1.1 Motivation

Industrial competitiveness demands the delivery of quality merchandise to the consumer at a low cost and in the shortest lead time. Manufacturers must always be concerned with re-evaluating what is the lowest production cost. Many factors, including those brought about by the ever-changing face and fancy of the consumer, challenge an already burdened system to be altered on a short notice with little budgetary latitude. As a result of system
changeovers, the production of marketable goods often gets hindered, as a result, heavy losses are incurred, leaving the customers dissatisfied. Often, faster order processing and shipment takes up the slack caused by manufacturing deficiencies, yet the leanest logistics only gets one so far.

To counter, the industry employs the highest quality resources, human and machine alike, and introduces automated subsystems, all in an effort to increase throughput while controlling labor costs. The demand for industrial robots has accelerated over the past several years due to significant innovation and lowering costs of ownership. Between 2010 and 2014, average robot sales increased by 17% per year. By the year 2014, sales of robots rose by 29% compared to 2010, to 229,261 units; the highest level of robot sales ever recorded for one year, as reported by International Federation of Robotics [1]. The various statistics on industrial robots can be seen in figure 1.1 and figure 1.2 which shows the growth of industrial robot production over the years. An industrial robot, as defined by ISO 8373, is an automatically controlled, reprogrammable, multipurpose manipulator, programmable in three or more axes. It may be either fixed in a place or mobile for use in industrial automation applications [1]. For safety reasons, the majority of these manipulators are still used in caged environments. Very little has changed regarding their interaction with the human worker.
Figure 1.1 (a) Estimated Worldwide annual supply of industrial robots (b) Estimated worldwide annual supply of industrial robots at year end by industries 2012-2014 [1]
1.1. MOTIVATION

CHAPTER 1. INTRODUCTION

Figure 1.2 Annual supply of industrial robots 2013-2014 and forecast for 2015-2018 [1]

An effort to automate an assembly process was made in summer 2015. The assembly process was carried out over a conveyor system that is present in the Fitts Department of Industrial & Systems Engineering. A case study included the assembly line where collaborative robots- Baxter from Rethink Robotics [2] and an industrial robot arm-Motoman [3] was staged to assemble MOSS blocks (Modular Robotics) [4] into functional toy-bots. The assembly process required precision and repetitive training through demonstration to the Baxter robot. The process also required pallets that were CNC milled for registration purposes. The pallets were used to introduce parts to the robot for assembly. The staging of the entire setup was done in a period of 3 months, which was time-consuming, cumbersome
and was not adaptive to changes in product design. It was realized that a higher level of automation would be the key to saving time and provide flexibility to the system. Also, a 3D vision-guided automated robot manipulation system has been presented in Cowley et. al [5] that motivated this thesis work. This publication [5] presents approaches to 3D perception and motion planning techniques of the manipulator to automate the general purpose robotic platform to recognize and manipulate the object of interest.

The thesis work involves the integration of existing technologies and uses modified intelligent computations to solve the problem of the initial training and setup of industrial robots at the conveyor systems which is expensive and time-consuming. A complete system is described in this thesis, that demonstrates an adaptive robot executing a planned trajectory, guided by a 3D depth sensor module without any pre-training. The computation is developed on the open source ROS [6] framework. The 3D sensor paired with the robot performs object recognition using an Apache-based [7] database containing a CAD model as the reference. Once the object is detected and recognized, a library designed for this application calculates the orientation of the object with respect to the reference model of the object using PCL [8], Geometry [9] and Transform(tf) [10] functions. Another library is built, that uses Baxter’s MoveIt [11] functions and also calculates the position of the recognized object with respect to the robot and 3D sensor. The position calculation is significant in planning an optimized path for the robot arms to perform a pick and place operation. The final result presented in this research that does not compromise speed and provides an automated solution that combines object recognition, path planning, and manipulation in real time without any previous training. The entire system, depicted in figure 1.3, illustrates the object recognition and pose estimation and the robot state after a pick of the recognized
1.2 Background

Since past 50 years, robotics and automation research groups around the world have invested considerable resources in evolving existing manual processes into automated mechanisms. The first industrial robot, conforming to the ISO definition was created in 1937.
1.2. BACKGROUND

by "Bill" Griffith P. Taylor, and was subsequently reported in Meccano Magazine [12][13][14]. This robot used parts from Meccano was a crane built specimen powered by a single electric motor. It was a 5-DOF robot which performed pick and place and rotational motions. In 1954 the first robotics patent was applied by George Devol [15]. He founded the company that produced robots called Unimation in collaboration with Joseph F. Engelberger in 1956. These robots used hydraulic actuators that were programmed using joint coordinates. The joint coordinates were stored in the robot’s memory during the robot training phase. A few years later Stanford arm [16] came into prominence, which was made by Victor Scheinman at the Standard University in 1969. The Stanford arm was a 6-axis robot arm. Following, ABB and KUKA jointly brought their first family of robots the IRB-6 and FAMULUS, which were microprocessor-controlled robot arms used for assembly and welding applications [17][18]. Since then, numerous developments have taken place in the field of industrial robots and its applications. Of late, vision-guided human-robot collaboration has been identified as one of the major research-oriented fields. Various hardware controls and software algorithms have been developed to create an efficient collaborative system to work alongside a human and aid industrial processes. Human safety has historically been a major concern. Several researchers around the world have put together various human-robot collaborative models that ensure human-in-the-loop safety. L. Wang et al. [19] in 2013, developed a cost-effective vision guided active collision model for human-robot collaboration. He proposed a solution that connected the virtual 3D models of the robot and human operators with a set of vision sensors to monitor collision detection in real time. This prototype aimed to improve the overall system performance and ensured zero robot programming for end users. The system featured alerts for the operator and stopped the robot or changed the robot trajectory path.
1.2. BACKGROUND

away from approaching operators as well.

Human-robot cooperation has always been found to have enhanced the organizational
proved the fact that human-robot collaboration can provide adaptability and reusability
of assembly systems. Vision inspection and guidance has been widely accepted to secure
the human-robot coexistence in a workspace. A workspace guided with multiple vision
module can be seen in T. Gecks et al [21] which made human-robot coexistence safer. A
collision detection algorithm has been introduced which is guided by inputs from multiple
cameras set at different angles. The cameras were used to monitor human-robot motion in
real time to change the robot motion path accordingly whenever a collision was detected.
An algorithm was presented in T. Gecks et al [21] that exhibited robust behavior with inex-
pensive computations. It used difference image processing algorithms in order to provide
real-time planning behavior.

Another interest and concern of industrialists and researchers throughout the world have
been to minimize the total production cost and increase the manufacturing productivity. A
step to address this concern has been presented in the body of work of Takata et al. [22] in
2011, where a human-robot allocation method has been suggested for a hybrid assembly
system. This method provided a selection method for initial human and robot allocation, to
minimize the expected production cost including labor and robot setup cost. The allocation
method also enhanced production volumes.

Technologies and state of the art that have made revolutionary enhancement to the research
and development in the field of industrial robotics to encourage Human-Robot cooperation
will be discussed as follow.
Robot Operating System (ROS) [6] is a collective software framework, used for robot software development. It is built on Linux Operating System platform. ROS-Industrial [23] is an open source project that extends the advanced capabilities of ROS software to commercially available industrial robot platforms. ROS-Industrial also enables intelligence in service robots. ROS-Industrial has been able to introduce new capabilities to the robots like mobile manipulation for logistics and warehousing, scanning plan and automatic path generation using pre-processed CAD model database. ROS-Industrial runs on several hardware platforms and is considered to be the key for working with any robot control and applications development. It ensures the quality of service, introduces toolchain for different models of a robot and performs standardization operations. One such example is Descartes [24], a software application introduced by ROS-Industrial that supports programming semi-constrained cartesian path planning for industrial robots. Descartes also assists application like robotic routing and blending/sanding. The key features of Descartes are path optimization, collision avoidance, near instantaneous re-planning and a plug-in architecture. Using machine vision and path planning abilities, Descartes can create and help servo a critical path using a robotic arm (5DOF) and perform industrial operations like laser trimming, inspection, and applications where the arm servos through highly twisted paths. ROS-Industrial is open source and is supported by a consortium that provides training. ROS-i has made a promising impact in the field of industrial automation and claims to be the future human-machine interface for the industry.

In recent years, collaborative Robotics has been a trend for researchers and industries throughout the world. HumaRobotics [25] has been developing several collaborative robot applications with NAO from Aldebaran, Baxter from Rethink Robotics, Q.bo from Thecor-
1.2. BACKGROUND

por a, the DARwIn-OP and the DARwIn-Mini. One of the interesting projects carried out by HumaRobotics was to bring collaborative capabilities to the ROS-enabled Baxter. HumaRobotics integrated Baxter with speech recognition, human posture detection, real-time interaction with the operator, and also added adaptive dialog abilities. Another domain of their Baxter project was to integrate Baxter with a commercially-available machine vision system. Machine learning techniques were introduced for better human-robot interaction. To ensure a better collaboration development of arm manipulator is necessary. Arm manipulator focuses on giving robotic systems the capability to behave like human limb segments in an anthropometric fashion, and to be able to hold objects with both arms to perform tasks such as assembly, trimming operation, or laser cutting. NIST and SwRI jointly developed ROS-Industrial Hilgendorf support software [26] for these purposes.

