DEVELOPMENT OF GRASP LOCALIZATION ALGORITHM BASED ON DEEP LEARNING AIDED VISION SYSTEM

18MHP109L - MAJOR PROJECT REPORT

Submitted by

HARIKRISHNAN S (RA1811038010006)
GAUTHAM GANESH PRASAD (RA1811038010009)
P JEMUEL STANLEY (RA1811038010043)

Under the guidance of

Dr. SENTHILNATHAN, M.E., Ph.D.
(Associate Professor, Department of Mechatronics Engineering)

In Partial fulfilment for the degree

of

BACHELOR OF TECHNOLOGY

in

MECHATRONICS ENGINEERING WITH SPECIALIZATION IN ROBOTICS

of

COLLEGE OF ENGINEERING & TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

May 2022

BONAFIDE CERTIFICATE

Certified that this project report titled "***Development of grasp localization algorithm based on deep learning aided vision system***" is the bonafide work of "**HARIKRISHNAN S (RA1811038010006), GAUTHAM GANESH PRASAD (RA1811038010009),** and **P JEMUEL STANLEY (RA1811038010043)**, who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. SENTHILNATHAN, M.E., Ph.D.
Guide
Associate Professor
Dept. of Mechatronics Engineering

Dr. G. MURALI, M.E., Ph.D.
Professor & Head of the Department
Dept. of Mechatronics Engineering

Signature of the Internal Examiner

Signature of the External Examiner

ABSTRACT

The ability to grasp is a manipulator-robot's fundamental motor skill which describes the configuration of the end-effector in order to securely grasp an object without slippage. Although there have been advances in robot grasping, determining the best pose for unknown objects requires precise estimation of the object's pose; failure in doing so would lead to unstable grasp. Even with a well calibrated perception system, objects with curved and narrow surfaces or in cluttered environments can still be very challenging to pick.

In this project, the aim is to use a machine learning algorithm to perform a 7-DOF robotic grasp on novel objects. The grasp problem is divided into three sequential systems namely: computer vision, grasp detection and robot control. The computer vision system's key task is to use an RGB-D camera to perceive the robot's environment in order to attain the pose and 3D model of the object. This is a critical step as the subsequent steps rely on the pose provided during this phase. The grasp inferences are obtained via deep learning approaches. A 7-DOF collaborative manipulator is used to demonstrate the solution.

The project's outcomes with suitable modifications shall be applicable in various fields involving unstructured scenes, such as e-commerce, grasping related tasks in service robots, agricultural and food industries, etc.

ACKNOWLEDGEMENTS

It has been a great honor and privilege to undergo **B.Tech** in **MECHATRONICS WITH SPECIALIZATION IN ROBOTICS** at **SRM Institute of Science and Technology**. We are very much thankful to the **Department of Mechatronics**, **SRM Institute of Science and Technology** for providing all facilities and support to meet our project requirements.

We would like to thank our Head of Department **Dr. G. MURALI, M.E., Ph.D.** for providing us with this opportunity. We are highly indebted to our project guide **Dr. SENTHILNATHAN, M.E., Ph.D.** for his expert guidance and constant supervision throughout the project. We express our heartfelt gratitude to **Mr. R PRAKASH, Lab Asst.** for his timely support in completing the project.

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely fortunate that we have got this all along the completion of our project work. Whatever we have done is only due to such guidance and assistance and we would like to thank them for their kind support.

HARIKRISHNAN S

GAUTHAM GANESH PRASAD

JEMUEL STANLEY P

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CV	Computer Vision
CPU	Central Processing Unit
CoG	Centre of Gravity
Dex-Net	Dexterity Network
DL	Deep Learning
DoF	Degrees of Freedom
FOV	Field of View
GPU	Graphics Processing Unit
GQCNN-4.0-PJ	Grasp Quality Convolutional Neural Network 4.0 for Parallel Jaw
GQCNN	Grasp Quality Convolutional Neural Networks
HD	High Definition
IMU	Inertial Measurement Unit
IR	Infrared
PointNet GPD	Point Net Grasp Pose Detection
POMDM	Partially observable Markov Decision Process
ReLU	Rectified Linear Unit
SDK	Software Development Kit
TCP	Tool Center Point
TOF	Time of Flight

LIST OF SYMBOLS AND NOTATIONS

Q	Grasp quality metric
γ	Skew coefficient of the image acquired (intrinsic parameter
Ω	Grasp Axis orientation
$A \times B$	Cross product of vectors A and B
X, Y, Z	(x,y,z) coordinates in metric units
(i, j)	Centre of image coordinates in pixel frame
${}^{R}_{C}T$	Transformation of C w.r.t R

CHAPTER 1

INTRODUCTION

1.1 Background

While humans find it easy to manipulate items, properly grasping unknown objects remains a challenging task in the field of robotics. Finding a reliable solution will advance the field of assistive robotics, in which the robot interacts with its environment. This would also improve the usage of robots in industrial applications in tasks such as assembly, binning, and sorting. For this purpose, the robot decides how and where to grasp the object. In a structured environment, such as in an industrial setup, a systematic pattern of working with fixed timings, specific tasks and scenes (perceptive view and object size) are well defined. Hence, the robot is well aware of where and how the target object will be placed at an instant, which eliminates the need for an intelligent decision for grasp. However, a certain level of intelligence is required to plan grasps in unstructured environments where the robot has minimal knowledge of the environment and the target object. So, the robot needs to decide on the best grasp configuration for the robot to pick the object efficiently which is decided by certain parameters. For a successful grasp, the grasp should fulfill certain desired properties that indicate how well the grasped object resists external disturbances and settles to a stable state after the disturbances vanishes. Such quality measures are used to choose an optimal grasp using various analytical and empirical approaches, one of which is deep learning.

The proposed solution utilizes a deep learning strategy for identifying suitable grasp configurations from an input image. In the past decade, deep learning has achieved major success on detection, classification, and regression tasks. Its key strength is the ability to leverage large quantities of labelled and unlabelled data to learn powerful representations without hand-engineering the feature space. Deep neural networks have been shown to outperform hand-designed features and reach state-of-the-art performance.

1.2 Objectives

The primary objective of the project is to execute a deep learning inference-based grasp with vision data as the primary source of information.

1. To generate grasp candidates using Deep learning approaches.
   1.1 Image acquisition and processing for deep learning model
   1.2 Sending the images as compatible input to the deep learning model

1.3 Acquiring the inferences from the deep learning model

2. Execute the grasp on ABB YuMi robot based on the deep learning inference.
   2.1 Receiving inferences from the deep learning model
   2.2 Executing the planned trajectory for the grasp using the YuMi robot

1.3 Perspective of the Project

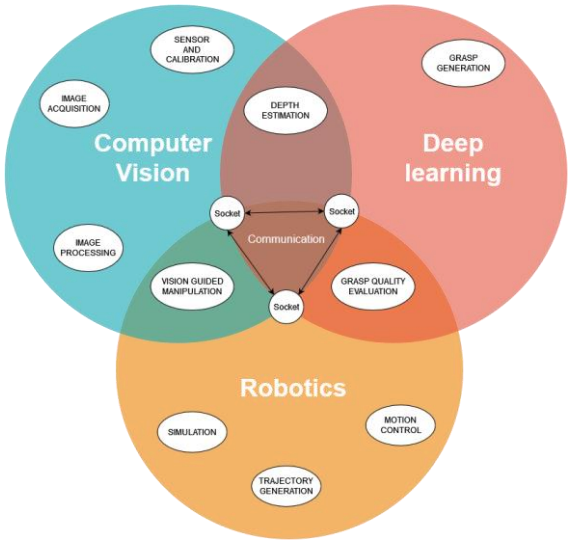


Fig. 1.1 Venn Diagram Representation of the Project

Mechatronics as a subject deals with the synergetic integration of different systems to provide useful functionality to the final model. The project when seen from a modularity aspect deals with three sub-systems namely computer vision, deep learning and robotics.

The project required building a setup to address certain limitations and features along with the mechanical properties of the robot and the gripper to pick the object of interest, which brings in the mechanical engineering aspect into the project. Being a pick-and-place operation, the trajectory planning of the robot introduces the motion control aspect of engineering. The computer vision system consists of a data acquisition system which provides information of the actual scene with reference to the camera. Establishing the communication pipeline lead to designing and studying the sub-systems from the point-of-view of integrating them, which is in correlation with the very definition of a mechatronics system.

1.4 Context and Motivation

- **Pre-trained deep learning networks**

  Given the limited duration of the project and availability of state-of-the-art trained networks, pre-trained model was chosen to continue with the project. It's

important to specify that the deep learning model does not track the object, hence the dynamic motion of the object is strongly discouraged.

- **Choice of image acquisition hardware**
    - Placement of camera: Based on the deep learning model's requirement, the camera is placed overhead at an optimal working distance.
    - Sensor resolution: Based on the deep learning model requirement.
    - Environmental conditions and object attributes: Since the hardware uses structured infrared light projection for disparity estimation, neither specular-reflective objects nor black coloured objects should be placed in the FOV of the camera. The proposed system is urged to be strictly used indoors for the same reason.

- **Gripper type and payload constraints:**

    The choice of parallel jaw gripper is convenient because the chosen deep learning network is trained for the same. Although the gripper is designed with a maximum payload capacity of 500g, the actual capacity depends on certain factors such as texture and CoG of the targeted object. The gripper also has a maximum extendibility of 50 mm making it vital for object segregation based on these above-mentioned constraints.

- **Choice of robot arm:**

    Since the 7-axis redundant robot offers more manipulable regions, it facilitates the placement of objects across a wide range of the robot's workspace. Even though a dual-arm grasp can be considered, the project proposes neither a cooperative grasp nor simultaneous dual-arm grasps for multiple objects due to the project duration and choice of deep learning network being pre-trained in nature.

- **Choice of software platform**

    Apart from the robot motion control which is coded on RAPID (High-level language used to control ABB industrial robots), the entirety of the software development is implemented on python to provide a common ground for the vision and deep learning systems, while at the same time accounting for the various library dependencies on python. For instance, the image acquisition software (Libfreenect) has

a dependency on python to work so, consequently python was chosen as a common framework.

1.5 Contribution and Significance

Since the project team was the first batch of students working with the ABB YuMi robot after its commissioning, every aspect of development and methods used for troubleshooting would serve as a valuable material for all future projects.

Having established a python-interface between the sub-systems and built a ready-to-go vision setup, will enable future collaborators to progress forward without having to deal from the first principles.

Robot manipulation and communication-related malfunctions have also been well documented with causes and recommended solutions which will serve as a first-hand guideline and reference for future troubleshooting.

The installation prerequisites for running the pre-trained Deep Learning model (GQCNN) have been structured as a primer to ease future collaboration and prioritize other aspects of the project (assuming the DL network is fixed) like transfer learning, vision-guided manipulation, and grasp localization.

1.6 Overview of Project Activities

In Deep learning two networks were finalized based on the type of input data one being point cloud data (Contact Graspnet [3]) and the other which is based on raw depth image (GQCNN [2]) and a literature review was done to understand the architecture, learning policy, and type of parent dataset. Both the networks were modified for inference and suitable code handling was done to store the inference in respective directories. The GQCNN code was modified to interface it with the Kinect and a separate module was created to send the grasp inference candidates to the python server through socket communication. A set of sample objects were chosen and the performance of the network was evaluated to compare it with the computer vision approach.

Literature survey on depth image acquisition and imaging hardware for the project was conducted as part of the pilot studies for the computer vision subsystem. A suitable camera sensor was chosen after determining the minimum working distances for the sensors available for the project. A black box model was made to establish a relationship between the depth intensity values and the actual depth measures and a regression model was used to find the

coefficients of the polynomial. The chosen camera sensor was mounted using a rigid mount at a location and pose considering all the constraints. The camera sensor was calibrated for its intrinsic and extrinsic parameters. The depth image acquired using the supported python libraries for the chosen camera sensor was used and its depth intensity values were converted to actual depth measures (requisite for the deep learning model) using the black box model. A segmentation mask was generated using the acquired depth image (another requisite for deep learning model). A computer vision approach solution to grasp localization was built to compare performance with the deep learning model.

Being the first student team to use the robot, manuals provided by ABB was referred time-to-time throughout the project. Once the necessary safety protocols and operating instructions were well understood, basic motion programs were executed using the FlexPendant. RAPID technical reference for RobotWare 6.13 was referred thoroughly for the pick-and-place routine using RAPID. Various move commands were executed simultaneously in order to understand the nuances. Although, YumiPy [49] (python-interface for the robot by Berkeley Autolab) was attempted, its failure lead to python and RAPID-based socket communication pipeline development between the robot and a remote PC. Initially, the robot acted as the server while the remote PC was the client. However, due to the introduction of multiple clients, the need for a python-based centralized server was recognized and hence the same was implemented in later stages of the project. Parallel to the online tasks, kinematic understanding of the robot was gained in simulations on MATLAB which included reachability, manipulability and comparison between the physical robot and the simulated model. Calibration of the arm and TCP was performed whenever certain discrepancies were noticed. Firmware updates related to RobotWare and SmartGripper Add-in were also performed.

1.7 Project Overview

The grasp localization problem is addressed with 3 subsystems namely: computer vision, deep learning, and robotics. The setup consists of an overhead Kinect camera, the ABB YuMi IRB 14000 dual-arm collaborative robot, and two computer systems. The acquired depth frame is processed simultaneously to generate a segmentation mask for the placed object and the actual depth values of the scene are sent as inputs to the deep learning model. The model, with these two inputs, generates grasp candidates and ranks them based on certain grasp quality measures. The best grasp is hence chosen and the pose is communicated to the robot for performing the grasp routine.

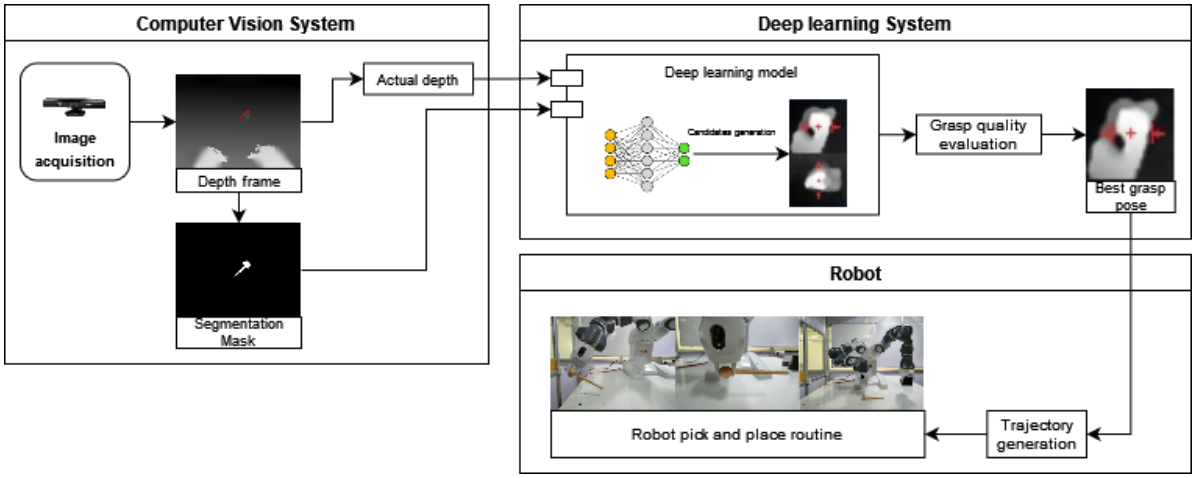


Fig. 1.2 Schematic Diagram of the Setup

1.8 Software Stack

The software stack illustrated in **Fig. 1.3** is organized based on the sub-systems and the functional components. This software stack is a collection of independent components that work together to support the execution of this project. The operating system used for deep learning and computer vision was Ubuntu 18.04 LTS, while the robot runs on RobotWare v6.13 OS. Python language was used primarily in the anaconda environment for deep learning, computer vision, and socket communication. RAPID language was used in the RobotStudio environment for motion-control commands and communication on the robot end. Libraries and tools stand below this layer that encapsulates various functions to be used in the software program. A communication layer exists between the sub-systems that are responsible for data transfer in the appropriate data format.

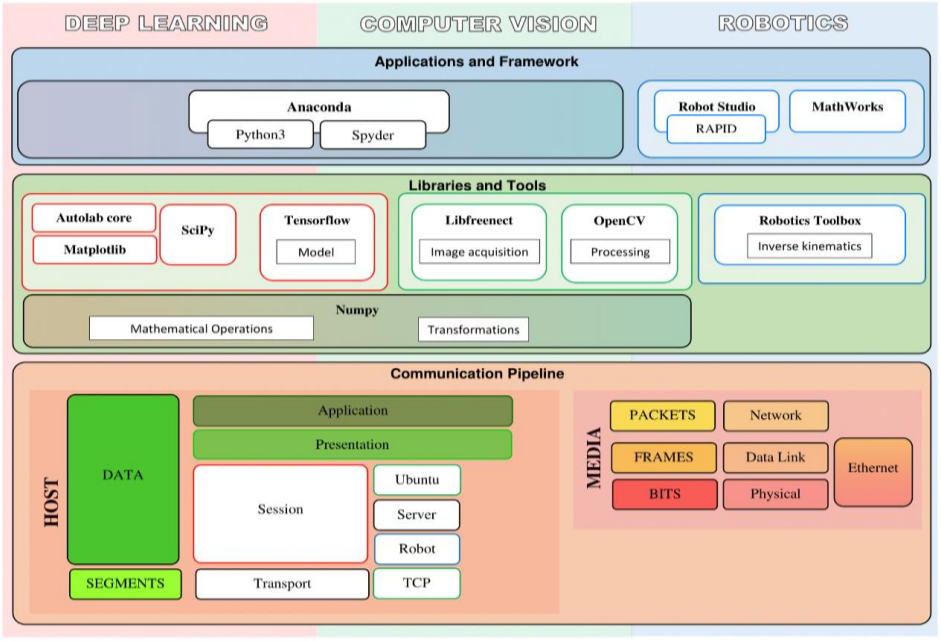


Fig. 1.3 Software Stack

1.9 Organization of the Report

The results of the project work have been compiled into a report as detailed below:

Chapter 1 provides the background information of the development of a grasp localization algorithm based on deep learning aided vision system, the project objectives, the perspective of the project, the context behind the project (choice of network, gripper type and limitation, choice of software platforms), the contribution and significance of the project, and an overview to the entire project along with the description of the software stack used in the project.

Chapter 2 describes the literature review on the topics required for the subsystems of the project i.e., computer vision, deep learning and robot control subsystems.

Chapter 3 gives an in-depth description on the computer vision subsystem – the literature survey, experiments to choose the best imaging hardware and for other processes required in the subsystem, building computer vision solution to the grasp localization to compare the performance of the project system with.

Chapter 4 gives a detailed description of the deep learning subsystem – the literature survey, experimenting with different pre-trained deep learning networks, the changes to the network code for the project requirements, and analysis of grasping policies and grasp representations.

Chapter 5 introduces the robot used in the project. It gives an overview of the operational, technical, and applications which every user needs to be aware of. Activities performed using the project with the robot have also been detailed. Finally, the rigid body transformations involved in the project are equated.