It is also important to have an optimized robot end effector trajectory planning system. An external 3D sensor or a vision module can prove to be essential for this purpose. Machine vision has played a major role in industrial applications since the 1950s. Robot arm planning has been a major research topic for several organizations as well. Optimized and collision-free trajectory planning has been explored since robots took on industrial tasks. A very novel method for collision-free robot arm planning in a cluttered environment using constraints from 3D sensor data has been introduced in Leeper et al. [27] in 2013. In this research [27], a constraint aware teleoperation controller was designed which tracked and updated a 6DOF end effector goal in real time to avoid the environment collisions, self-collisions and also solved its joint limits at the same time.

Object localization and recognition using depth sensor have been a repeated area of exploration. A brilliant library and utility incorporated in PR2 robots from Willow Garage [28],
presented by Binney et al. [29] in 2013, where a depth sensor was used to guide the pr2 robot to perform object detection and grasping operation can be seen in figure 1.4.

Figure 1.4 PR2 robot performing (a) Detection (b) Recognition and (c) Grasp simulation [30]

Key technological elements in the aforementioned research are Collaborative Robots, ROS, 3D depth Vision, Path Planning, and CAD models. In this thesis work, all these elements
are integrated into one application. In the following chapter, the process-pipeline will be discussed and the different modules in the pipeline will be introduced. In the later chapters, each module will be described and results will be presented. Finally, the thesis will end with a conclusion and the future work prospects.
2.1 Introduction

The research work integrated the vision and robot manipulation modules together. An object in the given workspace was detected and recognized using a 3D depth sensor. A Kinect Xbox 360 was used in this case. The recognition was done using a CAD model as its reference, which was stored in a local server database. The process used inexpensive computations that made use of open source libraries like PCL, Geometry, and Transform
as introduced earlier. The object recognition was performed in real time, using a library called object recognition kitchen (ORK) [31] that assisted a comparison between the CAD mesh file and the point cloud data streamed by the Kinect.

Figure 2.1 Overview of the Process pipeline

The position of the recognized object was calculated with respect to the robot in order to plan a trajectory using Baxter-MoveIt configuration and library function for the robot arms to perform a pick and place operation without any previous training or simulation.
2.2. CONTRIBUTIONS

The entire pipeline was executed in real-time and the process is shown in the figure 2.1. This process pipeline proposed a cost efficient solution which also saved time. The end product ensured zero programming for the end users. This approach presented in this thesis ensures flexibility to industrial product design changes as only a CAD model is required for the database in order to carry out the complete recognition and manipulation operation in real-time. This thesis work has also demonstrated the use of an inexpensive 3D sensor which is a Xbox Kinect in this case that costs approximately 200 USD, instead of an industrial machine vision that costs thousands of dollars. This Kinect was used to provide a robust and fast recognition solution in 3-dimensional space. The final object of interest (OFI) was chosen to be a cylindrical can, to impress upon the idea that the objects do not compulsorily need to be symmetrical in nature. The database used for this purpose is called CouchDB which has been built over a framework of Apache as introduced earlier [7].

2.2 Contributions

The entire setup and integration of all modules took considerable time through a thorough learning process. Though the research leverages existing technologies, it proposes an innovative solution for an industrial assembly system. The contributions made in this research could be summarized as follows.

1. Setting up a Kinect with the Baxter took some time. Due to the software versions and open source development, the right kind of driver that could accommodate and prove to be compatible with each other was an issue to solve. This was possible by using the right driver from OpenNI [32] and Primesense [33], that supported all the
2.2. CONTRIBUTIONS

2. Point cloud data was processed and computed using the PCL library. The working library was coded using the existing algorithms. The computations were optimized in order to accommodate the type of incoming data. The computation was also integrated with ROS to provide real time results.

3. Object recognition was performed using ORK library, and was designed in order to perform recognition in real time and also output the object position and plane position that would be required for robot trajectory planning. All these libraries were compiled with dependencies and issues were troubleshooted and debugged to yield optimal results.

4. The application was programmed using C++. The kinect was also calibrated for this purpose in order to couple this external 3D sensor with the Baxter robot.

5. 3D scanned data down-sampling and optimization was performed. Also the CAD models were designed and embedded with safe grasping fiducials.

6. Three libraries for the purpose of localizing the object were developed. One for the arm motion planning of Baxter, second library for gripper action, and third for finding the object in workspace and to calculate its position. These libraries integrated MoveIt functions efficiently and enabled subscribing to ROS topics internally, which contained localization messages.

7. The 3D geometry calculations and transformations were computed as well. They were planned sequentially to perform almost exact estimation of the object. These
calculations were done using functions from the ROS 3D geometry libraries.

8. Launch files were created that integrated all the libraries and performed initialization of Kinect, Baxter arms, Baxter's trajectory planner and Baxter's grippers. As a result, the entire application and its several nodes and processes could be triggered very easily. The integration was realized in compact launch files with already introduced required pre-settings.

9. Several issues between Baxter and Workstation such as timing issues, unsupported kinematic parameters, and mismatch of ROS messages from the workstation to the robot were solved to execute a successful robot manipulation.

This pipeline sees integration of several modules together. The following chapter will explore in to each module with in-depth detail and also the result will be discussed.
CHAPTER 3

OBJECT DETECTION

3.1 Introduction

This chapter introduces the first module of the pipeline, object detection. Object detection is the process or the technology related to computer vision where objects are segmented from one another in a cluttered environment. In this research, streaming point cloud data has been taken as an input for the detection algorithm and performed segmentation and clustering of objects as an output. The detection is performed in a planar workspace and
uses segmentation algorithm to segment planar sections and then use Euclidean Clustering methods to cluster the point clouds in the scene. The hardware or the 3D depth sensor used for this purpose is a Kinect Xbox 360 v1. In the following sections of this chapter, all the hardware and software components will be discussed in details and the results will be presented.

### 3.2 Kinect - An Introduction

The Kinect sensor is mounted on a horizontal bar connected to a small base with a motorized pivot. It is designed to be positioned lengthwise above or below the video display. The device features an RGB camera, depth sensor and multi-array microphone as seen in figure 3.1 and the specifications can be found in table 3.1 [34], [35].

- **RGB Camera**: It stores three channel data in a 1280x960 resolution. This makes capturing a color image possible.

- **Infrared (IR) emitter & IR Depth sensor**: The emitter emits infrared light beams and the depth sensor reads the IR beams reflected back to the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor. This makes capturing a depth image possible.

- **Multi-Array Microphone**: This contains 4 microphones for capturing sound. Because there are four microphones, it is possible to record audio as well as find the location of the sound source and the direction of the audio wave.

- **Accelerometer**: A 3-axis accelerometer configured for a 2G range, where G is the
acceleration due to gravity. It is possible to use the accelerometer to determine the current orientation of the Kinect.

**Table 3.1** Kinect Hardware Specifications

<table>
<thead>
<tr>
<th>Kinect Features</th>
<th>Array Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewing angle</td>
<td>43° vertical by 57° horizontal field of view</td>
</tr>
<tr>
<td>Vertical tilt range</td>
<td>± 27°</td>
</tr>
<tr>
<td>Frame rate (depth and color stream)</td>
<td>30 frames per second (FPS)</td>
</tr>
<tr>
<td>Audio format</td>
<td>16-kHz, 24-bit mono pulse code modulation (PCM)</td>
</tr>
<tr>
<td>Audio input characteristics</td>
<td>A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression</td>
</tr>
<tr>
<td>Accelerometer characteristics</td>
<td>A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.</td>
</tr>
</tbody>
</table>
For initializing the Kinect nodes in the ROS environment open source ROS packages for Kinect, such as OpenNI has been used to serve as the driver for all the sensors and frames present on the Kinect as stated earlier in chapter 2. For operating with Baxter the camera_optical_depth_frame of Kinect is used to perform detection operation.

### 3.3 Object Detection-Methodology

Detection of OFI present within the field of view was a vital process and required point cloud segmentation and clustering. The Kinect depth sensor fetched point clouds of the environment, where each point contained the 3D vector information of the point in space, that is, x, y and z and their normal information. The algorithm to detect the objects in the workspace was written in C++, that was built on top of the ROS [6] framework. Segmentation
was performed in real time from the streaming depth information fetched by the Kinect. Instead of a static offline point cloud, the input array was selected to be a point cloud pointer type that was filled and refreshed continuously as the Kinect streamed point cloud data to achieve real-time clustering. This process can be divided into two stages:

- **Segmentation**: First, the dominant plane was detected from the point cloud using the Random sample consensus (RANSAC) algorithm \[36\]. The inliers were taken into consideration and the outliers were subtracted, leaving only the OFI in the field of view. The OFI was detected along with its defining features.

- **Clustering**: Clustering is the process of the grouping of point clouds based on similar features or distances. For this research, Euclidean distances are taken as the parameter for clustering the objects from the streaming point cloud. The point clouds were clustered by using the PCL library as demonstrated in Radu Rusu thesis \[37\]. As a result, the OFI along with its major defining feature could be visualized in the cloud viewer and Rviz as well.

### 3.4 Algorithm

The process of detection can be represented into the following mathematical algorithm. Assuming a point cloud in our scene with a planar work space and objects that were placed on it. The goal of this algorithm is to find and segment from each object into individual clusters that lie on the plane. We use a Kd-tree structure in order to find the nearest neighbors to compute the Euclidean distances. The algorithmic steps follow \[37\]:

1. A Kd-tree structure was created for the input point cloud \(P\).
2. An empty array of clusters $C$ and a stack for points that need to be checked $Q$ is created.

3. For every $p_i \in P$, following steps are performed in a loop:

   - First $p_i$ is added to the current queue $Q$;
   - For every point $p_i \in Q$, the following was done:
     - A set of point neighbors $P_k^i$ of $p_i$ was searched within the sphere with radius $r < d_{th}$;
     - For every neighbor $p_i^k \in P_i^k$, it was checked whether the point was already been processed or not; if not add it to $Q$.
   - After all points in $Q$ was processed, $Q$ was added to the list of clusters $C$, and $Q$ was reset to a zero.

4. When all points $p_i \in P$ were processed and were added to the list of point clusters $C$ the algorithm was terminated out of the loop.

### 3.5 Software Pipeline

The coding pipeline flow for the above algorithm can be seen in the figure 3.2. The flow diagram describes step by step execution of the algorithm to perform clustering.
Figure 3.2 Objects on the table are detected and segmented
3.6. RESULTS

The software flow shown above was carried out in real time, and no offline data processing was done. The results were obtained immediately and was visualized in the ROS visualization rviz [38].

3.6 Results

The result obtained from the pipeline in figure 3.2 can be seen in the figure 3.3, 3.4, 3.5, where the left pane shows the scene view containing a planar table with objects on it. The right pane shows the final output with only the clusters present on the planar surface minus the planar surface point cloud. The detected objects within the field of view were compared with the reference CAD model in order to recognize the object of interest if present. This will be discussed in the next section.

![Image of RGB Camera view (left); Only clustered detected objects without the plane (right)](image)

**Figure 3.3** RGB Camera view (left); Only clustered detected objects without the plane (right)
3.6. RESULTS

CHAPTER 3. OBJECT DETECTION

Figure 3.4 RGB Camera view (left); Only clustered and detected objects with axis colors(right)

Figure 3.5 RGB Camera view (left); Original point cloud merged with detected objects(right)
CHAPTER 4

OBJECT RECOGNITION AND POSE ESTIMATION

4.1 Introduction

Human possess remarkable ability to determine and classify objects simply by looking at them, quickly and very accurately. But using a computer or an artificial vision to do this same task is very difficult and complex. The human developed vision paired with artificial
intelligence knows exactly what feature to look at when it comes to pointing out a defect in an industry product. The process of identifying a specific object using matching, learning, or pattern recognition algorithm in a digital image or video by using a feature or appearance-based techniques can be defined as object recognition. This chapter will illustrate the recognition phase using a CAD model as its reference. There are several ways to perform recognition using CAD data; Edge Detection, Primal Sketch, Marr, Mohan and Nevatia [39] are just a couple of methods. In this research edge detection method has been used. The shape and edge point clouds were used to compare with the reference model to predict and instance in the field of view. The algorithm used for this purpose was called Correspondence Grouping algorithm [40] to perform 3D Object Recognition. The algorithm uses a Hough's 3D Voting Scheme[41]. In this research the algorithm was computed in C++ style, to cluster the point cloud and point-to-point correspondence between model and scene from the camera were obtained. Each cluster represented a possible instance of the OFI in the scene. Once an instance was confirmed using the depth as well as registered (colored) point cloud it marked the object as recognized and embedded confidence percentage and unique ID information. This algorithm also outputted the transformation matrix which contained 6DOF pose estimation information of the object in that scene with respect to the reference model. Following which functions from ROS 3D geometry libraries were used in order to localize the object and plane with respect to the robot reference frame and camera frame. The output was the position information of both object and planes in the current scene. The CAD mesh model was stored in a database with almost the exact measurements as that of the original object. The recognition process used the ORK library [31] for comparing the mesh file with the streaming point cloud. The mesh file in the database was of the format
4.2. BACKGROUND  

CHAPTER 4. OBJECT RECOGNITION AND POSE ESTIMATION

.stl type, though .obj can be used as well to get better results as .obj contains the surface and color information where .stl is greyscale in nature and do not contain the color information. The point cloud from the Kinect was of the format .pcd [40] type. The chapter will also introduce some background and related work of various object recognition methods, then discuss all the methods and approaches used for this research and finally present the results at the end of the chapter.

4.2 Background

The nature of computer vision is to tell a story by processing a single image, or a sequence of images or a video. The recognition of objects has been studied for more than four decades, researchers all around the world have put together a significant effort to introduce and develop computer vision algorithms for the purpose of recognizing general objects at the camera view. The problem of object recognition has been studied extensively in cognitive science, neuroscience, psychology and robotics [42,43]. Recognition of objects in real time has made the problem even more interesting as now computation time also plays a major role. One such example can be seen as performed by Labb M. [44] where a real-time segmentation and recognition is performed using image and depth information which needs to be pre-captured using a depth sensor and set as a reference model. There are several kinds of recognition, one such demonstration can be found in Bourdev et al. [45] where body parts are detected using a data-driven search procedure to recognize the tightly clustered 3D joint configuration and 2D image appearance as well. Also, researchers have put efforts to recognize actions with the help of body positions, one such work can be seen in Maji et al. [46]. A poselet activation vector containing distributed pose and appearance
of people, and combines various representation with other sources of information such as interaction with different objects and perform action recognition. It has been a major trend to develop solutions that are not only efficient but also have low computational cost. A quick method of object detection can be seen in C. Gu et al [47]. In this method, a multi-component approach is presented to detect objects. It uses a visual cluster from the data which have appearance and configuration spaces, and then each classifier for each component are trained and a second classifier aggregates the responses from multiple components at the category level. Also to reduce computation an object selection window is adopted, a segmentation based selection process is performed that produces only 500 windows per image, while preserving the high object recall. In the next section, we will look into the methodologies used in this research and their respective outcomes.

4.3 Approaches

As discussed earlier, the concern for the overall recognition method was to find a robust and computationally fast technique. Moreover, getting the method to work in real time was another concern vital to the metrics of success. Throughout the entire process of recognition, the foundation to the algorithm to perform recognition has been the Point cloud Correspondence Grouping algorithm. This is a clustering algorithm using the 3D Hough voting scheme [41] for finding the highest number of correspondences and predict on if an instance is present in the scene with respect to the reference. The computational flow of the correspondence algorithm is shown in figure 4.1.
4.3. APPROACHES

CHAPTER 4. OBJECT RECOGNITION AND POSE ESTIMATION

Figure 4.1 Correspondence Grouping Algorithm using Hough 3D Scheme Computation Flow

31
4.3. APPROACHES  

CHAPTER 4. OBJECT RECOGNITION AND POSE ESTIMATION

This algorithm could be tested by the following commands:

```
arunava@jeet:~$ ./correspondence_grouping milk.pcd milk_cartoon_all_small_clorox.pcd
```

where milk.pcd is the reference model and milk_cartoon_all_small_clorox.pcd is the scene point cloud. Alternatively, all the parameters and the radii in units of cloud resolution can be specified

```
arunava@jeet:~$ ./correspondence_grouping milk.pcd milk_cartoon_all_small_clorox.pcd milk.pcd milk_cartoon_all_small_clorox.pcd -r --model_ss 7.5 --scene_ss 20 --rf_rad 10 --descr_rad 15 --cg_size 10
```

**Figure 4.2** Commands and parameter to run Correspondence Grouping Algorithm

The parameters could be defined as below:

- **model_ss** val: Model uniform sampling radius (default 0.01)
- **scene_ss** val: Scene uniform sampling radius (default 0.03)
- **rf_rad** val: Reference frame radius (default 0.015)
- **descr_rad** val: Descriptor radius (default 0.02)
- **cg_size** val: Cluster size (default 0.01)
- **cg_thesh** val: Clustering threshold (default 5)
4.3. APPROACHES

4.3.0.1 Result

Sample point cloud was downloaded from PCL documentation website [40] to test the algorithm. The scene model contained 2 bottles and one milk carton. The reference point cloud was the milk carton. Using the corresponding grouping algorithm instances were found and the milk carton was recognized in the scene. This operation was tested in offline. The result obtained can be seen in figure 4.3.

![Figure 4.3](image)

**Figure 4.3** Identified Milk Carton in red (top left); Corresponding points between reference and scene model (top right); Recognized milk carton in red in gray-scale (bottom left); Command window output (bottom right) [40]
4.3. APPROACHES

Chapter 4. Object Recognition and Pose Estimation

In figure 4.3 the red marked box is the identified milk carton in the point cloud scene at the top left. The top right pane shows the number of correspondences between the reference and scene model. The bottom left pane shows the recognized milk carton in grayscale. The bottom right pane shows the command window output containing translation and rotational matrices.