Chapter 6 delivers information regarding the integration and establishment of communication between the different subsystems.

Chapter 7 provides the conclusion and future scope of the project.

CHAPTER 2

LITERATURE REVIEW

A detailed literature review was conducted to understand the existing systems and approaches. For deep learning literature review was required to understand the type of input data, and datasets as well as to understand what network parameters will suit the purpose (in the project parallel jaw gripper is used). Learning to enhance the quality of depth image, point cloud reconstruction techniques, and understanding the software aspect of different sensors was paramount for computer vision. Technical, operational, and application manuals provided by ABB were thoroughly referred to before and while using the robot for perennial operational guidance. The review was essential to understand the manipulability and dexterity of the robot arm. Since every robot has certain limitations in its workspace, quantitatively analysing it was essential for gaining insights into the placement of the object. The sources were mainly published articles, journals, conference papers, and manuals.

2.1 Deep Learning

In a recent paper "**A Survey on Learning-Based Robotic Grasping**" **[1]** the authors gave a comprehensive overview of existing terminologies in the field of vision-based robotic grasping and manipulation. It also focuses on the categorization of approaches and the transfer problem of simulation to real-life scenarios. Additionally, in the paper named "**Learning ambidextrous robot grasping policies [2]** a model-free (no prior assumptions about object's geometry), deep learning approach was proposed to predict the probability of success of candidate grasps from images. The training process is modelled as a POMDM where the robot tries to maximize the reward (probability of grasp success) provided with imperfect state observations of the environment. The network is trained specifically for different types of grippers i.e., (parallel-jaw, suction) to make the grasp approach specific and the inputs to the network are depth values of the scene, segmentation mask, and the camera intrinsic parameters. The dataset used to train the network is a popular open-source research project by Berkeley automation known as DexNet 4.0, (which is the fourth iteration of the revised sequence for the purpose of grasp localization). On the other hand the paper "**Contact-GraspNet: Efficient 6-DoF Grasp Generation in Cluttered Scenes" [3]** talks about a model based grasping which circumvents reasoning about the physics of contact and grasp generation by pre-defining a set of grasps in the object frame and transform those grasps according to the 6-DoF object pose

exclusively for a cluttered environment. The dataset used here is ACRONYM and Shape Net. The inputs to the network are depth values of scene, segmentation mask based on point net, K (camera intrinsic parameters), RGB image **OR** point cloud data xyz (meters) with corresponding *XYZ* colour keys.

2.2 Computer Vision

The calibration model suitable for software-based calibration of Kinect-type RGB-D sensors was proposed in **Calibration of Kinect-type RGB-D sensors for robotic applications [4].** Additionally, it describes a two-step calibration procedure assuming a use of only a simple checkerboard pattern.

A depth image denoising and enhancement framework using a light convolutional network to address the depth frames of poor quality acquired using different camera sensors was proposed in **Fast depth image denoising and enhancement using a deep convolutional network [5].** Similar networks and other approaches have been reviewed for this purpose (**[6]**,**[7]**,**[8]**,**[9]**).

Point cloud approach was also reviewed to further the scope of the quality of input and types of grasp generation networks. For a multiple camera setup to acquire point cloud for the object of interest so as to generate a depth frame from it and/or to use a network to generate grasp candidates using point cloud was proposed in **Calibration of Multiple Kinect Depth Sensors for Full Surface Model Reconstruction [10]**.

The documentations referred for the computer vision subsystem were **Microsoft Kinect v1 manual, Kinect SDK documentation (**Kinect SDK developer toolkit for acquiring different kind of data using Kinect)**, PyKinect documentation (**Kinect python library)**, Intel Realsense D435i SDK documentation, Stereolabs Zed Mini SDK documentation** and **OpenKinect Libfreenect documenatation (**Kinect python library)**.**

2.3 Robotics

An important aspect of workspace analysis is finding the areas of good and reduced manipulability, for which one such measure is the Yoshikawa's manipulability measure that describes how close the robot is to a singular condition. Over the years, it has been proven to be beneficial for design and control of robots during task planning to have a quantitative measure of the robot arms in positioning and orienting the end-effector. Furthermore, manuals provided by ABB was used as a constant reference throughout the project.

CHAPTER 3

COMPUTER VISION

3.1 Description of the Tasks in CV

The computer vision subsystem provides the appropriate input for the deep learning model to generate the grasp candidates. The following are the aspects that had to be addressed and the tasks involved to build a suitable computer vision system for the project:

1. The vision system should address the requirements for the deep learning model, which consists of perspective for the camera (of the object to be grasped – overhead view), spatial resolution of the object in the image, the size of the image (640 × 480), quantization of the image and the kind of imaging modality (infrared image).
2. Testing to find the appropriate camera sensor for the subsystem (which addressed the requirements and had python supported libraries) and finding various ways to improve the quality of the acquired depth image.
3. Establish relationship between actual depth measures to the depth intensity acquired using the sensor to later convert the depth intensity map to actual depth values image.
4. After addressing the sources of noise and potential occlusions, mount and calibrate the camera in the found location.
5. Acquire the depth images using python libraries, as the entire project is mostly based on python language.
6. Process the depth intensity map into an actual depth value image and generate segmentation mask which are required as inputs to the deep learning subsystem.
7. Build a computer vision-based grasp localization solution to compare with the results of the deep learning model.

In the following subsections, all the steps and procedures involved in the tasks mentioned above have been discussed in detail with the pictorial and graphical analysis necessary.

3.2 Scene Constraints

As the deep learning model used in the project is a pretrained network, the input to the network must be similar to the images used to train it to obtain better results. These constraints were made note of from the documentations on the GQ-CNN deep learning network and the dataset used to train the model, which is Dex-Net dataset [12].

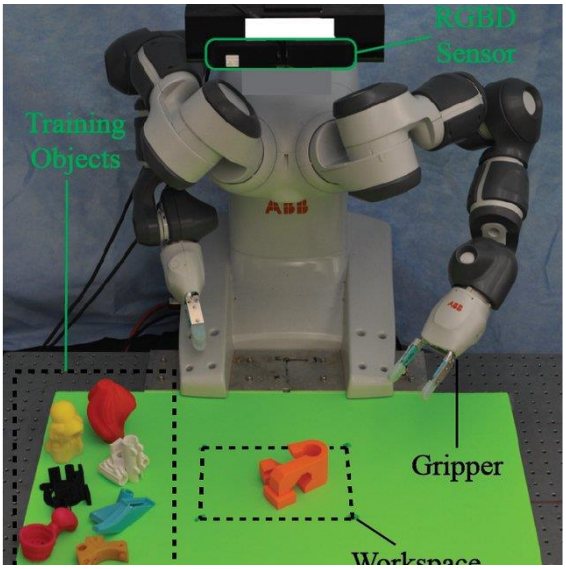


Fig. 3.1 Dex-Net Setup with Overhead Camera [12]

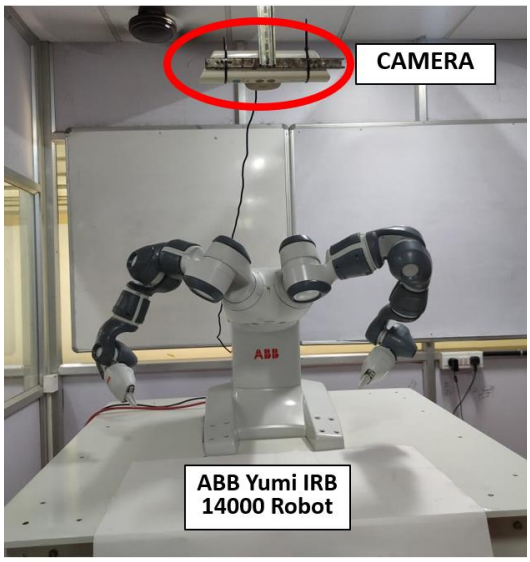


Fig. 3.2 Current Project Setup with Overhead Camera

One of those constraints was the overhead position of the camera. From **Fig. 3.1**, it can be observed that the depth images were taken in that position for the dataset and hence had to be followed for reliable inference (refer the project setup in **Fig. 3.2**).

The field of view and the spatial resolution of the object in the image had to be comparable with the images in the dataset. It can be compared using **Fig. 3.3** and **Fig. 3.4**, the spatial resolution of the object in the depth image is comparable between the Dex-Net dataset and the images acquired using Kinect.

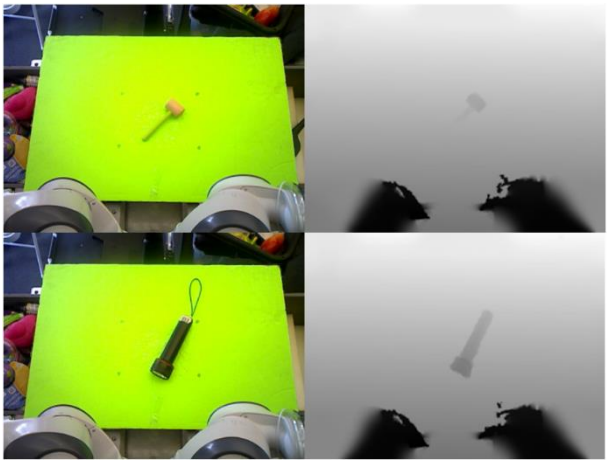


Fig. 3.3. Dex-Net Dataset RGB and Depth Frame [12]

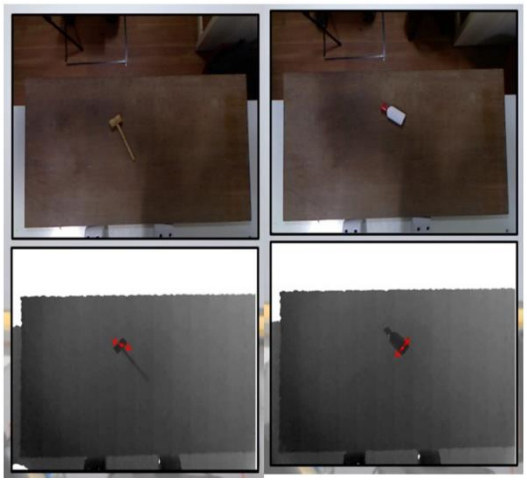


Fig. 3.4. Project Inferences RGB and Depth Frame

As the depth image is acquired using infrared imaging, the possibilities of noise such as sunlight and potential occlusions such as the position of the robot arm occluding the scene had to be addressed. The project is setup indoors (refer to **Fig. 3.2**), thus eliminating the possibility of noise created by sunlight. The texture of the object to be grasped could also potentially cause noise. Reflective surfaces and black-coloured objects affect the infrared depth image. The possibility of robot arm occluding the view has been addressed in the robot control subsystem. The arms are kept away from the FOV for both Dex-Net setup and in the project setup (refer **Fig. 3.1** and **Fig. 3.2**).

One of the other scene constraints is the workspace, only a well-defined part of the table on which the ABB YuMi robot has been placed is being used to detect the object to be picked. As observed in **Fig. 3.3** and **Fig. 3.4**, the green background and the brown board used in the project setup defines the maximum scope of the workspace. Additionally, some part of this defined workspace is blocked while or after acquisition to address the problem of acquiring depth images with objects out of the workspace and is later useful in the generation of the segmentation mask to localize the appropriate object in the depth image shown in the **Fig. 3.5.**

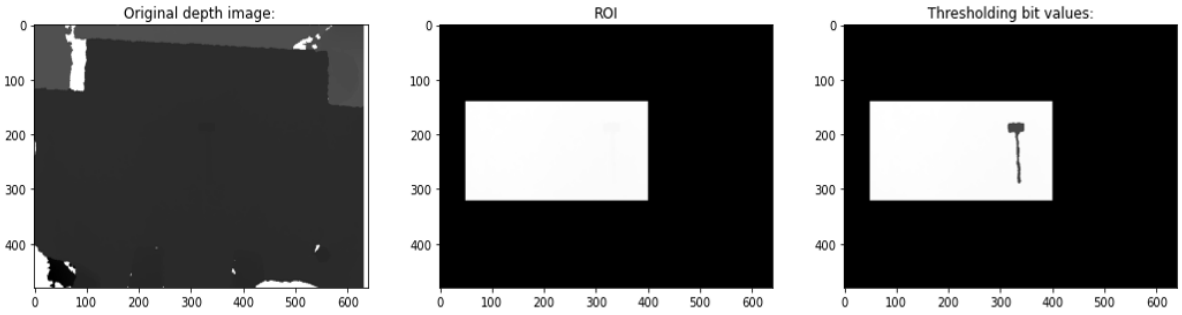


Fig. 3.5 Depth Intensity Image (left), Region of Interest (middle), Thresholding for Segmentation (right)

As the camera is placed considering the minimum working distance and the appropriate perspective, the camera is not moved at any point in the project. It is then calibrated in that position and further used for the intended purpose. Another constraint with respect to the minimum working distance, is objects of height (dimension of the object along the optical axis or camera *z*-axis) 8 cm and above cannot be used as it falls lower than the lower limit of the working distance and causes noise for that area of the object above 8cm (would be shown in white in the depth image). This can be observed in **Fig. 3.6**, where the top part of a paper cone becomes white and when converted to a segmentation mask, it neglects that part of the object as background or not part of the object. The depth image is not shown as the higher quantization of the image would not be visually comprehensive.

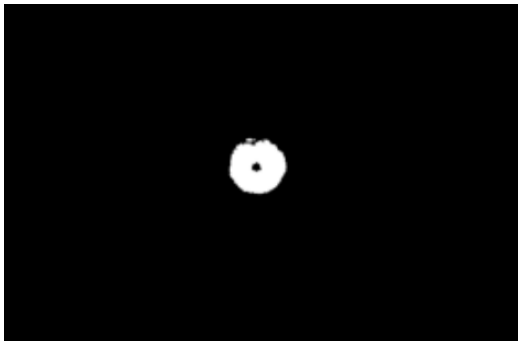


Fig. 3.6 Segmentation Mask for Cone Exceeding Working Distance

3.3 Imaging Hardware

The DexNet dataset has used Primesense Carmine 1.09 which is a short-range 3D camera for short-range scanning and for small objects (**Fig. 3.7**). As the camera was meant for short range and small objects, it could resolve depth much higher than the cameras available for the project. Hence the camera sensor which gave the best results was chosen for the project. The imaging hardware used in the project is shown in **Fig. 3.7**.

Fig. 3.7 Primesense Carmine 1.09 (top left), StereoLabs Zed Mini (top right), Intel Realsense D435i (bottom left), Microsoft Kinect v1 (bottom right)

The size of the image acquired using the Primesense Carmine camera was 640 × 480, and the deep learning network also mandates that as a requisite.

In order to achieve reliable inferences from the deep learning model, the input to the model must be as similar to the dataset used to train it as possible. Hence the choice of camera sensor must justify the qualities of Primesense Carmine 1.09 to maximum extent. The camera sensors available for the project team that could be used for the project were Microsoft Kinect v1 sensor, Intel Realsense D435i and StereoLabs Zed Mini.

A certain protocol was made to systematically analysis and determine which sensor was the closest analog to the Primesense Carmine 1.09 sensor. The following is the protocol:

- Quality of the depth images acquired using the SDK – the quality was assessed based on the amount of noise generated by the camera in the depth image, minimum and maximum working distances to achieve comparable the spatial resolution of the object

of interest and the depth resolution (visibility of the contour of the object in the depth image). The primary assessment of the sensor is made using the SDK as most of the limitations of the sensor could be identified directly using the SDK. It is also faster to implement and test than to test using the corresponding python libraries for the sensor.

- Depth images acquired using Python libraries – The camera sensors which showed promising results were further tested based on the same quality measures used earlier for the depth images acquired using the python library for that sensor.
- Experiment and determine the working distance – The camera is tested for ranges of 10cm which is determined based on the possible mounting restrictions in the laboratory near the robot (for example, Intel Realsense was tested for ranges of 20-30cm, 30-40cm and 40-50cm).
- Depth images tested on the deep learning model – The depth images acquired using the camera sensors that has succeeded in the previous steps are sent as one of input to the deep learning model along with the segmentation mask.

Described below are the analysis for each camera sensor using the forementioned protocol:

- StereoLabs Zed Mini – This is a stereo camera which also has an IMU used for depth sensing and many other applications created by StereoLabs [14]. The sensor could not acquire depth image for the order of the objects used for the project (small objects which can be picked using 5cm wide grasp – gripper limitations) that satisfies the quality measures used for the first step of the protocol. It was tested in various distances and failed to provide suitable depth image. Hence this sensor was not further experimented or tested with. As observed in the **Fig. 3.8**, the depth frame for the object (mallet) contains noise and is not reliable. In **Fig. 3.9**, the reconstructed depth image contain noise that would affect segmentation and it is not comparable with Primesense Carmine that would affect the inference from the deep learning model.

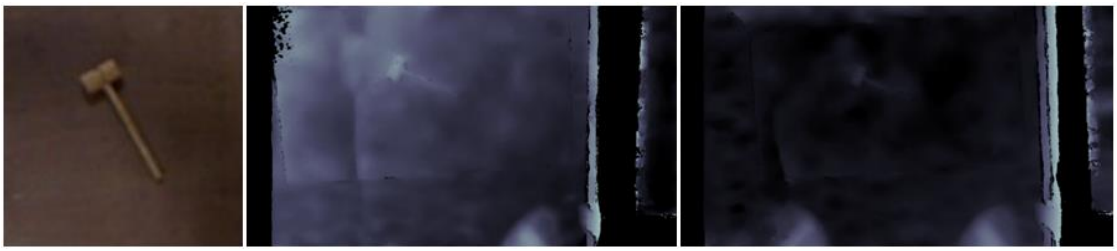


Fig. 3.8 Object Used –Mallet (left), Depth Frame at 56cm (middle), Depth Frame at 90cm (right)

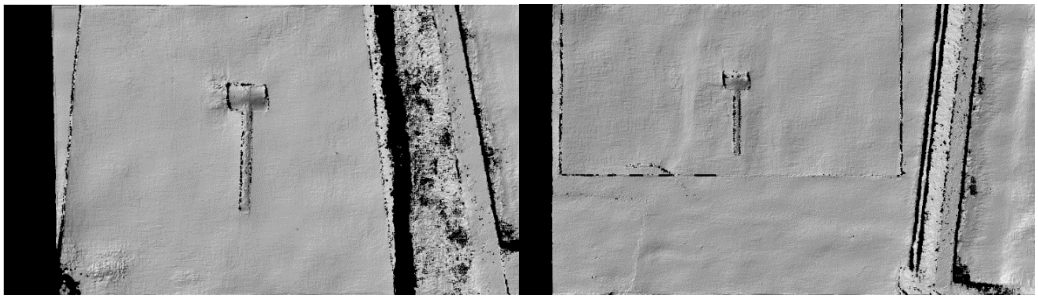


Fig. 3.9 Reconstructed Depth Image for Object at 30cm from Camera (left), Reconstructed Depth Image for Object at 90 cm from Camera(right)

- Intel Realsense D435i – This is a stereo camera which has IMU used for depth sensing applications and many more. The SDK for the camera applied filters on the depth image such as hole filling, gaussian blur, temporal filter, and many other filters, hence making it harder to determine the actual depth image from the sensor but it showed promising results. The sensor was tested for the further steps: acquiring depth images using its python library and checking for suitability. The sensor was tested for various working distances based on repeated trials and then narrowed down to a range which was then divided into ranges of 10cm to accurately determine the best working distance. These tests were done for the small objects which are suitable for the project. The final ranges were 20-30cm, 30-40cm and 40-50cm (as in **Fig. 3.11**) where the camera gave better results for 25cm in the 20-30cm range (as in **Fig. 3.10**). The result was better than StereoLabs Zed Mini, but the quality measures were not satisfactorily addressed. Contrast stretching was done on the acquired depth images to comprehend those as the images were of higher quantization (as in **Fig. 3.11**).

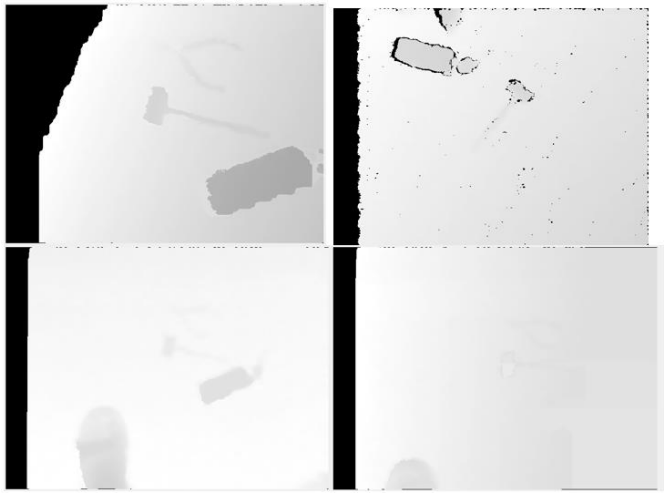


Fig. 3.10 Depth Image for Objects at 25cm (top left), Depth Image of Objects without Hole Filling Filter at 35cm (top right), Depth Image of Objects at 42cm (bottom left), Depth Image of Object at 45cm (bottom right)

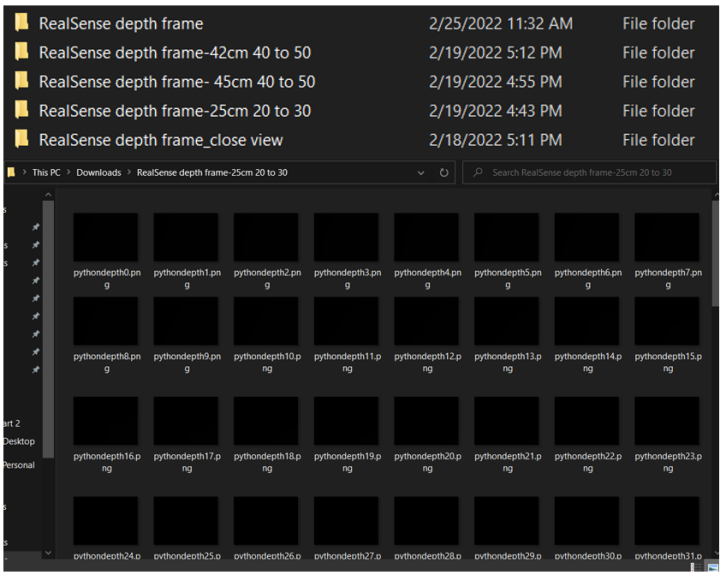


Fig. 3.11 Working Distance Experiment for Intel Realsense D435i

- Microsoft Kinect v1 – The sensor consists of an RGB camera and infrared emitter and receiver which map the depth by projecting structured infrared light. The camera was first tested using its SDK as per the protocol, and another point cloud generation was also tested to explore the generation of depth image from the point cloud data. Upon on satisfactory results, depth images were acquired using its python library to further confirm its quality. The working distance was deduced by repeated trials and showed promising results for about 75-95cm for the order of objects used in the project. The distance was set at 85cm overhead the workspace table, and the deep learning model was able to generate reliable inferences for those depth images.

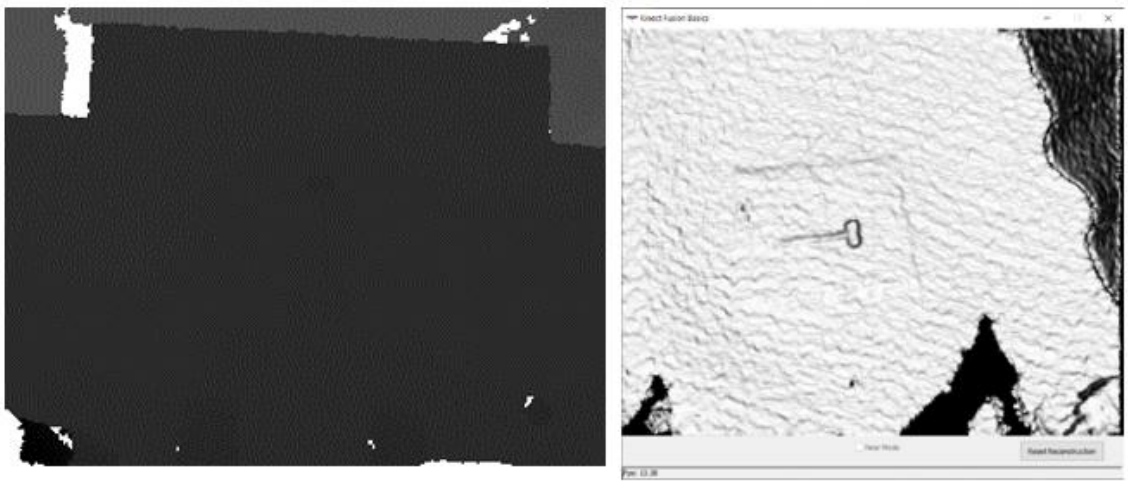


Fig. 3.12 Depth Image Acquired using Python Library(left), Reconstructed Depth Image using SDK (right)

3.4 Image Acquisition Mode

Two approaches were tested to acquire better depth images: using single camera sensor and using multiple camera sensor.

Single camera setup: The single-camera setup was made based on the scene constraints and other restrictions, and the depth images acquired directly from the sensor (Kinect sensor) were tested in the network (GQ-CNN). The methods to improve the quality of depth images acquired using a single camera setup were explored. The hardware and the protocol mentioned above were used for the initial acquisition, later further methods were explored to improve the depth images. A survey was made on those various approaches [13]. Few deep learning methods for denoising and enhancing the resolution of the depth images were evaluated on. The generation of depth image using point cloud data was tested using MATLAB, but a single Kinect sensor kept at the working distance (85cm approx.) could not project enough points for the object point cloud data of interest.

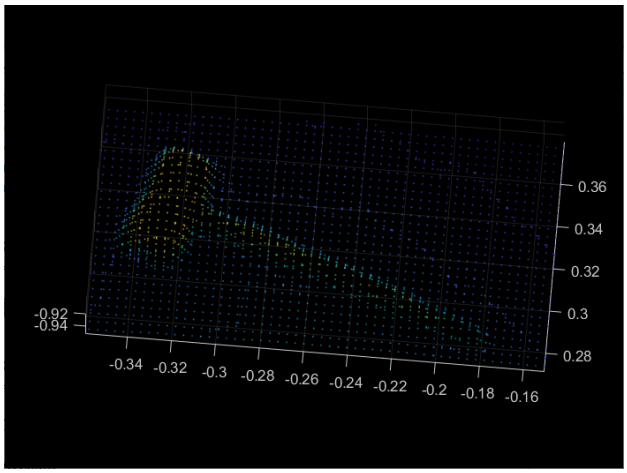


Fig. 3.13 Portion of the Point Cloud Acquired using Kinect SDK with the Mallet about 80 cm away from the Camera

Multiple camera setup: As the generation of point cloud using single Kinect sensor was insufficient to create depth image of the required resolution, multiple point cloud acquired using multiple Kinect sensors (2 Kinect sensors) were acquired separately to avoid the interference of the IR projection caused by the two Kinect sensors when used simultaneously for acquisition. Attempts on merging the two-point clouds based on feature matching failed again due to the lack of point data for the objects point cloud (as in **Fig. 3.14**) isolated from the background noise using limits in the three dimensions of the generated point cloud data. Although the point clouds acquired with the Kinect sensor in very close proximity to the object of interest (with the least amount of Infrared projection noise) is yet to be tested and could possibly produce better point cloud data for the objects used in this project. The success in the

acquisition of compatible depth images using the Kinect sensor led to the further progress of the project using the depth images acquired using the python libraries without using any other supplements (enhancement, denoising, etcetera).

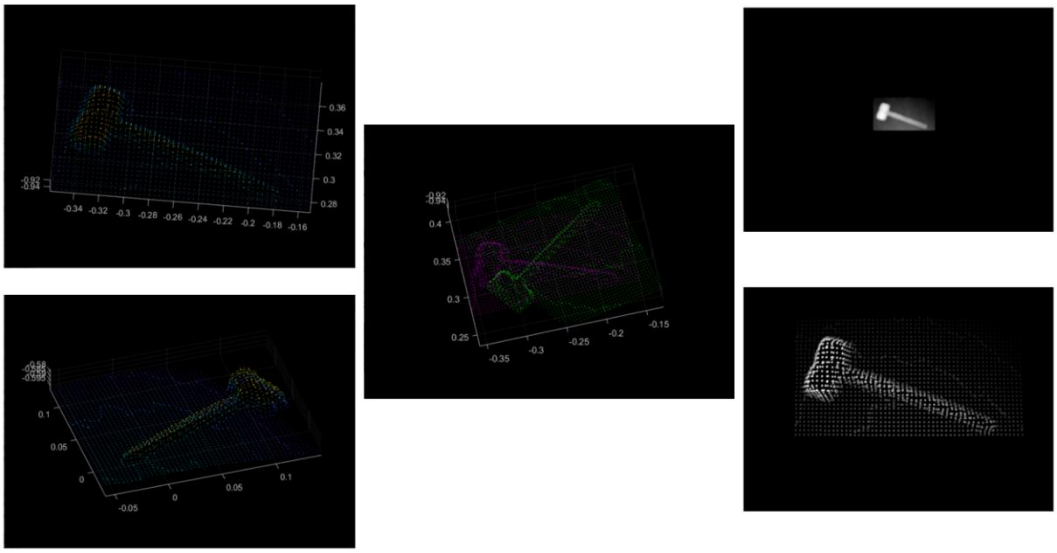


Fig. 3.14 Portion of the Point Cloud Acquired by Separate Kinect Sensors at Different Perspectives (left top and bottom), Failed Matching Point Clouds based on Features (middle), Generated Depth Image using One Point Cloud Data (right top), Generated Depth Image with Camera Height Reduced (right bottom)

3.5 Software Libraries

In the first step of the protocol to determine the suitable camera sensor, the SDKs for the corresponding cameras were used:

1. StereoLabs Zed Mini SDK – provided many features for recording and changing the resolution of the depth image acquired. The depth images acquired using the sensor worked well for larger objects but failed to produce suitable results for the objects used in the project. Hence this camera sensor was not further explored. The SDK user-interface (UI) is shown in **Fig. 3.15** [14].

Fig. 3.15 StereoLabs Zed Mini SDK Multiwindowed UI (left), StereoLabs Zed Mini SDK Depth Viewer (right)

2. Intel Realsense D435i SDK – the SDK provided an intuitive camera calibration app and an app to acquire depth images of various image sizes and resolution with the choice of applying various post processing filters onto the depth image. The SDK user-interface (UI) is shown in **Fig. 3.16** [15].

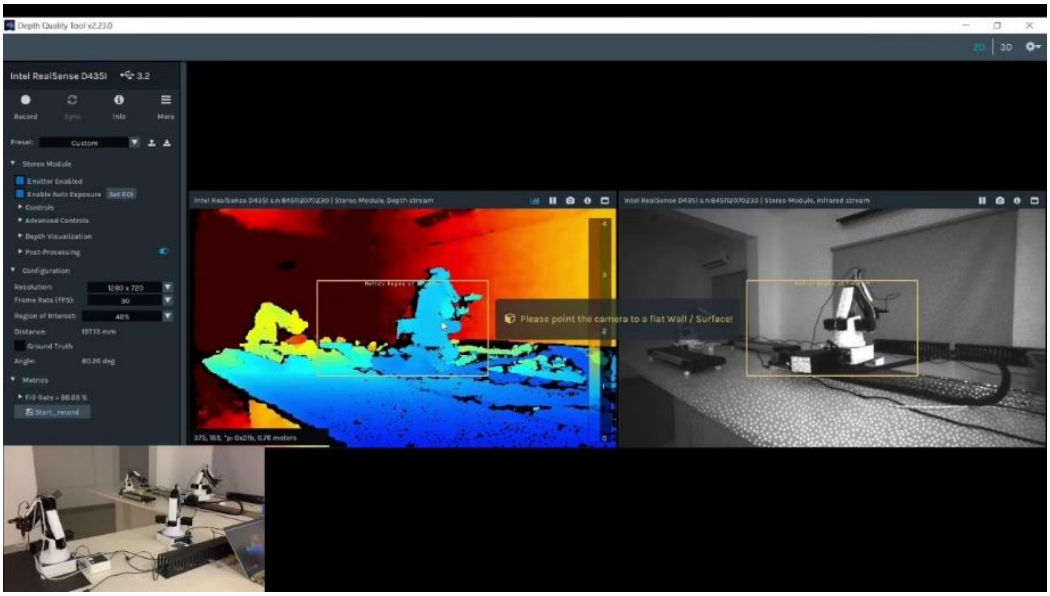


Fig. 3.16 Intel Realsense D435i SDK UI – Depth Viewer

3. Kinect Studio SDK – the SDK provided numerous apps for various kinds of image and data acquisition using Kinect sensor. The Depth Basics – WPF application was used to initially test the depth images using Kinect. The Kinect Fusion Explorer application was used to acquire the point cloud data to test the forementioned point cloud approach [16].

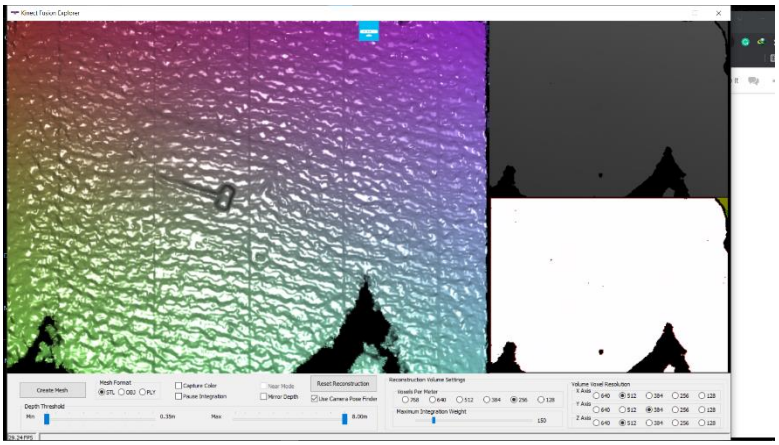


Fig. 3.17 Kinect Fusion Explorer Application in Kinect Studio (SDK)

MATLAB was also used to acquire depth images using the Image acquisition Toolbox and was also extensively used to test the point cloud approach for depth image generation. It was used to acquire the images for IR camera calibration as the corner points of the

checkerboard calibration board was comparable visible in the depth images acquired through MATLAB than the ones acquired using the python library (as in **Fig. 3.18**).

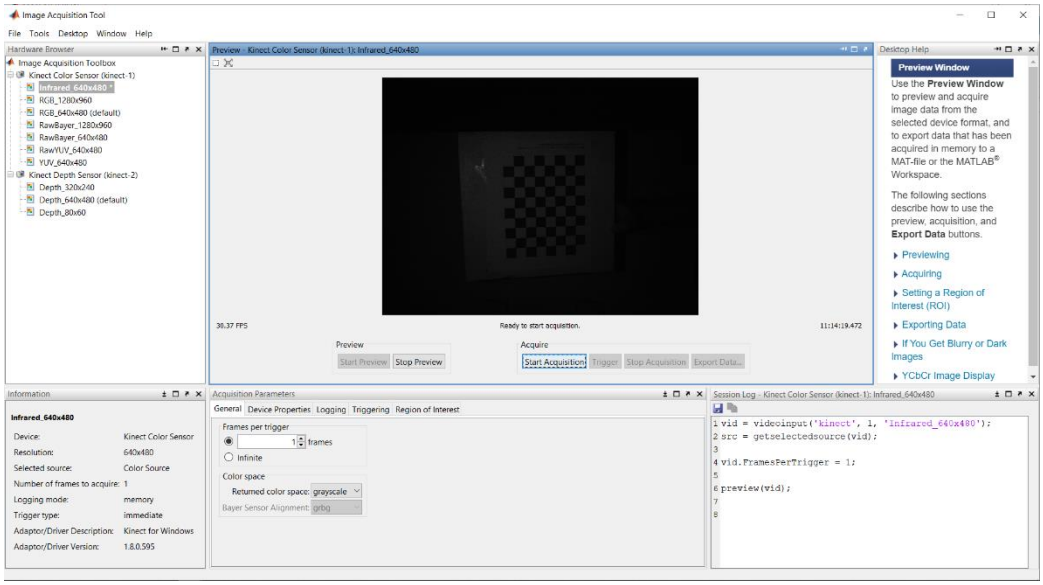


Fig. 3.18 MATLAB Image Acquisition Toolbox with Kinect v1 (Viewing IR Image preview stream)

One of the important factors to be considered when basing the entire project on python framework is that the libraries and the corresponding dependencies must have the same python version. All the python modules used in the project are in python 3.6. The python libraries used for the cameras tested for this project are:

1. For Intel Realsense D435i – pyrealsense2 is the python wrapper provided for the library librealsense v1.12.1 which was used for testing [17]. This facilitates python use of the toolkit provided in the SDK. It also works in both Ubuntu and Windows systems.
2. For Microsoft Kinect v1 – Various python libraries were tested, and many failed as it caused a mismatch or used older versions of python and anaconda. PyKinect was one of the first python libraries to be tested for the Kinect sensor, but it used older versions of python, but later PyKinect for python 3.6 was found and experimented to give minimal success in acquiring depth images. Then OpenKinect's Libfreenect for Kinect v1 was tested and obtained successful acquisition of depth images using Kinect sensor [18]. The deep learning and the computer vision subsystems are handled using an Ubuntu system, and Libfreenect is compatible with both windows and ubuntu operating systems.
3. The rest of the libraries are the usual python libraries used for computer vision solution such as numpy, cv2, matplotlib, PIL, etcetera.

3.6 Modelling and Calibration

3.6.1 Modelling

The deep learning model requires a depth image with actual depth values, so actual depth values must be calculated from the depth intensity image and that image must be sent to the network. For this objective, a black box model approach was adopted to establish a relationship between the depth intensity and the actual depth measure. For this, depth intensity image of a flat object (calibration board) was taken at measured distances from the camera in intervals of 5mm for a range of 60cm to 100cm (as in **Fig. 3.19**). Using this map, a regression model was used to find the coefficients of the polynomial that states the relationship between the depth intensity and actual depth measure values. This polynomial is then used to convert the depth intensity image into the actual depth value image needed for the network (as in **Fig. 3.20**).