4.3.1 Reference 3D Scan

A possible verification method is proposed that uses an intermediate 3D scan data of the OFI as a reference. The 3D scan is obtained using a high end laser scanner. The OFI was scanned in a 360 degree rotating platform enabling 360 degree scanning of the object. Both the top and bottom faces were taken in to consideration separately to obtain the 3D point clouds of all the surfaces and features. Later, using a model optimization platform, the scanned 3D point clouds of top and bottom faces where merged together in order to receive the 3D point cloud for the entire body. Some of the 3D scanned models used as test cases are shown in figure 4.4.
4.3.2 CAD Model alignment and augmentation

The scanned models were matched with the CAD model to align the point cloud and the features exactly with the actual object. This was done using readily available commercial software packages where several methods such as Gaussian surfaces and best fits were used to get a point cloud that is aligned with the CAD model. The output point cloud contained approximately million points that had the 3D vector and its surface normal information. This alignment can be seen in figure 4.5 wherein the left pane the point cloud is present and in the right pane the point cloud in green is aligned with the CAD model. Some features are also highlighted with information of those regions in the figure 4.5 right.
Figure 4.5 Scanned Point Cloud alignment with its CAD model

The 3D scanned file format had 6 axes of information, namely, normal x, normal y, normal z, x, y, z. The CAD model was also augmented with fiducials in order to mark points for the robot to grasp safely. The augmentation was performed in a CAD software. The fiducials that were embedded on the CAD when converted to point cloud format it supplied the added information as robot grasping locations on the previously existing point cloud. The fiducials created can be as seen in figure 4.6 marked as blue in the right pane when compared with the real object.
4.3.3 Results and Outcomes

The above methods could not be used as a robust means for recognition in an industrial environment. There were several drawbacks to the above methods. Firstly, it brought into the picture too many hardware equipment such as 3D scanner, which was expensive. Also, the software license for this hardware equipment was expensive, whereas the goal of this research was to provide an easy and cost-effective industrial solution. Secondly, due to the usage of so many references, the amount of computation increased greatly. Putting everything back into the robot’s ROS environment became challenging. Thirdly, keeping into account the fact the entire process needed to be real time, the huge point cloud data retrieved from the scanner was a major concern. The point cloud was too dense and had to be down-sampled several times to get a less dense point cloud which could be comparable to the point cloud obtained by that of a Kinect. This took a large time, and eventually even after down-sampling the clouds could not be compared and the algorithm failed to locate
any instance in the field of view as the down-sampled reference model lost many vital features. If the reference point cloud was not down-sampled, it took excessive orders of magnitude to be processed, all with limited results. The 3D scanned cloud before down-sampling can be seen in figure 4.7(a) and the top view of 3D point cloud obtained by the Kinect can be seen in figure 4.7(b). A considerably huge difference could be seen.

![3D scanned dense point cloud of the metal bracket](a) ![Top view of Kinect captured 3D point cloud of the metal bracket](b)

Figure 4.7 (a) 3D scanned dense point cloud of the metal bracket (b) Top view of Kinect captured 3D point cloud of the metal bracket

### 4.4 Final Approach

Due to several drawbacks as discussed in section 4.3.3, it was essential to find a faster, more robust and cost effective method to perform recognition. Also, it was undesirable and impractical to use so many input reference models to assist in recognition. Hence a CAD model was used solely for the recognition method as described in the following chapters.
4.4. FINAL APPROACH  CHAPTER 4. OBJECT RECOGNITION AND POSE ESTIMATION

4.4.1 Recognition using CAD Model

The CAD model was selected to be used as the sole reference input model in the form of a mesh file. The supported formats were .stl and .obj. A local database, called CouchDB, was loaded with a mesh file or the CAD model of the OFI with a unique id. Then a library named Object Recognition Kitchen (ORK) was used which used the registered (color information) point cloud and compared the mesh file with the object in the scene. The importance of using color information in this recognition pipeline was to be able the texture information along with the greyscale CAD mesh file. The mesh file contained tessellated surfaces that were compared with the registered point cloud information found in the scene. It used the same process pipeline, that could find the planes in the scene and then the objects over the plane were recognized as well. The library was used as an aid along with the registered depth information and correspondence algorithm with point cloud library operation. The result of the above process was published in forms of rostopics namely, `table_arrays` and `recognized_object_arrays`. Those topics were subscribed and could be visualized in Rviz. This provided a fast robust recognition solution from a considerably larger distance and at different light exposures as well.

4.4.2 Pose Estimation

The next task was to determine the position of the object with respect to the robot. The correspondence algorithm with ORK provided the initial 6DOF of the position of the object with respect to the reference model. In order to estimate the position with respect to the robot, matrix manipulations of transform frame were computed. The three major
transforms were:

1. The static transform between Kinect and the robot base frame.

2. The transform from the object frame to Kinect optical depth frame.

3. The transform from the object frame to the robot reference base frame.

The transform was calculated and computed using tf and geometry ROS packages. This was programmed in C++ leading to a new ROS package created for this purpose. The rostopics were subscribed to and a tree-like structure was formed using tf amongst all the required geometry frames, then the transform of one frame to another frame on the tree was computed for both object frame to Kinect frame and robot base frame, respectively. After calculating both the transforms, they were multiplied to find the position of the object with respect to the robot. To describe a tf tree which is generally complicated and very big in structure, a small part of the tree connecting the robot base frame and Kinect camera link and other neighboring links is shown in figure 4.8 to illustrate the tf tree transformation. The static transform was found using ROS packages that also helped in calibrating the Kinect with respect to one of the robot joint frames using markers. After the calibration process, the tf static transform was given out as a result with the x, y, z, yaw, pitch, roll information between Kinect and the robot base.
4.4. FINAL APPROACH  CHAPTER 4. OBJECT RECOGNITION AND POSE ESTIMATION

**Figure 4.8** Tf Tree showing link between Robot base and Kinect Frames to calculate Tf frame transformations

The software flow can be visualized in the following flow diagram representation in figure 4.9
Figure 4.9 Flow diagram representation for the pose estimation process
4.5 Results

The object recognition results were found to be robust and time efficient and computationally cheap. The results were visualized in rviz as shown in figure 4.10.

![Figure 4.10](image-url) Detected table with normal indicated with blue arrow (top left); Detected object with embedded unique ID (top right); Recognized object published as rostopic in command window
Figure 4.11  (a) Bar Chart showing the confidence match percentage, (b) Kinect Position on Baxter

A plot for the percentage of recognition confidence versus distance (measured in inches)
from the depth sensor can be seen in the figure 4.11 (a). Beyond 104 inches it fails to find planes and the object is also lost. The camera was set up at an inclined angle as in figure 4.11 (b) from the robot head to the table while measuring the recognition efficiency.

![Output window showing calculated object and plane position](image)

**Figure 4.12** Output window showing calculated object and plane position

The result of the pose estimation was nearly exact and both the position of the object plane and the object could be calculated and displayed. The recognized object position could be seen in the command window as in figure 4.12. The position is refreshed after every 0.5 seconds. The object position is shown in the window with x,y,z,row,pitch and yaw information with its time stamp.
In the next chapter how the object position information is utilized to plan a trajectory for the robot arms and perform pick and place operation will be discussed in details.
5.1 Introduction

Robot manipulation and end effector action is an expansive area of work. It involves processes such as trajectory planning, path optimization, collision avoidance, grasping, and simulation. This has also been a major topic of research as already seen in chapter 1 section 1.2 in the literature survey. In the previous chapter, the vision module was discussed and various phases of the vision operations that demonstrated real-time object recognition
and position estimation with respect to the Kinect and the Robot’s base frame. The next task was to use the position information to devise a way for the robot to synchronize with the system and plan a path to perform recognized object manipulation and execute a pick and place operation. To perform this stage, the manipulator selected was Baxter from Rethink Robotics. In order to perform this operation using Baxter, Baxter’s MoveIt Library and MoveIt configuration were used to plan a path from Baxter’s initial joint state to the object location using optimized path planning and robot dynamics. The entire process was executed in real time through sequential planning. The planning was first done in Cartesian space, and later it was converted into the respective robot joint angles. The computation for the robot manipulation has been inspired from Baxter project of CWRU robotics [48]. The flow of planning may be best understood with the aid of figure 5.6.

5.2 Baxter - An Introduction

Baxter is a 7DOF collaborative industrial robot. It was launched and made commercially available by Rethink Robotics in 2012. It has two configurations; one that uses a proprietary application built on a Linux Kernel by the name of Intera. It can be used by any end-user who has no programming language experience. The other configuration uses the ROS framework, also called the Baxter Research Robot; is an open platform where the robot can be simulated and programmed to perform desirable operations. For this research, the Research Configuration is used. A research Baxter robot can be seen in figure 5.1. Some facts about Baxter [49] follows:
• 14 DOF (7DOF/arm), 5'10" to 6'3" tall friendly robot, weighs 165lbs to 305lbs, with a max payload of 5lbs, and gripping torque of 10lbs.

• 3rd gen, Core i7-3770 intel processor with HD graphics, 4GB RAM, and 128GB SSD.

• Camera has an effective resolution of 1280*800 pixels, and infrared sensor range of 4-40cm.

• Requires 120v supply voltage, 6 amps rated current.

• The operating system used is either Intera or ROS based on the robot configuration or type.
5.2. BAXTER - AN INTRODUCTION

5.2.0.1 Baxter Arms

Baxter Arms [50] are an integral part of Baxter that differentiates Baxter from other robots and makes it safe and collaborative in nature. A Baxter has two 7 DOF arms which provide kinematic redundancy that improves manipulability and safety. A Baxter arm with its joints can be seen in figure 5.2.

![7DOF Baxter arm and its various joints](image)

*Figure 5.2 7DOF Baxter arm and its various joints [50]*

- Used in various applications such as kitting, packaging, loading and unloading, machine tending, material handling
The arm joints as shown in 5.2 are called as follows:

- S0 - Shoulder Roll
- S1 - Shoulder Pitch
- E0 - Elbow Roll
- E1 - Elbow Pitch
- W0 - Wrist Roll
- W1 - Wrist Pitch
- W2 - Wrist Roll

Baxter Arms are collaborative in nature because of the safety mechanisms it employs. Its method of operation relies on a system of Series Elastic Actuators (SEA). This actuation technique or mechanism helps in the safe movement of the robot links. The motor/gearing elements are joined using a spring. The output of the actuator is also implemented with the same mechanism. Implementing such a mechanism results in "gains in stable, low noise force control, and protection against shock loads". The springs employed within these gear motors are deformable when a collision occurs with a human-in-the-loop. Also, these springs help in regulating the torque of the actuators. The spring sends out torque measurements which can be used to control the torque, impedance, and inertia of the arms. The inside of Baxter’s arm with a Series elastic actuator can be seen in figure 5.3.
Baxter end-of-arm tool plate is not equipped with safety elements like series elastic actuator, instead several sensors are installed in it. The Baxter end-of-arm contains:

1. In-hand camera.
2. Accelerometer.
3. Infrared Range Sensors
4. Navigators or Cuff buttons
5.3. MANIPULATION LIBRARIES

The camera in the hand acts as an eye for the Baxter arms and which can be used to assist object manipulation and also can be used for external camera calibration. A Baxter end effector with an camera can be seen in figure 5.4.

![Figure 5.4 Baxter Arm equipped with camera](image)

**Figure 5.4** Baxter Arm equipped with camera [50]

### 5.3 Manipulation Libraries

There were several libraries used for the purpose of object recognition and robot manipulation. The libraries built for this thesis, made use of functions from open source functional libraries such as MoveIt and Inverse Kinematics (IK) to exploit Baxter’s arm features to perform desirable operations. Firstly MoveIt and IK will be described briefly in sections 5.3.1 and 5.3.2 and then the libraries built for this thesis work will be introduced briefly in
sections 5.3.3, 5.3.4 and 5.3.5.

### 5.3.1 MoveIt

The MoveIt [51] is a motion planning platform that enables capabilities of robot kinematics namely Jacobian, FK, IK and also helps in motion planning. The MoveIt also helps in representing and creating the environment to the robot for trajectory planning simulation. This feature helps in robot representation, collision avoidance, constraint evaluation. Both C++/Python language style can be used to exploit the functions of this library to manipulate and retrieve information about the robot such as robot joints, scenes, and motion plans. MoveIt is initiated in Baxter by adding joint trajectory action server of the robot. The result can be visualized and planned in the MoveIt Rviz plugin that helps in simulation as well as the execution of robot arm trajectories. A representation of Baxter in MoveIt Rviz plugin can be seen in figure 5.5.

### 5.3.2 IK Service

The IK or the inverse kinematics [52] in Baxter helps in communication with ROS nodes present on the robot’s operating system. It also helps in controlling the arm of Baxter using the joint states and joint angles of Baxter’s arm. Each joint such as shoulders, elbow and wrist can be programmed using the IK service to any joint location. The IK service enables easy trajectory planning in the Cartesian space of the robot arm.
IK also serves as a conversion bridge between Cartesian (x, y, z, roll, pitch, yaw) representation to actual robot 7-DOF joint states.
5.3.3 Object Finder

This library was built that included functions of 3D geometry functions. Object Finder was built to ensure easy localization of the object in the 3D workspace placed in front of the robot. This library subscribed to the recognized object array information, and with the help of transform functions it calculated the object position with respect to Baxter and Kinect. The library calculates all the transform between the object, Baxter and Kinect frames and is able to localize the object and also the plane. The output of this result is 6-dimensional that is x, y, z, roll, pitch and yaw Cartesian information of the object and also plane with respect to the Robot.

5.3.4 Baxter Arm Operation

This library took into account of 3D geometry and MoveIt functions. It incorporated the arm dynamics and also used the 3D geometry to perform calculation of a path. This library also initiates opening and closing of the gripper. It uses the 3D geometry and takes into account object centroid, major axis and plane normal into account to find the optimized trajectory from initial to object location.

5.3.5 Baxter Gripper

This is comparatively a smaller library that is used to carry out the open and close operation of the Baxter gripper. Also it publishes information of the gripper position and gripper open close operations.
5.4 Method

As seen in the figure 5.6 after the OFI position information was sent to the Baxter MoveIt, an action was triggered using MoveIt functions and the gripper was set to true and open. At first, the robot’s initial joint angles were calculated and stored. A trajectory was planned from the initial joint angle position to the joint angle where the object has been localized. To optimize the path and to select the right path, parameters such as the plane normal, major axis of the object and centroid of the object was taken into account. The path was first planned in Cartesian space. The cartesian path was taken as an input and the final robot joint path was planned. The robot’s joint angle was generally represented in a 7-dimensional matrix. As the final optimized robot joint specific path was planned, the Baxter arm (right arm in our case) servo to the object location. Based on the object’s centroid, the gripper was closed using MoveIt gripper close function. The above procedure was carried out in real time and took considerably less than a millisecond for MoveIt to plan this trajectory. Three new libraries were created in order to utilize Baxter’s arm features which have been described in previous sections. The robot trajectory was also simulated in MoveIt to compare the calculated path versus simulated path from same start and goal points, and both were found to be the same, which validates the vision guided trajectory planning and manipulation. The software pipeline has been briefly described in figure 5.6.
5.4. METHOD

CHAPTER 5. ROBOT MANIPULATION

Figure 5.6 Software flow for robot Arm path planning and object manipulation
5.5 Manipulation Issues

During robot manipulation, there were several issues that needed to be solved to perform a successful manipulation. In this section we will discuss those issues and how they were solved.

1. **Kinect-Robot Calibration**: Even though the entire pipeline, once correctly planned and tuned, still the Kinect failed to synchronize with the Robot. Upon investigation, the reason found was that Kinect needed to be calibrated with the robot, even though a static transform was established between the Kinect and the robot and declared during the launch process. This generated warnings and dropped all the messages being passed between the Kinect frame and robot reference frame, as can be seen in the figure 5.7 (a). To solve this, markers as seen in figure 5.7 (b) were used for the calibration of the Kinect with respect to one of the hand camera using Baxter calibration packages.

![Figure 5.7](image-url)  
(a) Warning generated due to kinect baxter frame mismatch (b) Calibration Markers

Figure 5.7 (a) Warning generated due to kinect baxter frame mismatch (b) Calibration Markers
2. **Workstation and Robot Timing Mismatch**: Another issue was to address the mismatch of timing between robot and workstation. Generally, ROS packages use the system time as the clock reference. But Baxter has its own CPU and clock, hence it was found that the robot did not respond even if there were 0.25 seconds of delay between the two systems, as can be seen in figure 5.8. To address this issue NTP [53] or the network time protocol was used to sync all the systems into same server time.

![Figure 5.8 Time mismatch error leading to operation failure](image)

3. **Unsupported Inertia by Kinematics and Dynamics Library (KDL)**: The Baxter came with a default configuration which also contains the robot’s inertia for its movements. The final issue was to address the unsupported inertia parameter by the KDL library, which was a Baxter specific parameter, but not common to the dynamics library of ROS. This error confused the robot between its own defined functions and the ROS KDL libraries. This can be seen in figure 5.9.
5.6. RESULT  

The result of the above process showed a trajectory planning by the robot arm. The results were seen in the command window, that showed the change of robot joint states and also the time required to plan the trajectory. The command window output can be seen in figure 5.10.

Figure 5.9 Unsupported parameter error window between ROS KDL and Baxter URDF

To solve this problem, a dummy transform link was introduced in the transform tree. Hence there was now two transform tree one was the parent tree and other was the child tree. The inertia was assigned to the child tree, while the parent tree still worked. The static transforms were declared from Kinect frame to the dummy link and another static transform was established between the dummy link the robot reference frame.
5.6. RESULT

CHAPTER 5. ROBOT MANIPULATION

The final output where the robot picked the can be seen in figure 5.11.

Figure 5.10 MoveIt planning command window output

Figure 5.11 The robot grasps and picks up the can
6.1 Conclusion

Through a long learning curve and after several trial and error experimentation the goals were achieved. The goal of the research was to automate the process of training robots in an assembly system with the help of vision guided intelligence and an open source development platform. Several attempts were made in each module in order to obtain the best results. The algorithms learned and developed were used in detection, recognition
and robot manipulation guided by a 3D sensor data. An object database was created that contained a CAD model of the OFI. An approach to use the CAD model to compare it with the processed point cloud to perform detection and recognition was introduced. The recognition presented is robust, fast and distance-sensitive in nature. The position of the object was calculated using open source libraries by manipulating the object frame, Kinect depth optical frame and the robot base frame transformation. The calculated position was a 6 DOF Cartesian co-ordinate matrices with respective time stamps. This data was taken as an input for the Baxter robot to plan an optimized path keeping in to account of parameters-plane normal, object centroid and major axis. Once the Cartesian path was created, it was transformed into the 7 dimensional planned robot joint values, to reach the object and manipulate it successfully, using parallel gripper actions.