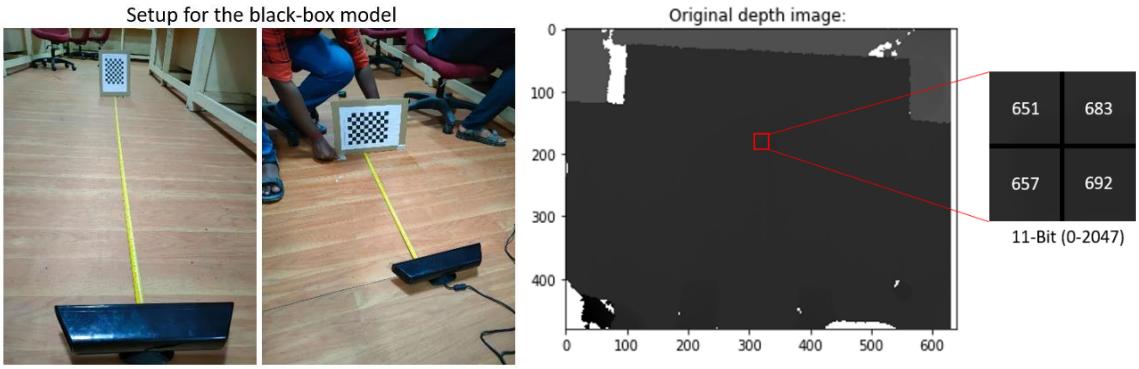
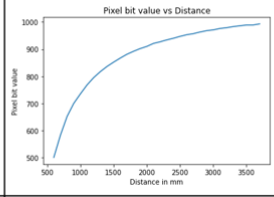
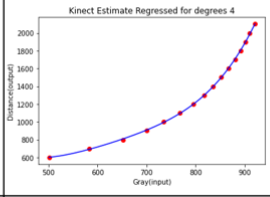
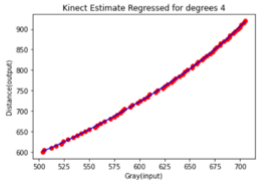


Fig. 3.19 Blackbox Model Experiment (left), Depth Intensity Image with 11-bit Values (right)

	Full range	Working range	Working range
			
Interval	100 mm	100 mm	5 mm
No. of points	31	16	83
Deg. of polynomial	9	4	4
Mean error	20 mm	5.63 mm	1.08 mm

Adopted Coefficients:

Intercept	β_1	β_2	β_3	β_4
7422.015152146081	-49.2547027	0.1295892	$-1.49627597 \times 10^{-4}$	$6.57479679 \times 10^{-8}$

Fig. 3.20 Regression on the Mapped Depth Intensity Values to Actual Depth Measures

3.6.2 Calibration

Calibration is the process done to obtain the intrinsic and extrinsic parameters of the camera used. The intrinsic parameters consist of focal length along *X* and *Y* axes of the camera, the optical axis of the camera about *X* and *Y* axes, and the optional skew coefficient. It relates the image plane coordinates to the camera coordinates. The extrinsic parameters describe the orientation and the displacement of the camera with respect to a defined world point.

This procedure is done to estimate the location of the image point in the world coordinate frame which is defined. Calibration process was conducted for Intel Realsense D435i to assess the accuracy of the depth estimation using the app available with the SDK.

An important aspect to note is that, before calibrating the camera for extrinsic parameters it must be mounted in the position and orientation deduced based on the restrictions mentioned earlier. The mount must be rigid to tolerate any disturbances to the mounted camera. This is due to the correlation of the image point to a point in the world coordinate frame which is defined using the fixed pose of the camera, and hence any disturbances could change the location of the world coordinate frame.

The calibration process involves a calibration board, which is of known dimensions and consists of patterns which can be detected using cv2 functions (python function), and the based on the position of the detected pattern points (here, corners points for the checkerboard 7 × 9 (as in **Fig. 3.21**) - 20mm dimension for each cell) it calculates the intrinsic and extrinsic parameters of the camera. When the calibration board is kept in various poses, the calibration algorithm matches the detected corner points and comparing the points to the known dimension of the calibration board, it calculates the pose of the calibration board and based on how the points are spaced between each other (by distortion) it calculates the focal length (f_x and f_y) and optical axis (c_x and c_y) in *X* and *Y* axes of the camera frame along with the skew coefficient (γ) if any (where image axes are not perpendicular). Hence, more the number of images of calibration board in various orientations and distances about the fixed camera (as in **Fig. 3.23**), the more accurate will the calculated intrinsic parameters be to the actual values of the camera. Given below are the calculated intrinsic parameters for the Kinect IR camera.

$$\text{Intrinsic parameters: } \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 580.606 & 0 & 314.758 \\ 0 & 580.885 & 252.187 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

As the Kinect v1 uses projection of structured IR (Infrared) light to estimate depth, it produces a speckle pattern in the image (as in **Fig. 3.22**). This pattern noise would cause a problem to the pattern detection (here corner detection) for the calibration board IR image. Hence the IR emitter was covered with a piece of paper to diffuse this noise. The IR images of the calibration board acquired using the python library (Libfreenect) could not be used for the calibration as the calibration pattern was not visible. So, to calibrate the IR (infrared) camera of the sensor, the Image Acquisition Toolbox for MATLAB (**Fig. 3.18**) was used as the calibration pattern is visible for it to be detected by the calibration cv2 function (cv2.cameraCalibrate()). All the code has been presented for reference [19].

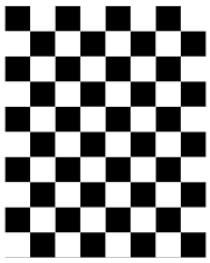


Fig. 3.21 Calibration Checkerboard 7×9 Board

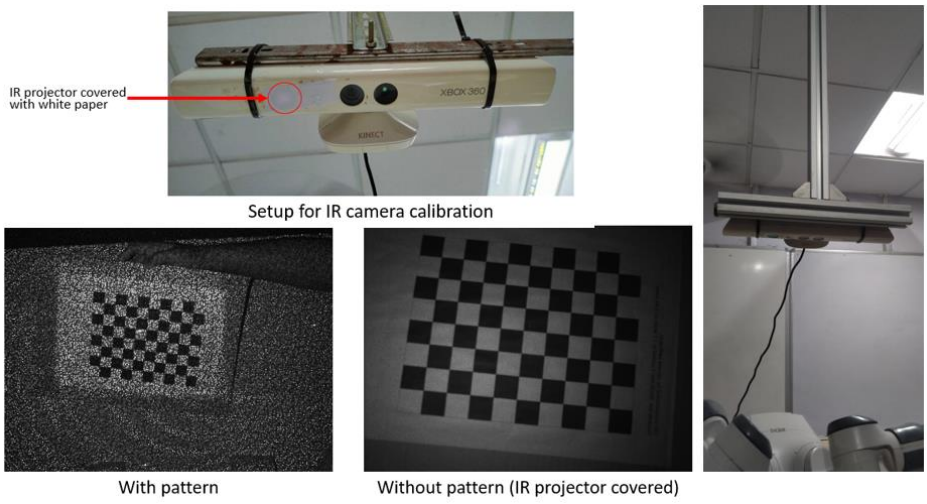


Fig. 3.22 Overhead Kinect with IR Projector Covered (top left), IR Image with Speckle Pattern Noise (bottom left), IR Image with Covered Projector (bottom mid), New Rigid Mount for Camera(right)

The extrinsic parameters were estimated by setting the first corner of the first image of the calibration board as the origin of the world coordinate frame. The position of this point was logged by moving the end effector to the point and noting down the position with respect to the robot's base coordinate frame. The pose of the camera (orientation and translation) is calculated with respect to the origin of the world coordinate frame, i.e., rotation matrix and

translation matrix are given in terms of the camera coordinate frame. Based on these matrices (transformation matrices), relationship between the image plane, camera coordinate, world coordinate and the robot coordinate frame are established.

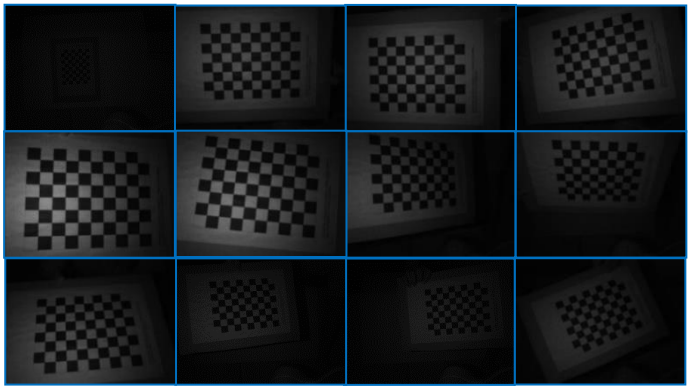


Fig. 3.23 Multiple IR Images of Calibration Board in Different Poses

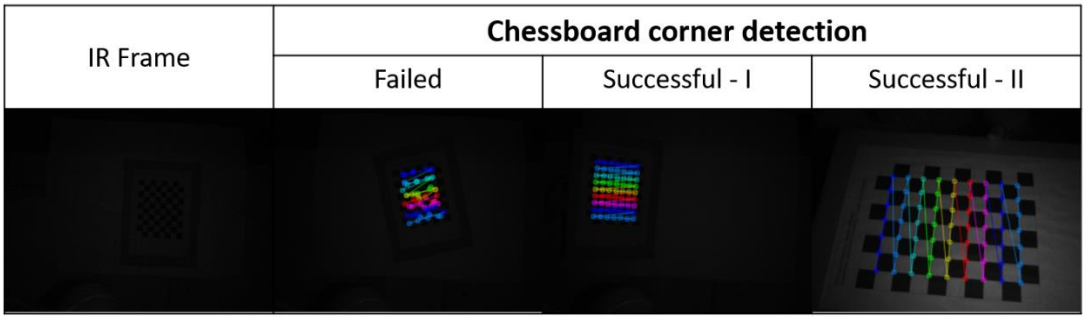


Fig. 3.24 Corner Detection Failure and Success for Intrinsic Parameters

Given below is the matrix multiplication to calculate the coordinates of a point in the world coordinate frame to the image plane. The world frame coordinate is multiplied with the extrinsic parameters (orientation and translation between the camera frame and the world frame) and then multiplied with the camera intrinsic parameters to get the coordinates in image plane from the camera coordinate frame.

$$s[I] = [C][R|t][T_w] \tag{3.2}$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{ccc:c} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{23} & t_z \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.3}$$

Here,

$s \rightarrow$ scaling factor

$I \rightarrow$ pixel coordinates

$C \rightarrow$ camera intrinsic parameters

$R|t \rightarrow$ camera extrinsic parameters

$T_W \rightarrow$ translation vector in world frame

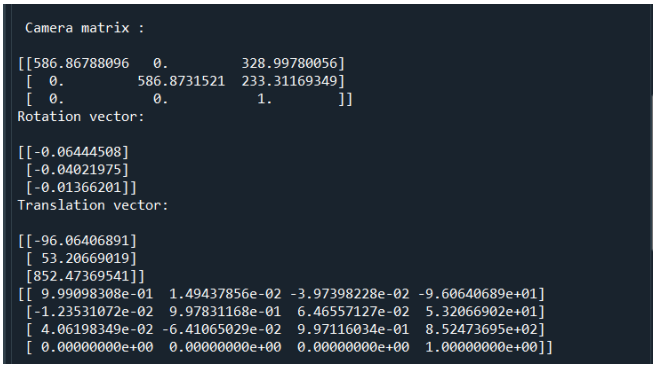


Fig. 3.25 Camera Calibration Output – Intrinsic and Extrinsic Parameters, Transformation Matrix

3.7 Image Segmentation

The deep learning model requires a segmentation mask along with the actual depth value image to localize the object of interest in the workspace. This is done using bitwise thresholding on the depth intensity image (**Fig. 3.26**). The pixel values above a certain threshold were set to 0 (black) and the values lesser than the threshold was set to 2047 (white) as the image was 11-bit image. The threshold pixel value was determined for the workspace by repeated trials such that anything above the workspace table would be segmented. The depth image is given a boundary to eliminate the noise or detection of objects outside the defined workspace (as in **Fig. 3.28**). [20]

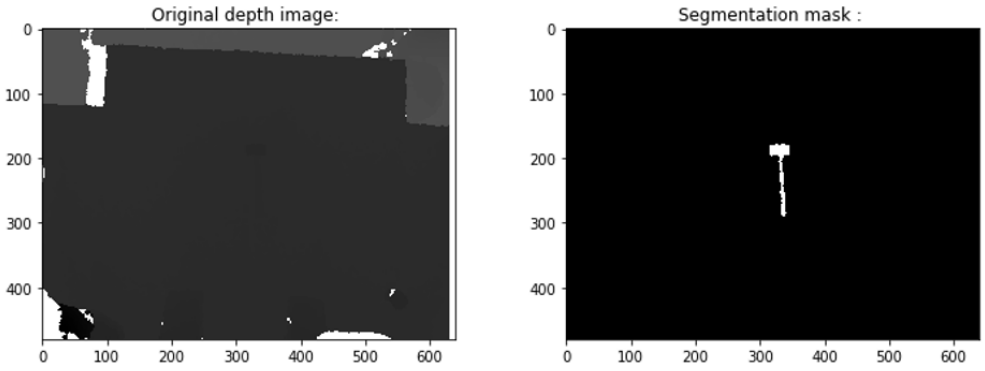


Fig. 3.26 Depth Intensity Image (left), Generated Segmentation Mask (right)

3.8 Computer Vision Approach for Grasp Localization

To compare the performance of the deep learning model, a computer vision solution to grasp localization has been implemented and tested. In the computer vision approach (as in **Fig. 3.27** and **Fig. 3.28**), the depth image is acquired, and the segmentation mask is generated as the first step, then using the segmentation mask, edges of the object are detected, and the contour of the object is found. A rectangle is fit to the contour based on minimum area possible,

and from this oriented rectangle we get the orientation of the contour on the plane with respect to the camera using the intrinsic parameters. The depth is calculated using the acquired depth image and the black box model generated polynomial relating depth intensity to actual depth measure.

The grasp is planned for the shortest dimension of the rectangle and the calculated centroid of the contour. The solution throws an error if the shortest dimension is greater than the grasp width (gripper limit – 5cm). If the grasp is suitable for execution, the quaternion for the grasp with respect to the camera is calculated and communicated to the robot control server. The output from the proposed solution is shown in **Fig. 3.29**)

The limitation to this solution is that the proposed system does not work qualitatively well for objects placed in an out of plane pose. In the solution, the object is assumed to be placed in-plane and the orientation in the plane. Qualitatively, the grasp approach must be along the normal of the top plane of the object, which would be oriented and not along camera *z*-axis if the object is kept out of plane.

The comparison on the calculated width from the acquired depth image to the actual width of the object has been conducted (as **Fig. 3.30**). The code for the computer vision solution for grasp localization is also presented. [21]

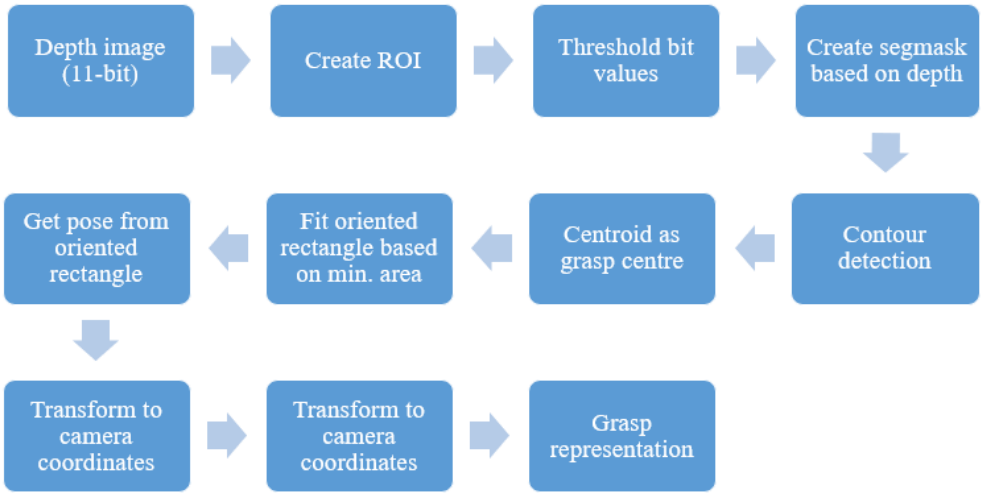


Fig. 3.27 Flowchart for the Computer Vision Solution for Grasp Localization

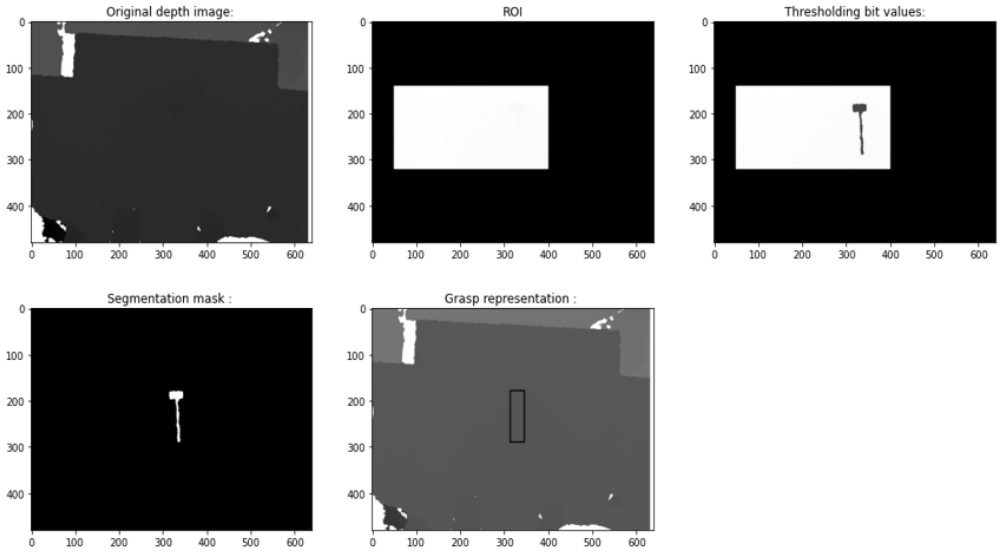


Fig. 3.28 Layers of Operations in the Proposed Solution

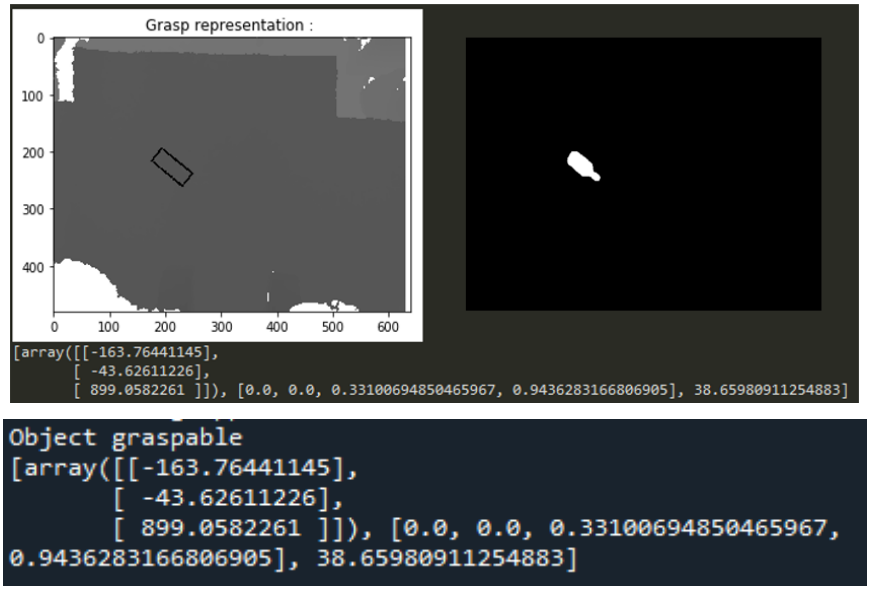


Fig. 3.29 Output from the Proposed Solution – World Coordinate, Pose in Quaternion and Angle about Camera *z*-axis (top), Output Window (bottom)

Table 3.1 Comparison between Calculated Width of Object to the Actual Width of the Object

Object	Calculated width (mm)	Actual width (mm)	Error (A-C)
Bottle	44.54	44	-0.14
Mallet	48.59	49	0.41

CHAPTER 4

DEEP LEARNING

4.1 Motivation for Deep Learning-based GL

To estimate a valid grip, traditional grasp planning based on computer vision-based algorithms includes registering images of the objects to a known database and often employs a variety of pre-processing approaches such as segmentation, classification, and geometric posture estimation [24,25,26,27]. This multistage technique is prone to numerous errors, and grasp knowledge is limited to the database, making it difficult to apply to unknown novel items. As an alternative, deep learning-based models can be used to estimate 3D object shape and location directly from colour and depth pictures, which is a more robust technique [28,29]. Recent robotics research has focused on structuring how the neural network integrates the different colour and depth streams from photographs [30], as well as adding synthetic noise to synthetic training images [31] to increase object recognition performance and also generalize better for novel objects.