6.2 Limitations

This research model has some of its limitation. To list them as follows:

1. At this point the Coke can is recognized only when the its presented upright. The reason for this limitation is the CAD model that is used, is inferior and does not contain surface normal information. Also images from various angles at different exposure can be taken and stored in the database to assist the 360 degree recognition.

2. The robot performs the grasping operation using the object’s centroid position, and does not have the capability to grasp it at some user specified or desired location.

3. The robot also does not take in to account the obstacles on its way during path planning process to perform the pick and place operation. Obstacle avoidance feature
is missing in this model.

6.3 Future Work

This work introduces several scopes for development and future work to advance robotics intelligence in industrial settings. Some topics of interest follows:

1. The linemod functionality of the PCL library can be used to perform a recognition while the object is in hand. At present, the linemod functionality starves for the system memory and hangs up the system. This can be improved by changing the software kernel properties. As a result the system memory starvation problem could be addressed and human to robot hand-off process could be executed.

2. The grasping operation can be made intelligent by using the fiducial information embedded on to the CAD model as a point for safe grasping. This will introduce a novel approach for handling of objects from human to robot.

3. Also, intelligence can be built into the robot to perform obstacle avoidance during robot arm manipulation operation.

4. The robot can be made mobile, but introducing wheeled platform instead of the standard fixed platform. That will bring into many more scopes of development of intelligence and features into the robot.
BIBLIOGRAPHY


<table>
<thead>
<tr>
<th>BIBLIOGRAPHY</th>
<th>BIBLIOGRAPHY</th>
</tr>
</thead>
</table>


Appendix A - Source Code

The code used for the project is given below.

///////////Cluster_Euclid.cpp///////////
#include <ros/ros.h>
// PCL specific includes
#include <pcl/io/pcd_io.h>
#include <pcl/console/time.h>
#include <pcl/point_types.h>
#include <pcl/kdtree/kdtree.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
//header for filtering
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/features/normal_3d.h>
//header for clustering purpose
#include <pcl/segmentation/conditional_euclidean_clustering.h>
//header for subscribing to kinect
#include <sensor_msgs/PointCloud2.h>
#include <pcl_conversions/pcl_conversions.h>
#include <pcl/point_cloud.h>
//header for model coeffs-->optional
#include <pcl/ModelCoefficients.h>
//header for segmentation
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/segmentation/extract_clusters.h>

ros::Publisher pub, pub_plane;

void cloud_cb(const sensor_msgs::PointCloud2ConstPtr& input){

    sensor_msgs::PointCloud2::Ptr clusters (new sensor_msgs::PointCloud2);
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>), cloud_f (new pcl::PointCloud<pcl::PointXYZ>);
    pcl::fromROSMsg(*input, *cloud);
    pcl::PointCloud<pcl::PointXYZ>::Ptr clustered_cloud (new pcl::PointCloud<pcl::PointXYZ>);
    std::cout << "PointCloud before filtering has: " << cloud->points.size () << " data points." << std::endl;

    // Create the filtering object: downsample the dataset using a leaf size of 1cm
    pcl::VoxelGrid<pcl::PointXYZ> vg;


pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_filtered (new 
  pcl::PointCloud<pcl::PointXYZ>); 
vg.setInputCloud (cloud); 
vg.setLeafSize (0.01f, 0.01f, 0.01f); 
vg.filter (*cloud_filtered); 
std::cout << "PointCloud after filtering has: " << cloud_filtered->points.size () << " data points." << std::endl; 

// Create the segmentation object for the planar model and set all the parameters
pcl::SACSegmentation<pcl::PointXYZ> seg; 
pcl::PointIndices::Ptr inliers (new pcl::PointIndices); 
pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients); 
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_plane (new 
  pcl::PointCloud<pcl::PointXYZ> ()); 
pcl::PCDWriter writer; 
seg.setOptimizeCoefficients (true); 
seg.setModelType (pcl::SACMODEL_PLANE); 
seg.setMethodType (pcl::SAC_RANSAC); 
seg.setMaxIterations (100); 
seg.setDistanceThreshold (0.02); 

int i=0, nr_points = (int) cloud_filtered->points.size (); 
while (cloud_filtered->points.size () > 0.3 * nr_points) 
{
  // Segment the largest planar component from the remaining cloud
  seg.setInputCloud (cloud_filtered); 
  seg.segment (*inliers, *coefficients); 
  if (inliers->indices.size () == 0)
  {
    std::cout << "Could not estimate a planar model for the given dataset." << std::endl; 
    break;
  }

  // Extract the planar inliers from the input cloud
  pcl::ExtractIndices<pcl::PointXYZ> extract; 
  extract.setInputCloud (cloud_filtered); 
  extract.setIndices (inliers); 
  extract.setNegative (false); 

  // Get the points associated with the planar surface
  extract.filter (*cloud_plane); 
  std::cout << "PointCloud representing the planar component: " << 
    cloud_plane->points.size () << " data points." << std::endl; 

  // Remove the planar inliers, extract the rest

}
extract.setNegative (true);
extract.filter (*cloud_f);
*cloud_filtered = *cloud_f;
}

// Creating the KdTree object for the search method of the extraction
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZ>);
tree->setInputCloud (cloud_filtered);
std::vector<pcl::PointIndices> cluster_indices;
pcl::EuclideanClusterExtraction<pcl::PointXYZ> ec;
ce.setClusterTolerance (0.02); // 2cm
cce.setMinClusterSize (10);
ce.setMaxClusterSize (2500);
ce.setSearchMethod (tree);
ce.setInputCloud (cloud_filtered);
ce.extract (cluster_indices);
std::vector<pcl::PointIndices>::const_iterator it;
std::vector<int>::const_iterator pit;
int j = 0;
for(it = cluster_indices.begin(); it != cluster_indices.end(); ++it) {
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_cluster (new pcl::PointCloud<pcl::PointXYZ>);
    for(pit = it->indices.begin(); pit != it->indices.end(); pit++) {
        cloud_cluster->points.push_back(cloud_filtered->points[*pit]);
    }
    *clustered_cloud += *cloud_cluster;
}
pcl::toROSMsg (*clustered_cloud , *clusters);
pcl::io::savePCDFileASCII (“record2.pcd”, *clustered_cloud);
clusters->header.frame_id = “/camera_depth_frame”;
clusters->header.stamp=ros::Time::now();
pub.publish (*clusters);
}

int main (int argc, char** argv)
{
    // Initialize ROS
    ros::init (argc, argv, “clust”);
    ros::NodeHandle nh;

    // Create a ROS subscriber for the input point cloud
    ros::Subscriber sub = nh.subscribe (“input”, 1, cloud_cb);
// Create a ROS publisher for the output model coefficients

pub = nh.advertise<sensor_msgs::PointCloud2> ("clusters", 1);

ros::spin();
}

////////////////////////BaxterMain.cpp////////////////////////

#include <BaxterControl/BaxterControl.h>   //header file
#include <object_recognition_msgs/RecognizedObjectArray.h> //using ORK
#include <object_recognition_msgs/TableArray.h> //using ORK
#include <visualization_msgs/MarkerArray.h>
#include <tf/transform_listener.h>
#include <tf/transform_broadcaster.h>

std::string Object_id = "";
double Object_assurance = 0;
geometry_msgs::PoseStamped Object_pose;
bool firstCB = false;

void objectCallback(const object_recognition_msgs::RecognizedObjectArray objects_msg) {
    double confident = 0;
    int id = -1;

    if (firstCB == false && (int)objects_msg.objects.size() == 1) {
        Object_id.assign(objects_msg.objects[0].type.key.c_str());
        firstCB = true;
    }
    for (int i = 0; i < objects_msg.objects.size(); ++i) {
        if (Object_id.compare(objects_msg.objects[i].type.key.c_str()) == 0) {
            if (objects_msg.objects[i].assurance > confident) {
                confident = objects_msg.objects[i].assurance;
                id = i;
            }
        }
    }
    if (id >= 0) {
        Object_pose.pose = objects_msg.objects[id].pose.pose.pose;
        Object_assurance = objects_msg.objects[id].assurance;
    } else {
        confident = 0;
    }
int main(int argc, char** argv) {
    ros::init(argc, argv, "Object_grabber");
    ros::NodeHandle nh;

    ros::Subscriber object_sub = nh.subscribe("/recognized_object_array", 1,
        &objectCallback);
    ros::Subscriber plane_sub = nh.subscribe("/table_array", 1, &planeCallback);

    BaxterArmCommander arm(nh);
    geometry_msgs::PoseStamped transed_pose;
    tf::TransformListener tf_listener;
    ros::Duration loop_timer(3.0);