4.1.1 Classification of Grasp Approaches

Vision-based robotic grasping approaches can be classified based on a variety of factors. Grasps for a physical robot are usually planned with images of target objects. There are fundamentally two ways to go about the classification based on the criteria of success:

Analytic Method [22]:

These are otherwise called geometric approach and evaluates performance based on physical knowledge i.e., based on laws of physics that rule the domain such as the ability to resist external wrenches, analyse the shape of the target object, etc.

Empirical Method [23]:

These methods are also called data-driven methods which typically approach the problem through the field of model building or in more general terms Machine Learning which has gained immense popularity in recent years.

Furthermore, depending on whether or not specific knowledge about the item is used to perform the specified task, techniques can be classified as model-based or model-free. Model-based approaches for known rigid objects typically include a pose estimation step and allow precise placement of the object. Model-free approaches directly propose to grasp candidates and typically aim for a generalization to novel objects.

4.2 Requirements of Deep Learning Networks

A detailed analysis was done to choose which deep learning network to use for the cause based on several parameters. The motivation was to prioritize two networks that worked on different types of input data (depth image and point cloud) respectively. GQCNN-4.0-PJ and Contact GraspNet 6-Dof stood out amongst the others due to the fact that it has been tested already on the ABB Yumi IRB 14000. The success rate was also relatively good for these networks with 97.8% and 88% respectively with limited use of dependencies and hardware as compared to another model like PointNet GPD. There were also a good number of citations for these networks with 655 for GQCNN and 147 for GraspNet with proper contributor's support in the github project. The other aspects taken into consideration are given in **Table 4.1** which includes the type of publication, the libraries, and the software used to implement the architecture of these networks. The GQCNN uses the TOF principle for scene reconstruction whereas the GraspNet uses Pascal 3D Plus software to do the same. Point Net GPD was not taken into consideration because of the intermediate library dependency called python-pcl which was discontinued by the contributors and is currently unable to provide functionality for python 3.x versions.

Table 4.1 Survey Details of GQCNN and Contact GraspNet

Title of the work / Specifications	**GQ-CNN**	**6 DOF-GraspNet**
Dataset used	DexNet	ShapeNet
Type of input data	Depth, Binary mask, Camera intrinsics	Point Cloud
	.npy, .png, .intr	.pcd, .stl
Type of scene reconstruction	ToF	Pascal 3D Plus, ACRONYM
Success Rate	97.80%	88%
Grasp representation	N/A	N/A
Evaluation metrics	FC	success rate and coverage rate
Dependencies	autolab-core, autolab-perception, visualization, numpy, opencv-python, scipy, matplotlib, tensorflow-gpu<=1.15.0, scikit-learn, scikit-image, gputil, psutil	pyrender==0.1.18, opencv-python, h5py, svg.path , shapely, tqdm , python-fcl , pyyaml , easydict , PySide , mayavi , rtree , scipy ,matplotlib
Robot attributes	ABB Yumi	Franka Panda
Type of publication	ICRA (International Conference on Robotics and Automation)	IEEE Conference Paper
Citation	arXiv:1703.09312 [cs.RO]	arXiv:1905.10520
No. of citations	655	147

4.3 Types of Datasets for Deep Learning

Datasets for grasp localization can be broadly classified into two categories one based on point cloud data and the other based on just image and depth data (planar). Different datasets are being made available to the community to tackle the problem of grasp localization with the highest efficiency.

4.3.1 Oriented Rectangle approach

Jacquard and Cornell's dataset contains a large diversity of objects, each with multiple labelled grasps on realistic images. Grasps are drawn as 2D rectangles on the image, darker sides indicate the position of the jaws. The grasp rectangle approach is relatively old and there are new methods implemented after this.

Fig. 4.1 Jacquard Dataset [46]

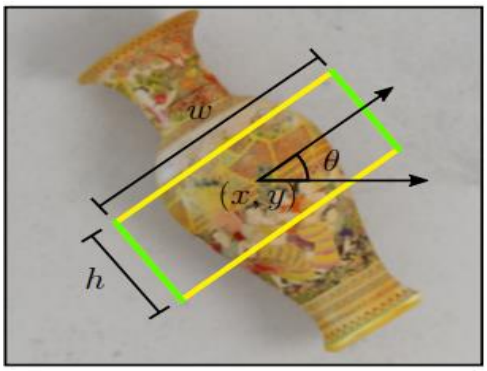


Fig. 4.2 Cornell Dataset Grasp Rectangle $g= \{x,y,h,w,\theta\}$ [47]

4.3.2 Grasp Image Dataset

The DexNet 2.0 contains 1,500 3D object mesh models where for each object hundreds of parallel-jaw grasps to cover the surface and evaluate robust analytic grasp metrics using sampling. For each stable pose of the object, we associate a set of grasps that are perpendicular to the table and collision-free for a given gripper model.

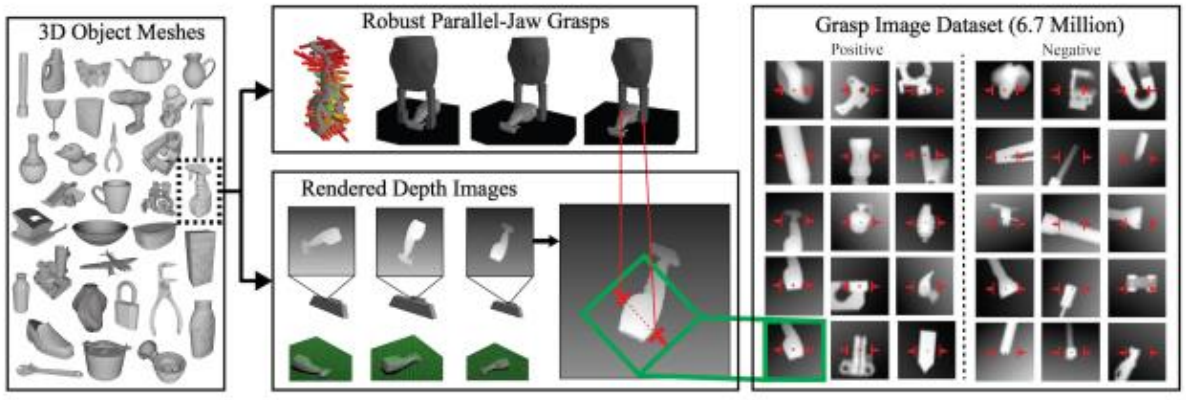


Fig. 4.3 Dex-Net 2.0 Pipeline for Training Dataset Generation [2]

Every grasp for a given stable pose is associated with a pixel location and orientation in the rendered image. Each image is rotated, translated, cropped, and scaled to align the grasp pixel location with the image centre and the grasp axis with the middle row of the image.

4.3.3 Point Cloud-Based Dataset

The ACRONYM dataset contains 20,000 parallel-jaw grasps for 8872 objects from 262 categories, totalling 17.7M grasps. The dataset uses a physics simulator to label grasps and even though being computationally more expensive, the simulation used in the dataset helps it to mimic real-world grasp performance with ease. The dataset also provides a mechanism to deal with scenarios where objects are in cluttered scenes or environments.

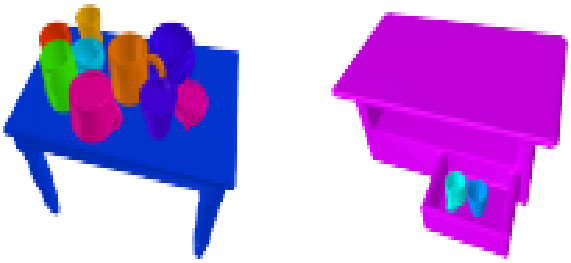


Fig. 4.4 Grasping Scenes with Structured Clutter [33]

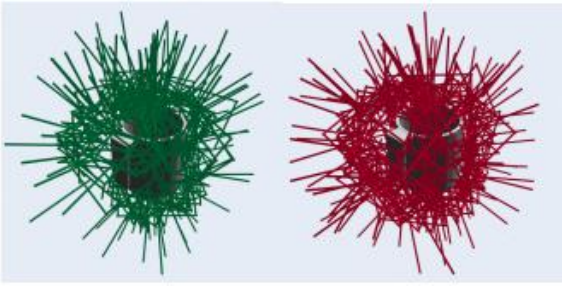


Fig. 4.5 Simulated Grasps [33]

The dataset entails a mechanism to randomly generate scenes with objects clustered on support surfaces of arbitrary meshes as shown in **Fig. 4.4.** The colours in the scene are just to show the segmentation of different objects. **Fig. 4.5** shows the status of simulated grasps where the red ones indicate failed grasps, and the green ones signify successfully simulated ones. Each object contains 2000 grasps and for easy visualization, only 15% of the grasps were taken into consideration in **Fig. 4.5.**

4.4 Hardware and Implementation Details

Both the models were run on CPU-based Ubuntu 18.04.6 bionic beaver system for inference the model is also tested out for run time boost ups with multiple GPU constructs to analyse the run time differences in the code during live inference. The CPU used is Intel Core i-5-4570 @3.2GHz x 4 with integrated intel HD graphics 4600 (HSW GT2). Speed-up is also evaluated in Nvidia GPU systems with A7650A CPU, 16 GB RAM on a A68 chipset motherboard. The GPUs used for testing were M-980 Ti, P-1070 Ti, T-2080 Ti, and A-3090. The results were noted down using the time module of python to analyse how different function domains take how much time for execution.

4.5 GQ-CNN

GQ-CNNs are neural network architectures that take as input a depth image and grasp, and output the predicted probability that the grasp will successfully hold the object while lifting, transporting, and shaking the object. In the project, the GQ-CNN takes the depth image and segmentation mask to generate the grasp candidates and output the grasp with the best score.

4.5.1 Parent Dataset

The parent dataset that is used to train the GQCNN policy is a popular research project called the "Dex Net" which was performed at the AUTO LAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab. The project took several iterations of rework and the latest iteration is termed "Dex-Net 4.0". The dataset generation was done in a hybrid approach which used both geometry and mechanics to generate synthetic training datasets. The dataset is divided into two difficulties, the former being the novel easy to grasp objects and the latter which is more challenging and intricate as compared to the first one. The Dex-Net 2.0 is a dataset associated with about 6.7-million-point clouds and analytic grasp quality metrics with parallel-jaw grasps planned using robust quasi-static grasp wrench space analysis on a dataset of 1,500 3D object models. The assumptions taken into consideration for this project are that we are using a parallel jaw gripper and a single overhead depth camera with known intrinsic parameters.

4.5.2 Architecture

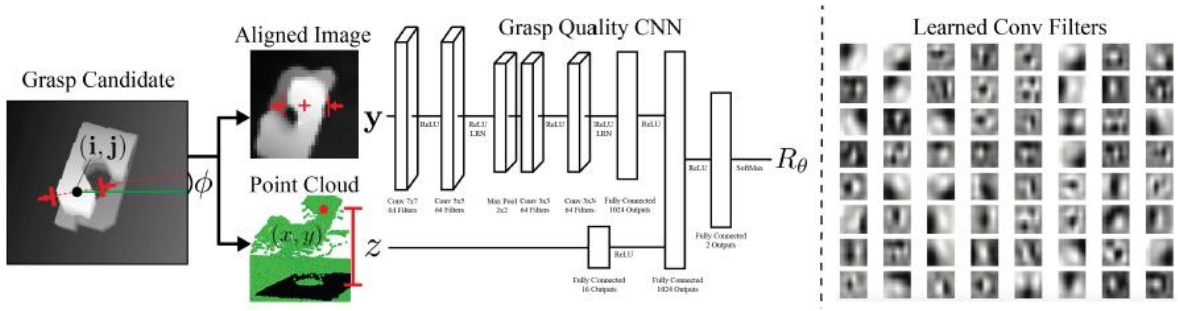


Fig 4.6 Architecture of the Grasp Quality Convolutional Neural Network (GQ-CNN) [48]

The GQ-CNN architecture, illustrated in **Fig. 4.6** and detailed in the caption, defines the set of parameters α used to represent the grasp robustness function Q. The GQ-CNN takes as input the gripper depth from the camera *Z-axis* and a depth image centered on the grasp center pixel $v = (i, j)$ and aligned with the grasp axis orientation Ω. The image-gripper alignment removes the need to learn rotational invariances that can be modelled by known,

computationally-efficient image transformations (similar to spatial transformer networks [32]) and allows the network to evaluate any grasp orientation in the image rather than a predefined discrete set. The network design comprises of four convolutional layers separated by two ReLU nonlinearities, three fully connected layers, and a separate input layer for the *Z*, the gripper's distance from the camera. The relevance of depth edges as features inspired the use of convolutional layers, and the use of ReLUs was motivated by picture classification results. There are around 18 million parameters in the network.

4.5.3 Inference and Grasp Representation

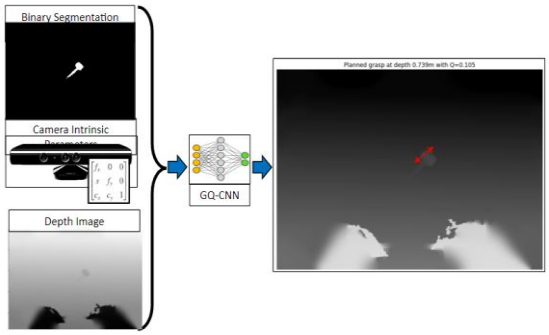


Fig. 4.7 Inputs and Inference of GQCNN-4.0-PJ Network [2]

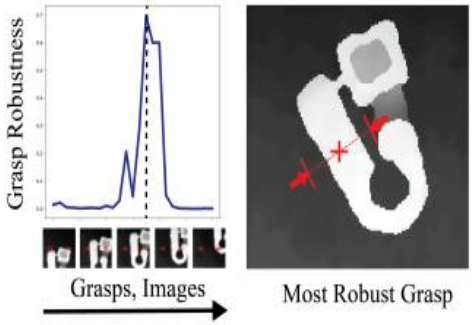


Fig. 4.8 Best Grasp *Q* [2]

The inference for the network is a transformation matrix (4×4) *Tgraspcam* which contains the orientation and translation of the grasp with respect to the camera frame. The quaternions are also provided by the network during inference to make it convenient for usage in ABB RobotStudio RAPID and trajectory execution in ABB Yumi. The GQCNN policy is trained in such a way to maximize the grasp quality measure *Q* which was simulated based on several friction and wrench mechanisms. The inference function *grasp.py* also provides the functionality to get to know about other features such as width in pixels for the grasp, approach angle, approach axis, contact normals, contact points, etc. Sample grasps for some objects are given in **Fig. 4.9.**

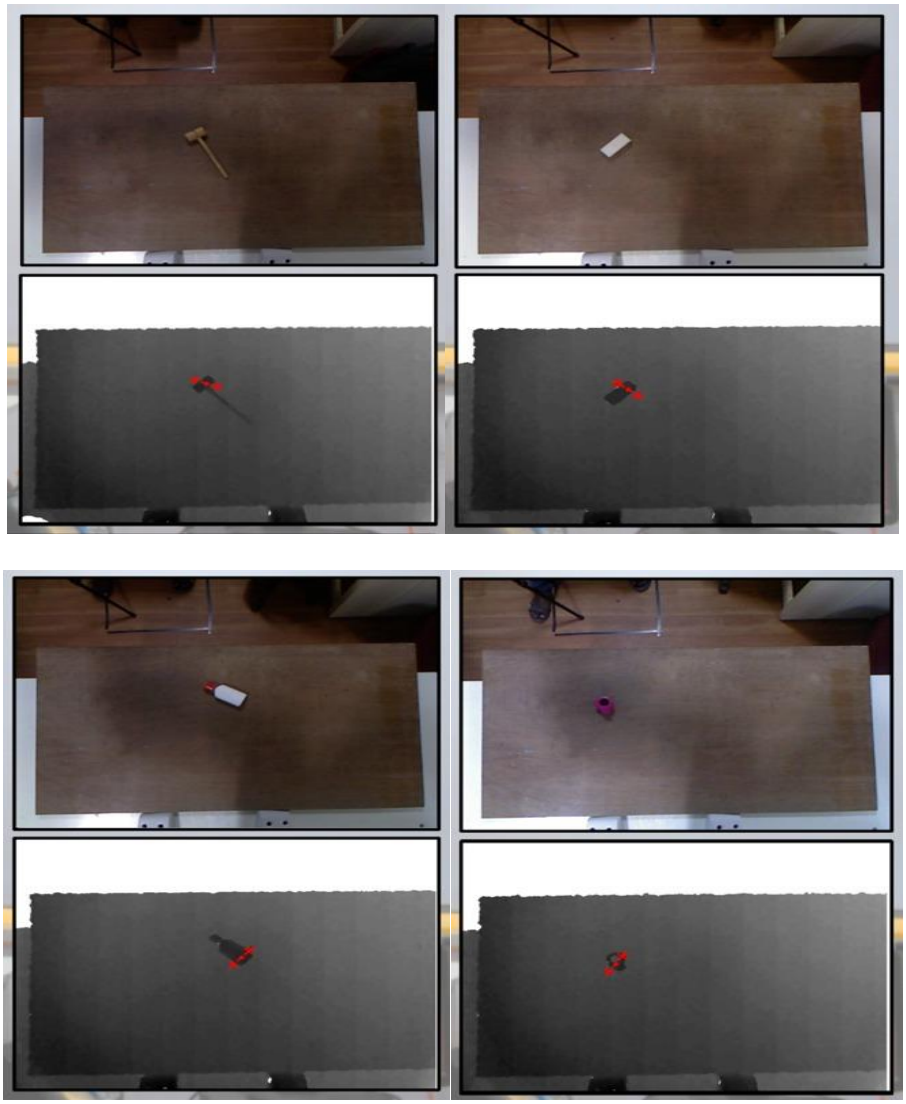


Fig. 4.9 Inferences taken for Different Objects (Mallet, Block, Bottle, Tape Holder)

4.6 Contact GraspNet 6-DoF

Contact GraspNet 6-DoF is an end-to-end network that efficiently generates a distribution of 6-DoF parallel-jaw grasps directly from a depth recording of a scene. The grasp representation treats 3D points of the recorded point cloud as potential grasp contacts. In a robotic grasping study of unseen objects in structured clutter the network achieved over 90% success rate.