    Object_pose.header.frame_id = "camera_depth_optical_frame";
    while (ros::ok()) {
        ROS_INFO("Arm is moved back");
        arm.ArmBack();
        ROS_INFO("Waiting for the object");
        if (Object_assurance > 0.8) {
            ROS_INFO("Best Similarity = %f ", Object_assurance);
            ROS_INFO("pose x is: %f", Object_pose.pose.position.x);
            ROS_INFO("pose y is: %f", Object_pose.pose.position.y);
            ROS_INFO("pose z is: %f", Object_pose.pose.position.z);

            bool tferr = true;
            while (tferr) {
                tferr = false;
                try {
                    tf_listener.transformPose("torso", Object_pose, transed_pose);
                } catch (tf::TransformException &exception) {
                    ROS_ERROR("%s", exception.what());
                    tferr = true;
                    ros::Duration(0.1).sleep();
                    ros::spinOnce();
                }
            }
            ROS_INFO("transformed Object pose x is: %f",
                transed_pose.pose.position.x);
            ROS_INFO("transformed Object pose y is: %f",
                transed_pose.pose.position.y);
            ROS_INFO("transformed Object pose z is: %f",
                transed_pose.pose.position.z);
        }
    }
}
ROS_INFO("Grab the Object!");

arm.rightMove(transed_pose.pose);

arm.grabObject(transed_pose.pose);
}

ros::spinOnce();
loop_timer.sleep();

return 0;
}

/////////////////////////////////////////////////ArmOperation.cpp/////////////////////////////////////////////////
#include <ArmOperation/ArmOperation.h>

ArmOperation::ArmOperation(ros::NodeHandle &nodehandle) : nh_(nodehandle),
  right_gripper(nodehandle), right_arm("right_arm"), left_arm("left_arm"),
  display_publisher(nh_.advertise<moveit_msgs::DisplayTrajectory>
  ("/move_group/display_planned_path", 1, true)) {

  ROS_INFO("Reference right arm robot frame: %s",
    right_arm.getPlanningFrame().c_str());
  ROS_INFO("Reference right arm end-effector frame: %s",
    right_arm.getEndEffectorLink().c_str());
  ROS_INFO("Reference left arm robot frame: %s", left_arm.getPlanningFrame().c_str());
  ROS_INFO("Reference left arm end-effector frame: %s",
    left_arm.getEndEffectorLink().c_str());
  right_arm.setPlanningTime(5.0);
  pick_offset << 0, 0, 0.3;
  hold_offset << 0, -0.2, 0.3;
  pre_grab_offset << 0, -0.2, 0;
  grab_offset << 0, 0, 0;

  right_arm_back_pose.position.x = 0.48336029291;
  right_arm_back_pose.position.y = -0.345984422306;
  right_arm_back_pose.position.z = 0.442497286433;
  right_arm_back_pose.orientation.x = 1;
  right_arm_back_pose.orientation.y = 0;
  right_arm_back_pose.orientation.z = 0;
  right_arm_back_pose.orientation.w = 0;

  left_arm_back_pose.position.x = 0.356899870469;
  left_arm_back_pose.position.y = 0.553228163753;
left_arm_back_pose.position.z = 0.333650371585;
left_arm_back_pose.orientation.x = 1;
left_arm_back_pose.orientation.y = 0;
left_arm_back_pose.orientation.z = 0;
left_arm_back_pose.orientation.w = 0;

global_pose_offset.position.x = 0;
global_pose_offset.position.y = 0;
global_pose_offset.position.z = 0;
global_pose_offset.orientation.x = 0;
global_pose_offset.orientation.y = 0;
global_pose_offset.orientation.z = 0;
global_pose_offset.orientation.w = 0;

global_joints_offset << 0,0,0,0,0,0,0;
right_gripper.set_mode(BaxterGripper::OPEN_CLOSE);
}

bool ArmOperation::rightArmBack() {
    ROS_INFO("Move right arm back");
    ROS_INFO("Release gripper");
    rightRelease();
    return rightMove(right_arm_back_pose);
}

void ArmOperation::rightExecute() {
    ROS_INFO("executing right arm plan");
    right_arm.execute(right_plan);
}

geometry_msgs::Pose ArmOperation::rightGetPose() {
    geometry_msgs::PoseStamped arm_pose = right_arm.getCurrentPose();
    geometry_msgs::Pose pose = arm_pose.pose;
    return pose;
}

Vector7d ArmOperation::rightGetJoints() {
    Vector7d joints;

    std::vector<double> group_variable_values;
    right_arm.getCurrentState()->copyJointGroupPositions(right_arm.
        getCurrentState()->getRobotModel()->getJointModelGroup(right_arm.getName()),
        group_variable_values);
    for (int i = 0; i < 7; ++i)
    {
        joints[i] = group_variable_values[i];
    }
void ArmOperation::rightGrab() {
    ROS_INFO("close gripper");
    right_gripper.open();
}

void ArmOperation::rightRelease() {
    ROS_INFO("open gripper");
    right_gripper.close();
}

bool ArmOperation::rightPlan(geometry_msgs::Pose pose) {
    ROS_INFO("requesting a cartesian-space motion plan");
    pose = addPose(pose, global_pose_offset);
    right_arm.setPoseTarget(pose);
    return right_arm.plan(right_plan);
}

bool ArmOperation::rightPlan(Vector3f plane_normal, Vector3f major_axis,
                              Vector3f centroid) {
    geometry_msgs::Pose pose;
    pose = normalToPose(plane_normal, major_axis, centroid);
    return rightPlan(pose);
}

bool ArmOperation::rightPlan(Vector7d joints) {
    ROS_INFO("requesting a joint-space motion plan");
    joints = joints + global_joints_offset;
    std::vector<double> group_variable_values(7);
    for (int i = 0; i < 7; ++i)
    {
        group_variable_values[i] = joints[i];
    }
    right_arm.setJointValueTarget(group_variable_values);
    return right_arm.plan(right_plan);
}

bool ArmOperation::rightPlanOffset(Vector3d offset) {
    geometry_msgs::Pose offset_pose = addPosOffset(rightGetPose(), offset);
    return rightPlan(offset_pose);
}

bool ArmOperation::rightMove(geometry_msgs::Pose pose) {
    if(rightPlan(pose)) {
        rightExecute();
    }
bool ArmOperation::rightMove(Vector3f plane_normal, Vector3f major_axis, Vector3f centroid) {
  if(rightPlan(plane_normal, major_axis, centroid)){
    rightExecute();
    return false;
  }
  return false;
}

bool ArmOperation::rightMove(Vector7d joints) {
  if(rightPlan(joints)){
    rightExecute();
    return true;
  }
  return false;
}

bool ArmOperation::rightMoveOffset(Vector3d offset) {
  if(rightPlanOffset(offset)){
    rightExecute();
    return true;
  }
  return false;
}

void ArmOperation::rightShowPath() {
  ROS_INFO("Visualizing planed Path");
  display_trajectory.trajectory_start = right_plan.start_state_;
  display_trajectory.trajectory.push_back(right_plan.trajectory_);
  display_publisher.publish(display_trajectory);
}

bool ArmOperation::leftArmBack() {
  ROS_INFO("Move left arm back");
  return leftMove(left_arm_back_pose);
}

void ArmOperation::leftExecute() {
  left_arm.execute(left_plan);
}

geometry_msgs::Pose ArmOperation::leftGetPose() {
  geometry_msgs::PoseStamped arm_pose = left_arm getCurrentPose();
}
geometry_msgs::Pose pose = arm_pose.pose;
return pose;
}

Vector7d ArmOperation::leftGetJoints() {
    Vector7d joints;
    std::vector<double> group_variable_values;
    left_arm.getCurrentState()->copyJointGroupPositions(left_arm.
        getCurrentState()->getRobotModel()->getJointModelGroup(left_arm.getName()),
        group_variable_values);
    for (int i = 0; i < 7; ++i)
    {
        joints[i] = group_variable_values[i];
    }
    return joints;
}

bool ArmOperation::leftPlan(geometry_msgs::Pose pose) {
    pose = addPose(pose, global_pose_offset);
    left_arm.setPoseTarget(pose);
    return left_arm.plan(left_plan);
}

bool ArmOperation::leftPlan(Vector3f plane_normal, Vector3f major_axis, Vector3f centroid) {
    geometry_msgs::Pose pose;
    pose = normalToPose(plane_normal, major_axis, centroid);
    return leftPlan(pose);
}

bool ArmOperation::leftPlan(Vector7d joints) {
    joints = joints + global_joints_offset;
    std::vector<double> group_variable_values(7);
    for (int i = 0; i < 7; ++i)
    {
        group_variable_values[i] = joints[i];
    }
    left_arm.setJointValueTarget(group_variable_values);
    return left_arm.plan(left_plan);
}

bool ArmOperation::leftPlanOffset(Vector3d offset) {
    geometry_msgs::Pose offset_pose = addPosOffset(leftGetPose(), offset);
    return leftPlan(offset_pose);
}
bool ArmOperation::leftMove(geometry_msgs::Pose pose) {
  if(leftPlan(pose)){
    leftExecute();
    return true;
  }
  return false;
}

bool ArmOperation::leftMove(Vector3f plane_normal, Vector3f major_axis, Vector3f centroid) {
  if(leftPlan(plane_normal, major_axis, centroid)){
    leftExecute();
    return false;
  }
  return false;
}

bool ArmOperation::leftMove(Vector7d joints) {
  if(leftPlan(joints)){
    leftExecute();
    return true;
  }
  return false;
}

bool ArmOperation::leftMoveOffset(Vector3d offset) {
  if(leftPlanOffset(offset)){
    leftExecute();
    return true;
  }
  return false;
}

void ArmOperation::leftShowPath() {
  ROS_INFO("Visualizing planed Path");
  display_trajectory.trajectory_start = left_plan.start_state_; 
  display_trajectory.trajectory.push_back(left_plan.trajectory_);
  display_publisher.publish(display_trajectory);
}