4.6.1 Parent Dataset:

The parent dataset that is used to train the contact graspnet is called the "ACRONYM"[33] which is used for grasp planning based on physics simulation. The dataset provides about 17 million parallel-jaw grasps which provide a high density of grasps per object as compared to the existing datasets. The unique feature of the acronym dataset is that it produces the entire 6-dof pose instead of just giving a planar grasp where the gripper is usually

aligned with the image plane like the Cornell grasping dataset and the Jacquard dataset. One other advantage of this dataset is that it works extremely well for cluttered scenes as well, where a mechanism is provided to generate scenes with multiple objects.

The maximum grip aperture of the Franka Panda gripper is 8cm in all of the grasps in the dataset. To obtain the object meshes, the project employs ShapeNetSem [34]. Because ShapeNet is a well-known object dataset, it makes it easier to combine results from closely related studies in pose estimation, shape completeness, semantic segmentation, and other fields. Meshes with several connected components are not allowed.

Fig. 4.10 ACRONYM contains 2000 Parallel-Jaw Grasps for 8872 Objects from 262 Categories [33]

4.6.2 Architecture

To construct simulated grasps under variable friction, the architecture employs the whole distribution of stable 7-DoF grasps. To create an asymmetric U-shaped network, the network uses the set abstraction and feature propagation layers proposed in PointNet++[35]. To ensure that inference fits in the GPU memory, the network accepts roughly 20000 random points from a point cloud of dimension (20000×3) as input and predicts grasps for only 2048 points that are far away from the vantage point. There are four heads in the network, each having two 1D Conv layers and per-point outputs. To compensate for data imbalance, the anticipated grip width is divided into ten bins or intervals. The loss is calculated using the acronym dataset's ground truth data and a weighted minimum average distance between the gripper points and the ground truth data. With an initial learning rate of 0.001 and a stepwise decay to 0.0001, the network employs the adam optimizer. To avoid collisions, rejection sampling is employed.

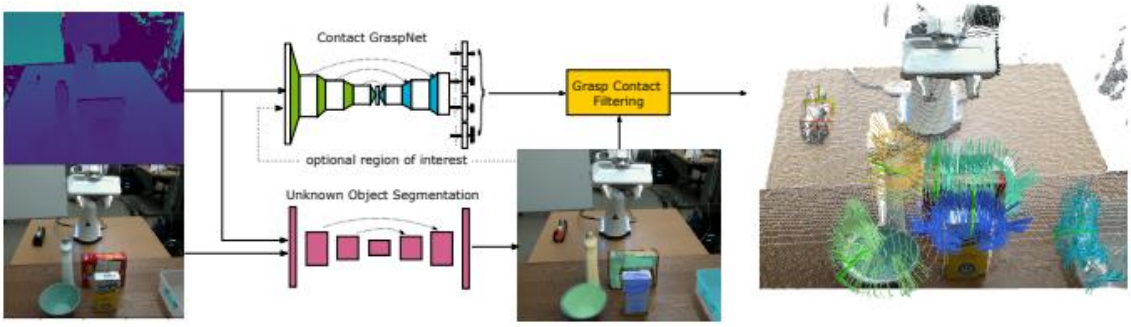


Fig. 4.11 Full Inference Pipeline of ContactNet with Contact Filtering based on the Region of Interest [3]

4.6.3 Inference and Grasp Representation

The contact grasp net saves the inference of the particular scene which is basically a transformation matrix of contact points with respect to the camera, the contact points themselves, and the corresponding quality measure of each grasp Q in three different folders as in **Fig. 4.14**. The grasps are rendered in a point cloud for visualization where each region is extracted for object segmentation and corresponding grasps are rendered **Fig. 4.13**. The grasp is represented in terms of vectors as described in the **Fig. 4.12**

$$T = \vec{c} + \frac{w}{2}\vec{b} + d\vec{a} \tag{4.1}$$

$$R = \left[\vec{b} \quad \vec{a} \times \vec{b} \quad \vec{b}\right] \tag{4.2}$$

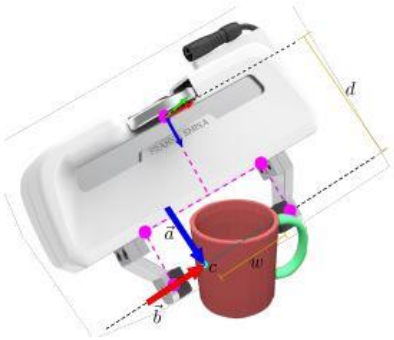


Fig. 4.12 Grasp Representation and Vector Description [3]

Live Inference **Stored Inference**

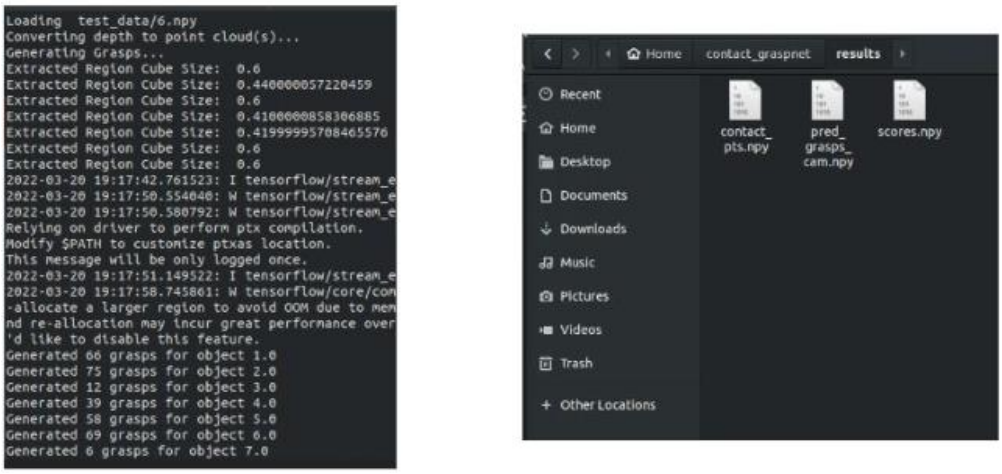


Fig. 4.13 Region Extraction and Grasp Inference Folders

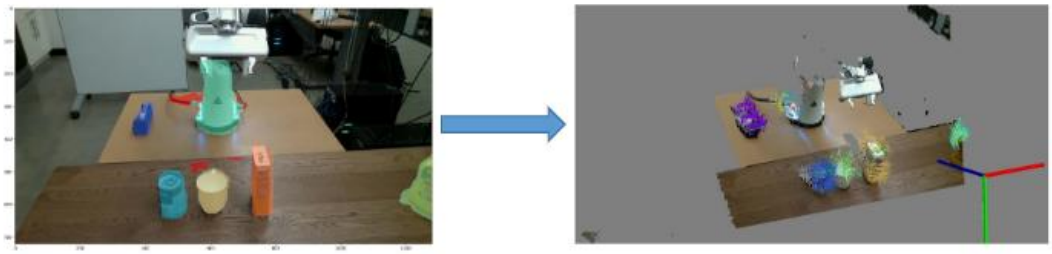


Fig. 4.14 Grasp Predictions for Cluttered Scene w.r.t Camera Frame at a Vantage Point [3] – Created Scene (left), Grasp predictions for individual objects (right)

The contact vector is defined as *c* with a being the approach vector and b being the baseline vector which is at distance *d* from the base of the gripper. $A \times B$ gives the vector which is perpendicular to both a and b which becomes a part of the rotation matrix. The network was also tested for inputs with only point cloud information of the object to produce grasp contact vectors and the inference image for a sample airplane toy is given in **Fig. 4.15.** It is recommended to test out different scenes or model parameters based on the sensor noise of the sensor which is being used as given in the grasp net repository to get the best result for pre-trained models.

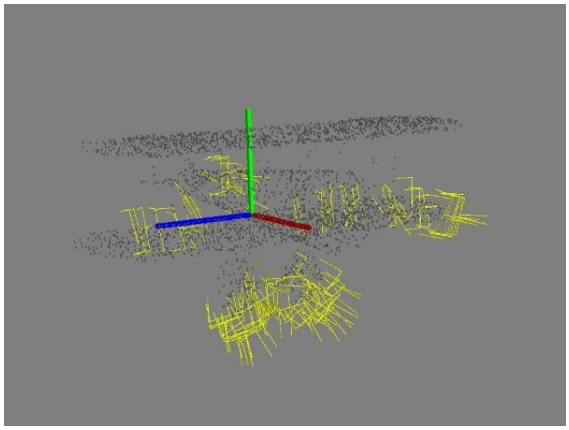


Fig. 4.15 Contact Points Representation for a Sample Airplane Toy

CHAPTER 5

ROBOT CONTROL

5.1 Pilot Studies

ABB YuMi was analysed for its compatibility with GQCNN for parallel-jaw grasp localization. Berkeley automation used the same model of IRB 14000 for successfully executing physical grasps on DexNet 1.0 through DexNet 4.0. Technical, operational and application manuals provided by ABB were thoroughly read before using the robot.

5.2 Robot Description

Apart from being ABB's first generation 7-axis dual arms robot, it is also the world's first truly collaborative dual-armed robot which is designed for humans and robots to work side-by-side with extreme accuracy while ensuring the safety of the human operator. Thanks to its flexible arms and an in-built camera-based part location system it is well suited to work with small parts, including assembly of toys, watches and automotive components.

The robot is equipped with an IRC5 controller and a robot control software, RobotWare. RobotWare supports every aspect of the robot system, such as motion control, development and execution of application programs. YuMi can handle a maximum of 0.5 kg load with a maximum reach of 0.559 meters without the smart grippers.[38]

5.2.1 Joint Nomenclature and Ranges

It is to be noted that Joint 7 is the external-axis joint that provides the robot an extra DOF and Joints 4 through 6 are the wrist joints.

Table 5.1 Joint working range [38]

Axis	Working range	Max. speed
1	-168.5° to +165.5°	180°/s
2	-143.5° to +143.5°	180°/s
3	-123.5° to +80°	180°/s
4	-290° to +290°	180°/s
5	-88° to +138°	180°/s
6	-229° to +229°	180°/s
7	-168.5° to +168.5°	180°/s

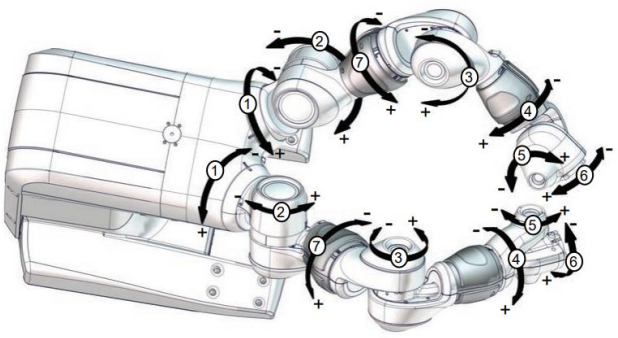


Fig. 5.1 Joint Nomenclature [38]

5.2.2 Smart Gripper

The IRB 14000 gripper is a smart, multifunctional gripper for part handling and assembly. The gripper in lab has one basic servo module and a vision module. The smart grippers communicate with the IRC5 controller over an Ethernet fieldbus. SmartGripper add-in is provided by the manufacturer to facilitate the operation and programming of the gripper. The add-in contains the RAPID driver, FlexPendant interface and configuration files. It is also to be noted that the gripper isn't arm specific and hence its chirality must be configured from the FlexPendant interface. The gripper also has a floating shell structure that helps absorb impacts during unexpected collisions.

5.2.3 Common Regulations

The robot is heavy and often extremely powerful regardless of its speed. Therefore, it is important that all safety regulations are followed while using the robot.

i. Never switch-off the robot directly. Use the FlexPendant to shut down the computer which is available under menu>restart>shutdown main computer.
ii. While testing the robot, keep the Cartesian velocities less than 100 mm/s to avoid damage to the sensitive gripper.
iii. Never forcefully move the robot arm. Either use lead-through option from FlexPendant or release brake manually.
iv. Never attempt to interrupt the robot physically while in motion unless lead-through is enabled during runtime.
v. It is recommended to simulate the robot routine in the virtual controller before executing in the physical robot.

5.2.4 Setting-up of the Robot

i. Switch ON the robot and wait for RobotWare to boot. While booting, the OS connects to the controller, after which the system and configuration files are synchronised with the controller.
ii. Once booted, the smart gripper's led will flash red lights as an indicator that the grippers are ready.

    Note: Every time you boot, the FlexPendant will display '*Event Message 71367: No communication with I/O device*'. This is normal as the controller establishes communication with the gripper after a period of 10 seconds. One must however be

cautious of *'Lost communication with I/O device'* as this either indicates a software or hardware related problem(s) with the gripper(s). For troubleshooting refer Appendix II.

iii. Always make sure to jog the grippers manually available in Menu > Smart Gripper > Left/Right Hand > Jog+/Jog - > Close. Calibrate the grippers manually if necessary. For calibration, refer Section 5.3.

iv. If the jogging doesn't pop-up any errors, continue ahead.

v. Use '*Manual*' mode to jog the robot using the joystick, else '*Auto*' to upload programs using RobotStudio. Refer **Fig. 5.2 (right)**

vi. For safety, the motors are always switched OFF after boot. Use the same side bar to switch to Manual mode and turn ON the motors.

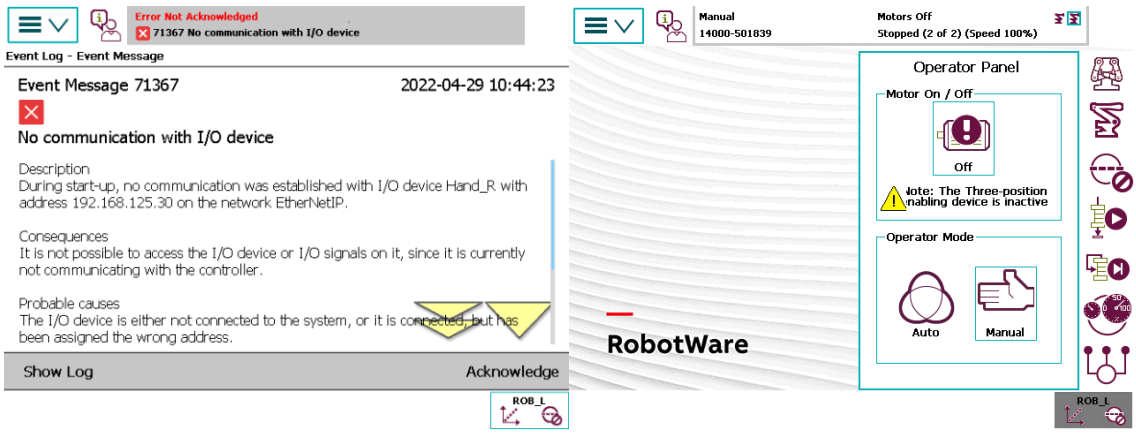


Fig. 5.2 Event Message (left), Switching Modes using FlexPendant (right)

5.3 Flex Pendant

Robotic teach pendants are wired or wireless handheld devices that come included with the industrial robot's control system. They may have several buttons or switches, or, as in newer models, a touchscreen display. They also have a display that shows the robot's commands and allows them to be edited. Furthermore, the display can be used to recall the robot's command history and even move the robot manually. Pendants make use of a keyboard for task input and simple programming for teach based methods.

ABB's teach pendant, known as the FlexPendant offers multiple functionalities. The FlexPendant is a handheld operator unit that is used for many of the tasks when operating a robot: running programs, jogging the manipulator, modifying programs, and so on. It is designed for continuous operation in harsh industrial environment. Its touchscreen is easy to clean and resistant to water, oil, and accidental welding splashes. It consists of both hardware

and software, hence it is a complete computer in itself. It is connected to the robot controller by an integrated cable and connector.

Fig. 5.3 FlexPendant Description

Table 5.2 FlexPendant Description

Marker	Description
A	Connector
B	Touch screen
C	Emergency stop
D	Joystick
E	USB Port
F	Three-positioning enabling device
G	Stylus pen
H	Reset button

There are dedicated hardware buttons on the FlexPendant. Buttons A-D are programmable buttons to which their respective functions can be assigned.

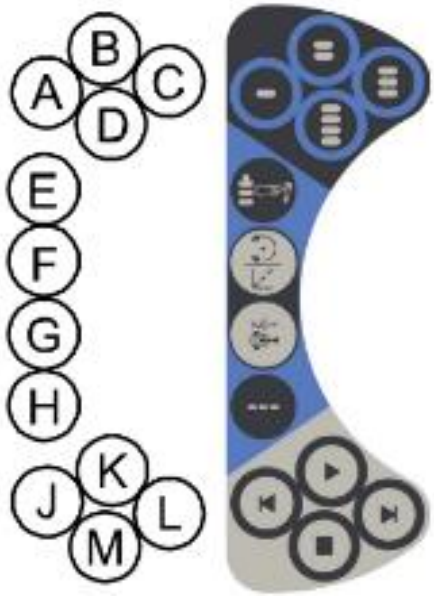


Fig. 5.4 FlexPendant Buttons

Table 5.3 FlexPendant Buttons Description

Button	Function
A-D	Programmable keys
E	Select mechanical unit
F	Toggle motion mode, linear/ reorient
G	Toggle motion mode, Axis 1-2 / Axis 4-6
H	Toggle motion increments
I	Three-positioning enabling device
J	Step back in RAPID code
K	Run routine
L	Step forward in RAPID code
M	Stop robot execution

5.3.1 Commonly Used Menus

Illustration in **Fig. 5.5** shows the important elements of the FlexPendant touch screen.

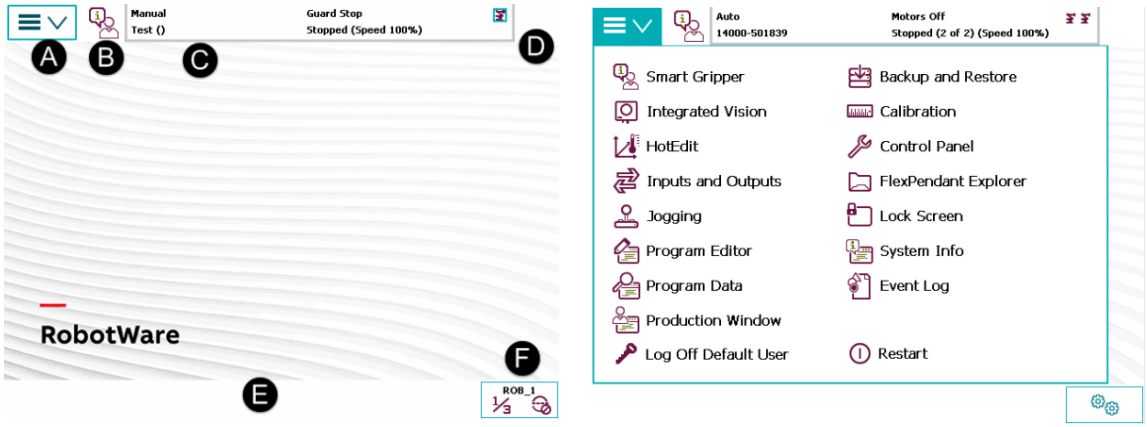


Fig. 5.5 FlexPendant Home Screen Elements (left), Main Menu Items (right)

Table 5.4 FlexPendant Menu Description

Marker	Name	Description
A	Main menu	Offers multiple views
B	Operator menu	The operator window displays messages from robot programs. This usually happens when the program needs some kind of operator response in order to continue.
C	Status bar	The status bar displays important information about system status, such as operating mode, motors on/off, program state and so on.
D	Close button	Tapping the close button closes the presently active view or application
E	Task bar	The task bar displays all open views and is used to switch between these
F	Quick set menu	The quickset menu provides settings for jogging and program execution

From the main menu, the following items can be selected:

- Smart gripper
- Integrated vision
- HotEdit
- Inputs and Outputs
- Jogging
- Program Editor
- Program Data
- Production Window
- Log Off Default User
- Backup and Restore
- Calibration
- Control Panel
- FlexPendant Explorer
- Lock Screen
- System Info
- Event log
- Restart

- Jogging menu: Jogging is to manually position or move robot using the joystick.

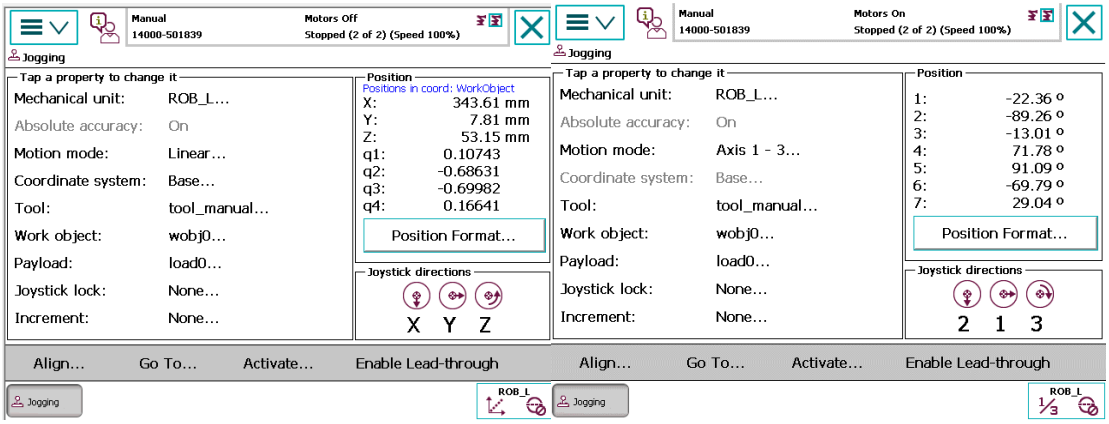


Fig. 5.6 Jogging Menu – Linear (left), Jogging Menu – Axis (right)

Procedure to jog the robot:

i) Make sure to keep the operation mode as '*Manual*'

ii) Go to Main Menu > Jogging. Here select the arm to be jogged under Mechanical unit i.e. ROB_L for left arm and ROB_R for right arm.

iii) In motion mode select one of the following:

- Linear: Cartesian straight linear motion of the TCP

- Axis: Individual joint space jogging

iv) Select the tool (tool_manual in our case) which defines the TCP w.r.t the flange

v) Select coordinate from to be base (i.e. base of the robot)

vi) Use the joystick to jog the axes

5.4 RAPID Programming

RAPID is a high-level programming language used to control ABB industrial robots.

5.4.1 Basic Program Structure

Every RAPID program consists of the following in order

```
MODULE <Module Name>
-- Variables, Constants, Tool object definitions --
PROC main()
-- Instructions --
END PROC
PROC <Procedure Name>
-- Instructions --
END PROC
FUNC <Function type> <Function name> <Function arguments>
-- Function definition --
ENDFUNC
ENDMODULE
```

Note: Every program (ROB_L / ROB_R) should have only one main procedure.

The workflow is sequentially listed below:

i) Create targets and path

ii) Specify work object

iii) Check reachability

iv) Command motion using move commands

v) Collision detection

vi) Edit code on RAPID editor in RobotStudio

vii) Test the program

5.4.2 Setting Up in RobotStudio

Before one starts to use RobotStudio follow these steps:

i) Go to File > Options > Licensing > Activation Wizard

- Enter the IP address of the RobotStudio server and click Finish (refer **Fig. 5.7**).

ii) Enable '*Auto*' operation mode from the FlexPendant.

iii) Navigate to Controller tab > Add controller > One Click Connect / Select system name (Here it is Test_new). Navigate

iv) In RAPID tab, select Request Write Access available under Actions section.

- If the Operation mode is *Auto*, write access is given automatically, else grant permission has to be clicked on the FlexPendant's pop-up menu.

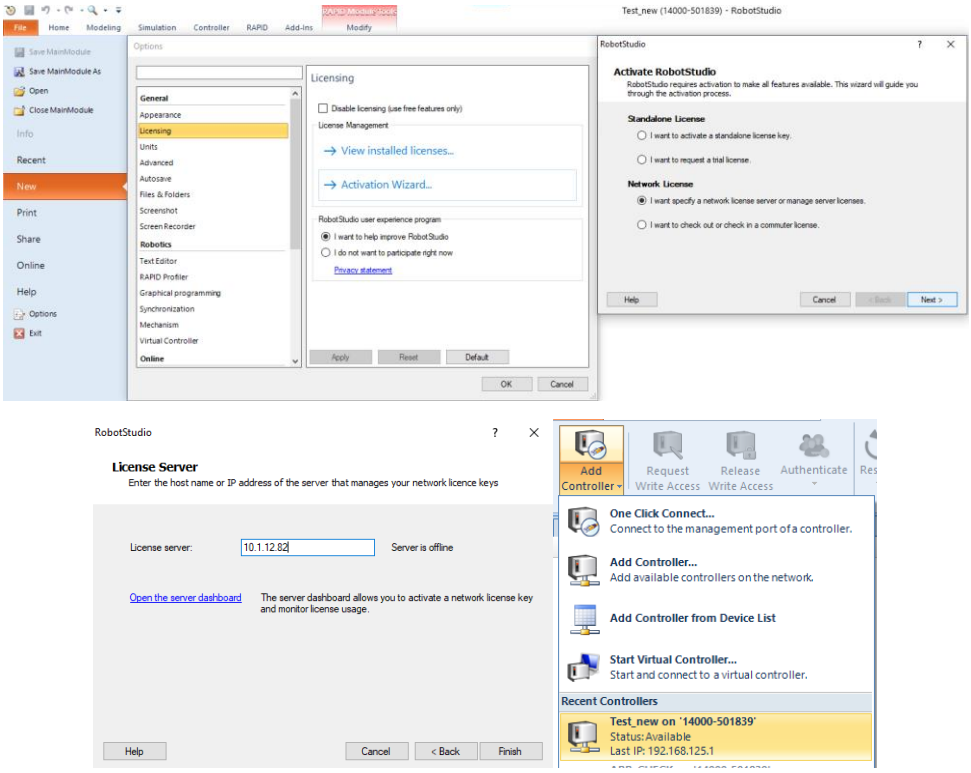


Fig. 5.7 License Screen (top and bottom left), Adding Controller (bottom right)

5.5 Firmware and Add-in Software

The robot currently runs on RobotWare v6.13, which is the operating system of the FlexPendant. Compatible RobotWare add-ins namely Smart Gripper is also installed while creating the system from scratch. For additional functionality, the robot can be equipped with optional software for application support, for instance communication features, network communication, multitasking, sensor control, etc. Appendix II-Robot Control Section-III may be referred for firmware related details.

5.6 Kinematic Understanding of the Robot

5.6.1 Manipulability

Besides the benefit of an increased workspace due to two arms, it is important to calculate the limits of the workspace in terms of reachability and problem regions within workspace using manipulability. Workspace analysis starts by calculating the forward kinematics i.e., the cartesian coordinates of the end-effector with respect to the base of the robot and calculating the Jacobian for each joint configuration. Once each pose is visualized as a point cloud for each arm, the arm span and workspace volume are calculated.

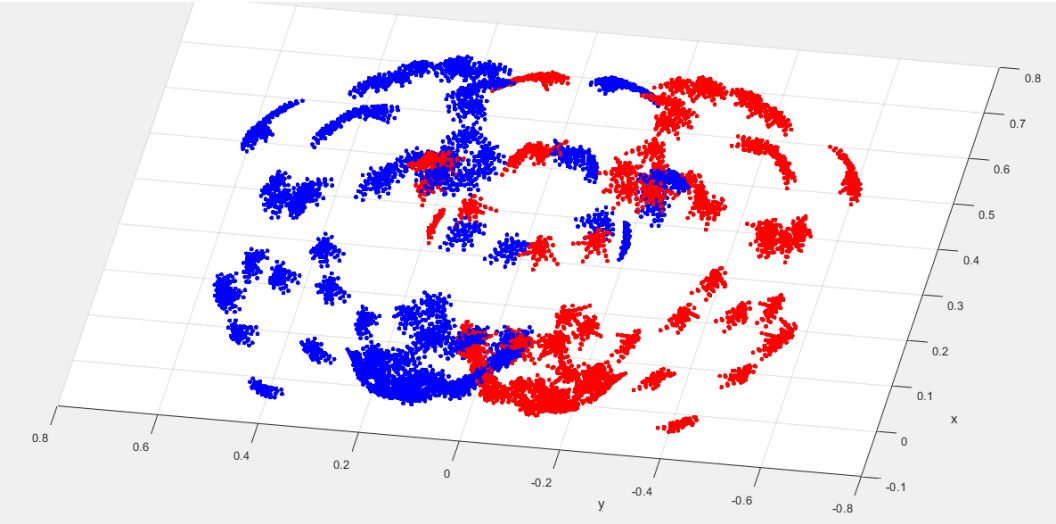


Fig. 5.8 *X-Y* Plane Projection of the Workspace of ABB YuMi

Another important aspect of workspace analysis is finding the areas of good and reduced manipulability by using Yoshikawa manipulability measure that describes how close the robot is to a singular condition. The manipulability index is calculated as:

$$m = \sqrt[2]{\det(JJ^T)} \qquad ;0 \le m \le 1 \qquad (5.1)$$

where, J is the Jacobian matrix of the robot arm at a particular joint configuration. This measure is proportional to the velocity ellipsoid at a configuration what describes the ability of

the robot's end-effector to move in each of the spatial degree of freedoms. The bigger the index, more the manipulability the robot has in its current position. The cases for *m* are as follows:

- m = 1: Isotropic
  - Velocity ellipsoid is spherical
- m = 0: Singular
  - Velocity ellipsoid is flat i.e., loses 1 DOF

The manipulability index is calculated for every configuration and a cut-off index is defined, below which a robot's configuration is considered poorly manipulable. Percentage of such configurations are also calculated.

Table 5.5 Manipulability Analysis

Number of configurations	Maximum manipulability index	Minimum manipulability index	Poorly manipulable configuration percentage
19683	0.1171	0.0016	20.38 %

Steps for Workspace and Manipulability analysis on MATLAB:

1) Load robot 'abbYuMi' using Robotics Toolbox with 'homeConfiguration'
2) Set joint limits of each joint (Refer robot object in workspace for body and joint names)
3) Iterate for every possible joint configuration and plot the points
4) Calculate the transformation matrix between base and TCP point using getTransform()
5) Calculate Jacobian matrix using geometricJacobian()
6) Calculate Manipulability index using $m = \sqrt[2]{\det(JJ^T)}$
7) Set cut-off index and store configurations for this condition

Note:

- Base frame in MATLAB is the same as the physical robot
- Joint configurations of robot and in MATLAB match

Table 5.6 Physical Robot vs MATLAB Virtual Robot

	LEFT			**RIGHT**		
	Flex Pendent	**MATLAB**	**Error (M-FP)**	**Flex Pendent**	**MATLAB**	**Error (M-FP)**
X	566.83	568.6	1.77	567.37	568.5	1.13
Y	-7.07	-5.0	2.07	5.14	5.9	0.76
Z	414.11	413.6	-0.51	412.88	414.6	1.72
Q1	0.29029	0.29748		0.29245	0.29766	
Q2	-0.06496	-0.06775		0.06356	0.06718	
Q3	0.87641	0.87515		0.87689	0.87437	
Q4	-0.37871	-0.37552		0.37615	0.37728	

5.6.2 Robot Joint Logging

In an attempt to find the optimum robot confguration, the joint angles for a set of move commands was recorded and plotted against time in seconds. For the experiment, a 3D cuboidal region was chosen and MoveAbsJ commands were used for joint-space motion for all the combinations of the paths formed by the corners of the cuboid.

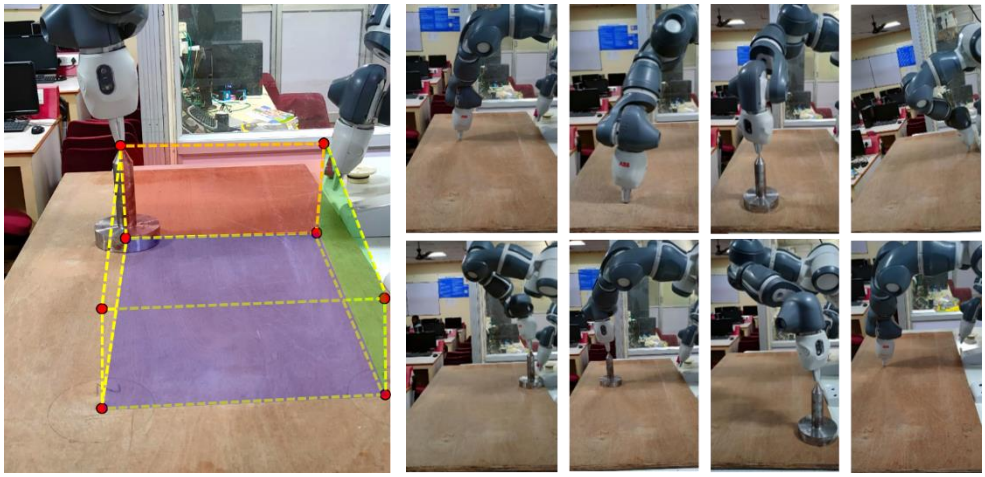


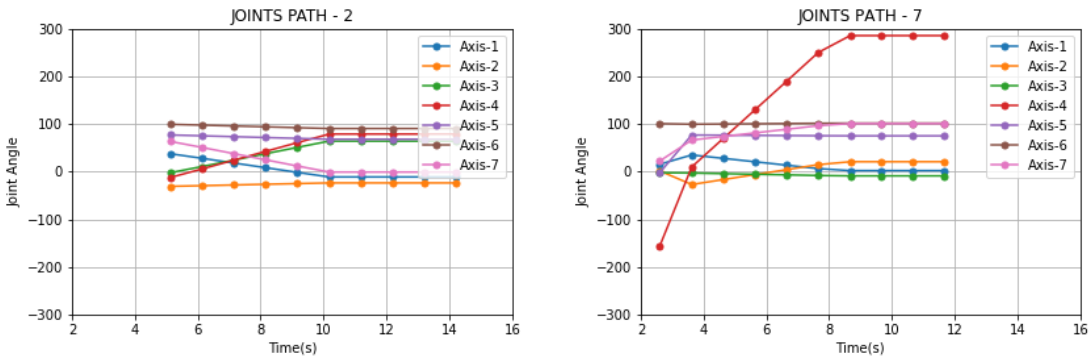
Fig. 5.9 Cuboidal Region Chosen for the Experiment (left), Experiment (right)

The joints were recorded in RAPID and were sent to the python-server via socket. The data is stored as .npy format with the Path name as the name of the numpy file. A separate code is used to visualize the recorded data. The figure below shows a minute sample of the recorded joints.

It is to be noted from the figures that Axis-4 achieves several quadrants; the maximum being 6. From the experiment, the take-away points are as follows:

i) Robot configuration isn't constant in a move command and hence changes in a move command

ii) Since axis-4 achieves various robot configurations, the optimum robot configuration should minimise the variance of *cf4*.



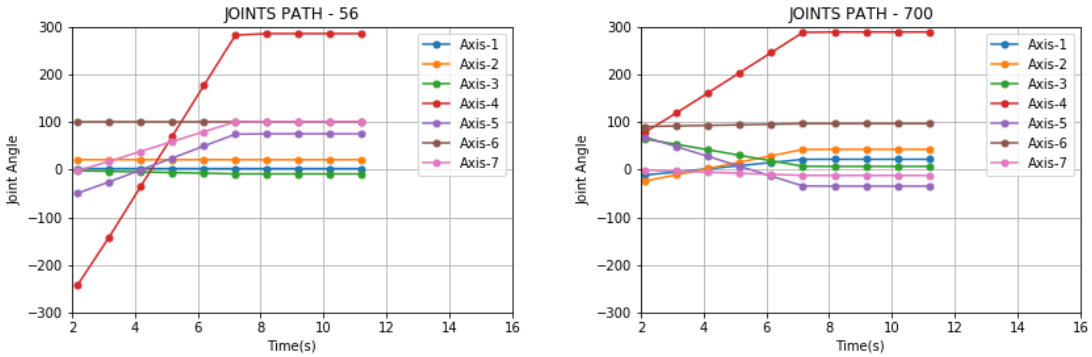
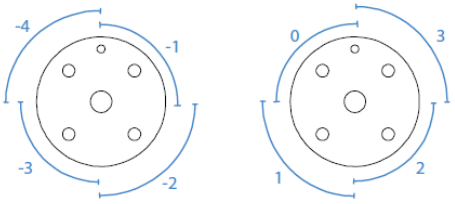


Fig. 5.10 Sample of Logged Joints for Commanded Move Commands

5.6.3 Robot Configuration

For a given pose of the end-effector, the robot can achieve it in multiple joint configurations. In other words, there are multiple inverse kinematic solutions for a given pose. Hence, in order to get a single solution, certain constraints are provided while solving the inverse kinematics problem. In RAPID, this constraint is expressed as a robot configuration data that consists of *cf1*, *cf4*, *cf6* and *cfx* that denote the joint position of axes 1,4 and 6. The parameter *cfx* however denotes the placement of the wrist center w.r.t the lower arm and the sign of *Axis-2* and *Axis-5*. The figure below describes the robot configuration in detail.



cfx	Axis 2 angle	Wrist center relative to lower arm	Axis 5 angle
0	Positive	In front of	Positive
1	Positive	In front of	Negative
2	Positive	Behind	Positive
3	Positive	Behind	Negative
4	Negative	In front of	Positive
5	Negative	In front of	Negative
6	Negative	Behind	Positive
7	Negative	Behind	Negative

Fig. 5.11 Robot Configuration Quadrants (left), Look-up Table for cfx (right)

Robot configuration in RAPID is expressed in confdata [*cf1*, *cf4*, *cf6*, *cfx*]. Specifying robot configuration is a must in cartesian space planning. For the purpose of finding an optimum robot configuration, an experiment was conducted wherein a grid region of 6×10 was created on the table, each grid being 50mm × 50 mm. The robot is then commanded to move to each and every region while its joints and calculated robot configuration (Using CalcRobT) are simultaneously recorded and stored.