81
return false;
}

bool ArmOperation::grabObject(geometry_msgs::Pose Object_pose) {
  Object_pose.orientation.x = -0.708866454238;
  Object_pose.orientation.y = 0.17589525363;
  Object_pose.orientation.z = 0.135739113146;
  Object_pose.orientation.w = 0.669435660067;
  geometry_msgs::Pose grab_pose = addPosOffset(Object_pose, grab_offset);
  geometry_msgs::Pose pre_grab_pose = addPosOffset(Object_pose, pre_grab_offset);
  geometry_msgs::Pose hold_pose = addPosOffset(Object_pose, hold_offset);
  geometry_msgs::Pose pick_pose = addPosOffset(Object_pose, pick_offset);

  ROS_INFO("Move to hold pose");
  if (!rightMove(hold_pose)) { return false; }
  ros::Duration(0.5).sleep();
  ROS_INFO("Move to pre grab pose");
  if (!rightMove(pre_grab_pose)) { return false; }
  ros::Duration(0.5).sleep();
  ROS_INFO("Move to grab pose");
  if (!rightMove(grab_pose)) { return false; }
  ros::Duration(0.5).sleep();
  ROS_INFO("Grab the Object");
  rightGrab();
  ros::Duration(0.5).sleep();
  ROS_INFO("Move to pick pose");
  if (!rightMove(pick_pose)) { return false; }
  ros::Duration(0.5).sleep();
  ROS_INFO("Move to hold pose");
  if (!rightMove(hold_pose)) { return false; }
  ros::Duration(0.5).sleep();
}

Eigen::Affine3d ArmOperation::transformPoseToEigenAffine3d(geometry_msgs::Pose pose) {
  Eigen::Affine3d affine;

  Eigen::Vector3d Oe;

  Oe(0) = pose.position.x;
  Oe(1) = pose.position.y;
  Oe(2) = pose.position.z;
  affine.translation() = Oe;

  Eigen::Quaterniond q;
  q.x() = pose.orientation.x;
q.y() = pose.orientation.y;
q.z() = pose.orientation.z;
q.w() = pose.orientation.w;
Eigen::Matrix3d Re(q);

affine.linear() = Re;

return affine;
}

geometry_msgs::Pose ArmOperation::transformEigenAffine3dToPose(Eigen::Affine3d e) {
    Eigen::Vector3d Oe;
    Eigen::Matrix3d Re;
    geometry_msgs::Pose pose;
    Oe = e.translation();
    Re = e.linear();
    Eigen::Quaterniond q(Re);
    pose.position.x = Oe(0);
    pose.position.y = Oe(1);
    pose.position.z = Oe(2);
    pose.orientation.x = q.x();
    pose.orientation.y = q.y();
    pose.orientation.z = q.z();
    pose.orientation.w = q.w();

    return pose;
}

geometry_msgs::Pose ArmOperation::normalToPose(Vector3f plane_normal, Vector3f major_axis, Vector3f centroid) {
    geometry_msgs::Pose pose;
    Affine3d Affine_des_gripper;
    Vector3d xvec_des,yvec_des,zvec_des,origin_des;

    Matrix3d Rmat;
    for (int i=0;i<3;i++) {
        origin_des[i] = centroid[i];
        zvec_des[i] = -plane_normal[i];
        xvec_des[i] = major_axis[i];
    }

    yvec_des = zvec_des.cross(xvec_des);
    Rmat.col(0)= xvec_des;
    Rmat.col(1)= yvec_des;
Rmat.col(2)= zvec_des;
Affine_des_gripper.linear()=Rmat;
Affine_des_gripper.translation()=origin_des;

pose = transformEigenAffine3dToPose(Affine_des_gripper);
return pose;
}

google::protobuf::Pose ArmOperation::addPosOffset(google::protobuf::Pose pose,
const google::protobuf::Vector3d& offset) {
  google::protobuf::Pose result;
  result.position.x=pose.position.x+offset[0];
  result.position.y=pose.position.y+offset[1];
  result.position.z=pose.position.z+offset[2];
  result.orientation=pose.orientation;
  return result;
}

google::protobuf::Pose ArmOperation::subPosOffset(google::protobuf::Pose pose,
const google::protobuf::Vector3d& offset) {
  google::protobuf::Pose result;
  result.position.x=pose.position.x-offset[0];
  result.position.y=pose.position.y-offset[1];
  result.position.z=pose.position.z-offset[2];
  result.orientation=pose.orientation;
  return result;
}

google::protobuf::Pose ArmOperation::addPose(google::protobuf::Pose pose_a,
const google::protobuf::Pose& pose_b) {
  google::protobuf::Pose result;
  result.position.x=pose_a.position.x+pose_b.position.x;
  result.position.y=pose_a.position.y+pose_b.position.y;
  result.position.z=pose_a.position.z+pose_b.position.z;
  result.orientation.x=pose_a.orientation.x+pose_b.orientation.x;
  result.orientation.y=pose_a.orientation.y+pose_b.orientation.y;
  result.orientation.z=pose_a.orientation.z+pose_b.orientation.z;
  result.orientation.w=pose_a.orientation.w+pose_b.orientation.w;
  return result;
}

std::vector<double> ArmOperation::quat2euler(google::protobuf::Quaternion
const double mData[4];
std::vector<double> euler(3);
const static double PI_OVER_2 = M_PI * 0.5;
const static double EPSILON = 1e-10;

84
double sqw, sqx, sqy, sqz;

mData[0] = quaternion.x;
mData[1] = quaternion.y;
mData[2] = quaternion.z;
mData[3] = quaternion.w;

sqw = mData[3] * mData[3];
sqx = mData[0] * mData[0];
sqy = mData[1] * mData[1];
sqz = mData[2] * mData[2];

euler[1] = asin(2.0 * (mData[3] * mData[1] - mData[0] * mData[2]));
if (PI_OVER_2 - fabs(euler[1]) > EPSILON) {
                    sqx - sqy - sqz + sqw);
    euler[0] = atan2(2.0 * (mData[3] * mData[0] + mData[1] * mData[2]),
                    sqw - sqx - sqy + sqz);
} else {
                    2 * mData[0] * mData[2] + 2 * mData[1] * mData[3]);
    euler[0] = 0.0;

    if (euler[1] < 0)
}
return euler;

/////////////////////////////////////////////////////ObjectFinder.cpp/////////////////////////////////////////////////////

#include <ObjectFinder/ObjectFinder.h>
ObjectFinder::ObjectFinder(ros::NodeHandle &nodehandle): nh_(nodehandle) {
    Object_confidence = 0;
    firstCB = false;
    object_sub = nh_.subscribe("/recognized_object_array", 1,
                              &ObjectFinder::objectCallback, this);
    plane_sub = nh_.subscribe("/table_array", 1, &ObjectFinder::planeCallback,
                              this);
    Object_pose.header.frame_id = "camera_depth_optical_frame";
    Object_id = "";
}

void ObjectFinder::objectCallback(const
object_recognition_msgs::RecognizedObjectArray objects_msg) {
  double confident = 0;
  int id = -1;

  if (!firstCB && (int)objects_msg.objects.size() == 1) {
    Object_id.assign(objects_msg.objects[0].type.key.c_str());
    Object_pose.header.frame_id = objects_msg.header.frame_id.c_str();
    firstCB = true;
  }
  for (int i = 0; i < objects_msg.objects.size(); ++i) {
    if (Object_id.compare(objects_msg.objects[i].type.key.c_str()) == 0) {
      if (objects_msg.objects[i].confidence > confident) {
        confident = objects_msg.objects[i].confidence;
        id = i;
      }
    }
  }
  if (id >= 0) {
    Object_pose.pose = objects_msg.objects[id].pose.pose.pose;
    Object_confidence = objects_msg.objects[id].confidence;
  } else {
    confident = 0;
  }
}

bool ObjectFinder::getObjectPoseKinect(geometry_msgs::PoseStamped &Object_pose, double &confidence) {
  ros::spinOnce();
  if (Object_confidence > 0.8) {
    Object_pose = this->Object_pose;
    confidence = this->Object_confidence;
    return true;
  }
  return false;
}

bool ObjectFinder::getObjectPoseTorso(geometry_msgs::PoseStamped &Object_pose, double &confidence) {
  if (getObjectPoseKinect(Object_pose, confidence)) {
    //stuff a goal message:
    bool tferr = true;
    while (tferr) {
      tferr = false;
    }
  }
  return false;
}
try {
    tf_listener.transformPose("torso", Object_pose, transed_pose);
} catch (tf::TransformException &exception) {
    ROS_ERROR("%s", exception.what());
    tferr = true;
    ros::Duration(0.1).sleep(); // sleep for half a second
    ros::spinOnce();
}

Object_pose = transed_pose;

return false;