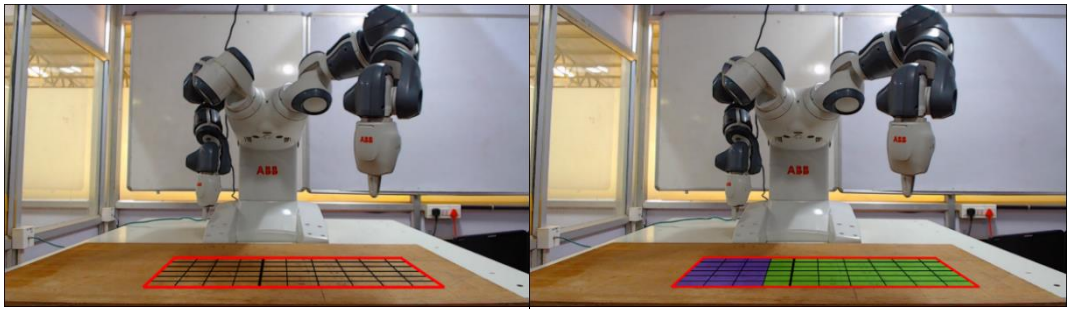


Fig. 5.12 Robot Configuration Grid

In attempt 1, (refer to **Fig. 5.13**) it is to be noted that the robot achieves various configurations and hence a common configuration doesn't exist. However, in attempt-2 for a different robot arm joint configuration, a region of 6 × 7 grid constitutes a common robot configuration while the other 6 × 3 grid constitutes another robot configuration. From this experiment, the chosen robot configuration is [-1, 1, γ, 4]

Here, *cf6* is γ as its configuration depends on the grasp orientation and hence is subject to change.

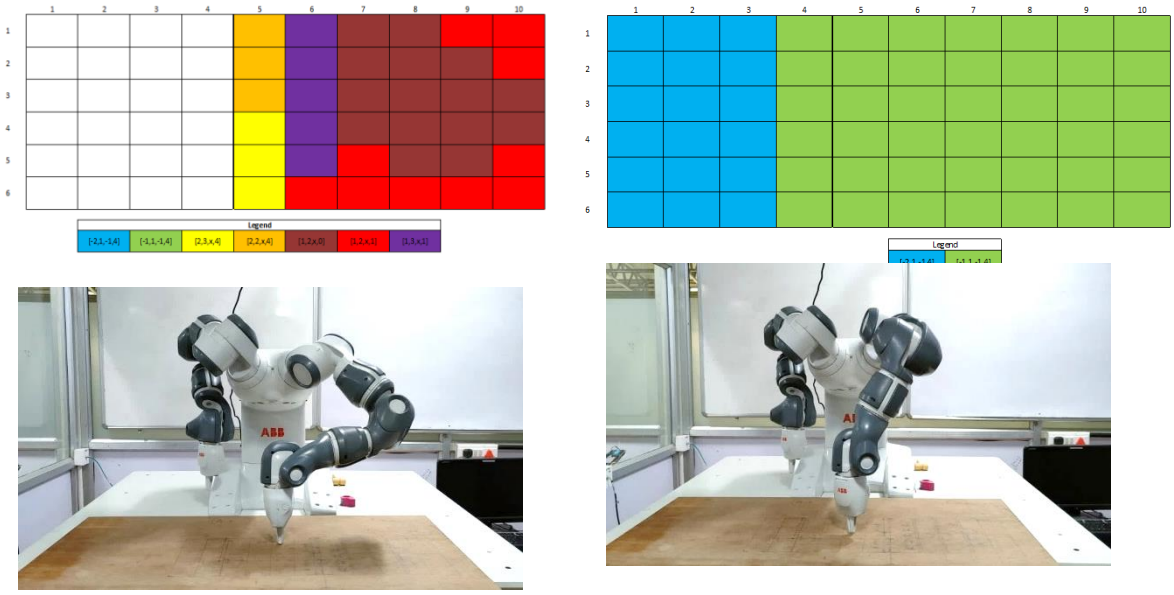


Fig. 5.13 Robot Configuration Experiment: Calculated Robot Configuration (top left and top right) for Robot Arm Joint Configuration (bottom left and bottom right)

5.7 Calibration of the Robot

Over time, the rotary encoders of the robot may go off the correct value due to multiple reasons. Hence, it is necessary to re-calibrate the robot arm when discrepancies are noticed.

5.7.1 Robot Arm Calibration

The calibration window is used to perform a rough calibration of each robot axis i.e., updating the revolution counter value for each axis, using the FlexPendant. Following is the procedure:

i) Go to Main Menu > Calibration > ROB_R/ROB_L > Manual Method (Advanced)

ii) Position the selected robot arm in the calibration position as illustrated in **Fig. 5.14** by aligning the markers provided for each joint.

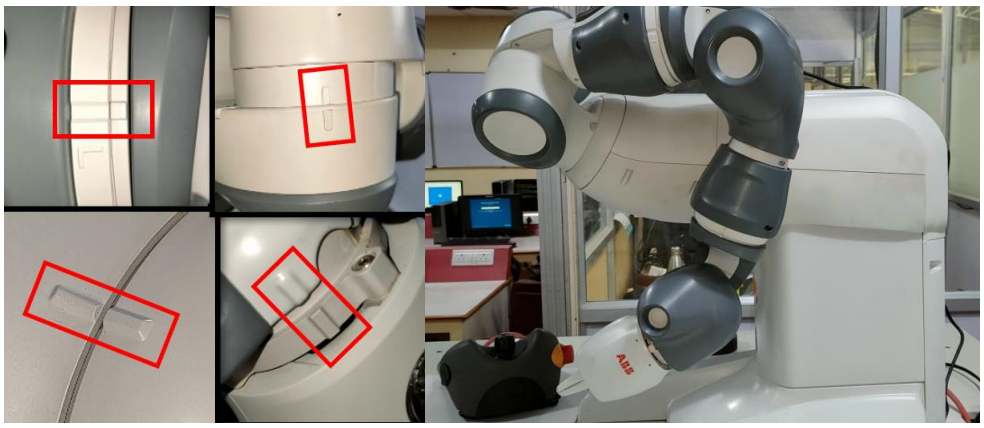


Fig. 5.14 Robot Arm Calibration Markers (left), Robot Calibration Pose (right)

iii) Click on Update Revolution counters > Yes > Select all axis > Update

Note: If a revolution counter is incorrectly updated, it will cause incorrect manipulator positioning, which in turn may cause damage or injury!

5.7.2 Gripper Finger Calibration

The smart gripper menu allows one to configure and jog the grippers manually using the FlexPendant. The chirality of the gripper can be changed using the configuration option. Jogging and calibration can be performed by opening the respective Hand option (Right/Left). Jog the gripper to its minimum by using Jog – and then click on *Calibrate*.

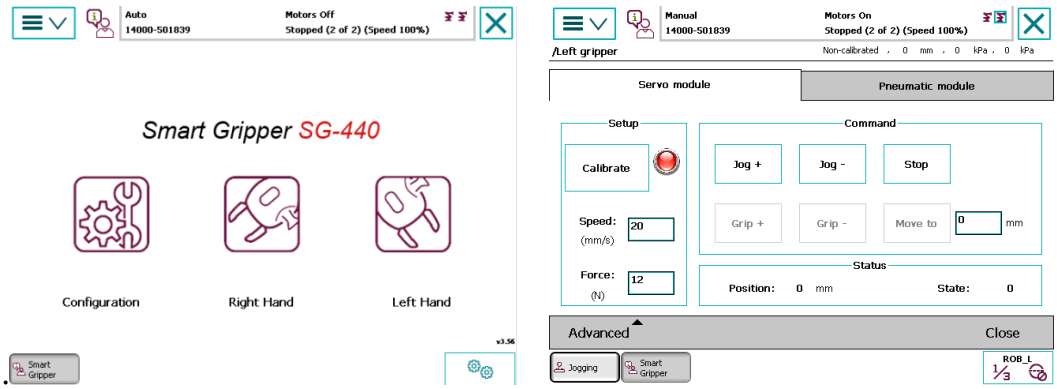


Fig. 5.15 Smart Gripper Menu (left), Gripper Jogging and Calibration (right)

5.7.3 TCP calibration

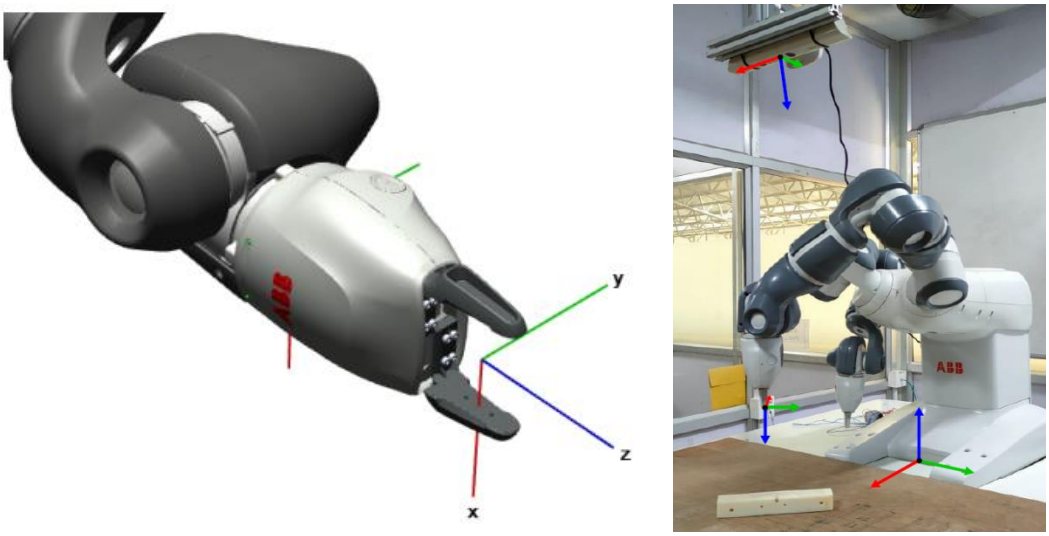


Fig. 5.16 TCP Frame Assignment (left), All Frame Assignments (right)

Although 9-point TCP calibration was performed using the FlexPendant, it was later found to be in vain as the gripper's tooldata was readily available in the Product manual of IRB 14000 smart gripper. The following is the tooldata definition for the same.

TASK PERS tooldata tool_manual :=[TRUE ,[[0,0,136],[1,0,0,0]],[0.244,[7.5,11.8,52.7] ,[1,0,0,0],0.00021,0.00023,8E-05]];

5.8 RAPID Routines

The RAPID routine is structured in such a manner that the robot's controller will run a stand-alone RAPID code which consists of the move commands which is connected to a python-based server via socket communication. The grasp pose is stored in a PERS (persistent variable), whose value gets updated every iteration.

Since the robot is the client, it requests for a connection to an already running server; in our case the python-based server. Once the connection is established, the robot sends a confirmation stating it is ready to receive grasp pose. The robot then moves to a previously specified home position if the arm is in a different pose. Upon receiving the grasp pose as string, a user-defined function is called to convert the string to robot target. This robot target is stored in the previously specified persistent variable. From this robot target, the grasp axis is calculated and the robot assumes a specific distance along the grasp axis based on the size of the target object. After aligning the TCP along the grasp axis the gripper opens and MoveL command is given for a linear motion along this axis with a low speed of 10 mm/s. Upon reaching the grasp pose, the gripper is closed and the object is picked and dropped at a specific drop point. Once dropped, the robot re-orients itself to the home position.

The home position is configured in such a way that it doesn't occlude the object in the camera's view and takes manipulability of the robot into consideration while moving towards the grasp axis pose.

5.9 Homogenous Transformations

It is known from the previous chapters that both deep learning and computer vision inference give us the pose of the grasp w.r.t the camera frame. It is important to note the frame assignment of the camera frame as it differs from module to module. It is known that extrinsic camera calibration in OpenCV provides the pose of the world w.r.t the camera. Since the world frame is placed at a convenient location, the world frame's origin w.r.t the robot's base frame is found by jogging the robot's end-effector to the origin. Here, only the translation vector was noted from the FlexPendant. The orientation was set as per our needs as described in the previous chapters. Refer to **Fig. 5.16** for frame assignments.

For the robot to pick the object, it needs to know the grasp pose in terms of the robot's base frame. With the known transformations, the pose of the grasp w.r.t the robot is mathematically calculated as described in the following equations:

- Transformation of camera w.r.t robot base:

$$ {}^{R}_{C}T = {}^{R}_{W}T \times {}^{W}_{C}T \tag{5.2} $$

$$ {}^{R}_{C}T = \left[\begin{array}{cccc} 0 & -1 & 0 & 516.88 \\ -1 & 0 & 0 & 62.24 \\ 0 & 0 & -1 & 254.11 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \times \left[\begin{array}{cccc} 0.9985 & -0.0426 & 0.0328 & 72.91 \\ 0.0443 & 0.9976 & -0.0518 & 51.63 \\ -0.0306 & 0.0531 & 0.9981 & -622.59 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{5.3} $$

$$
{}^{R}_{C}T = \left[\begin{array}{ccc:c} -0.0443 & -0.9976 & 0.0518 & 465.24 \\ -0.9985 & 0.0426 & -0.0328 & -10.67 \\ 0.0306 & -0.0531 & -0.9981 & 876.70 \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right] \tag{5.4}
$$

${}^{W}_{C}T \rightarrow$ Obtained from camera extrinsic calibration

- Transformation of grasp w.r.t robot base:

$$
{}^{R}_{G}T = {}^{R}_{C}T \times {}^{C}_{G}T \tag{5.5}
$$

The transformation obtained from Eqn. (5.5) is then converted to the robot's compatible format i.e., in quaternion and translation vector.

CHAPTER 6

INTEGRATION AND PERFORMANCE ANALYSIS

6.1 Software Integration and Hardware

The communication interface has basically three systems the python server, the deep learning client, and the Robot Studio client. In the first sequence, the deep learning system interacts with the Python server to establish socket communication. Then a command is sent via the python server to the deep learning client stating a message to start the live inference process. The deep learning client takes the depth image using libfreenect and automatically generates a segmentation mask which is sent to the network to get the grasp inference in the form of translations and quaternion. The grasp information is encoded into a string and is sent to the python server.

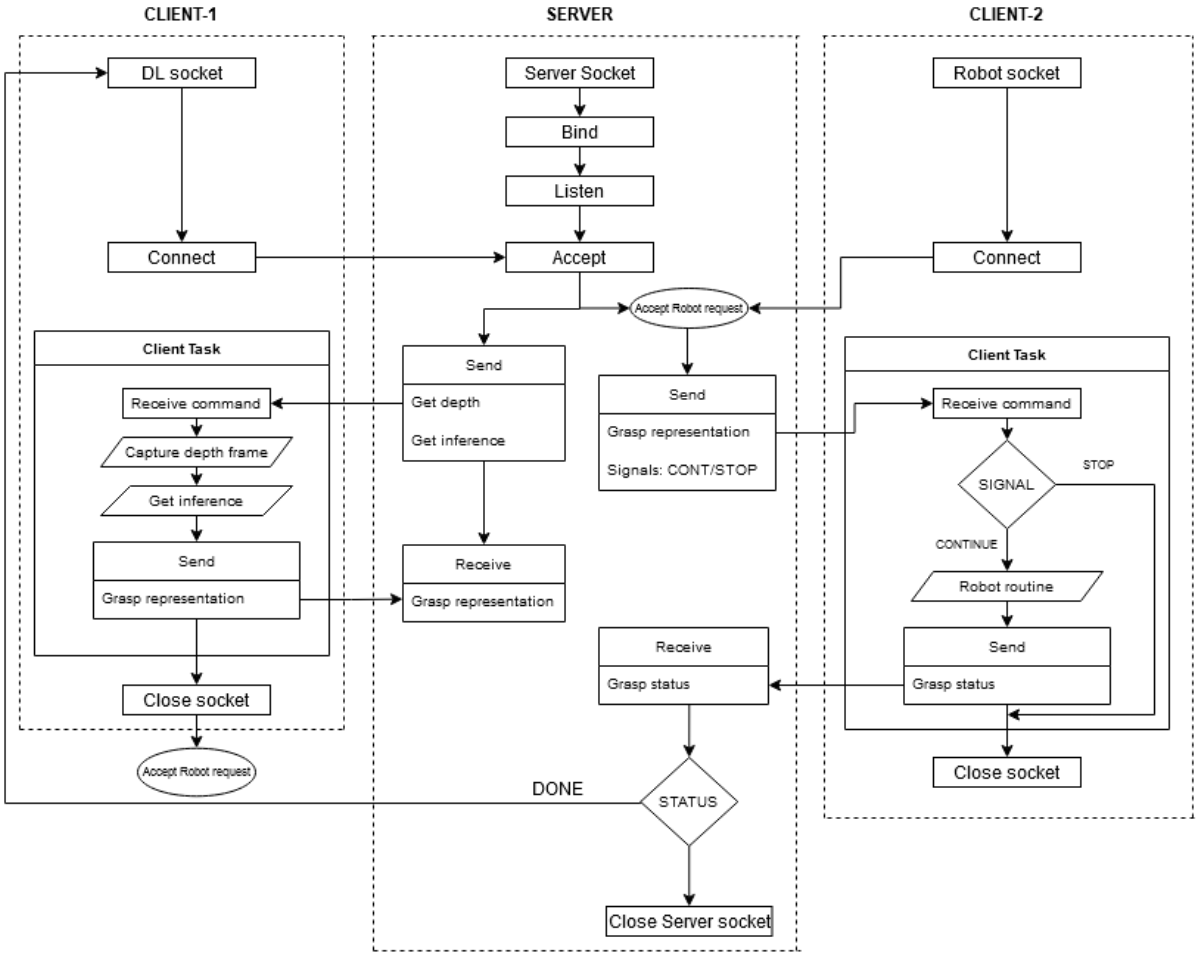


Fig. 6.1 Information Flow Sequence for the Socket Communication

Once the grasp information is received by the python server the socket is closed between the DL client and python server, the next set of execution starts and now the communication is established between the python server and the Robot Studio. At this point in time, the grasp string representation is sent to the Robot studio where the RAPID code is structured to explicitly extract the grasp information from the string and do a pick and place routine. After the routine is done, a message is sent to the python server stating whether the grasp is executed or not. When the python server receives a positive command the socket communication is closed and the pipeline meets the end condition. **Fig. 6.1** gives a systematic schematic of the overall communication and the information flow between the python server and the two clients i.e., deep Learning and robot.

6.2 Communication Information

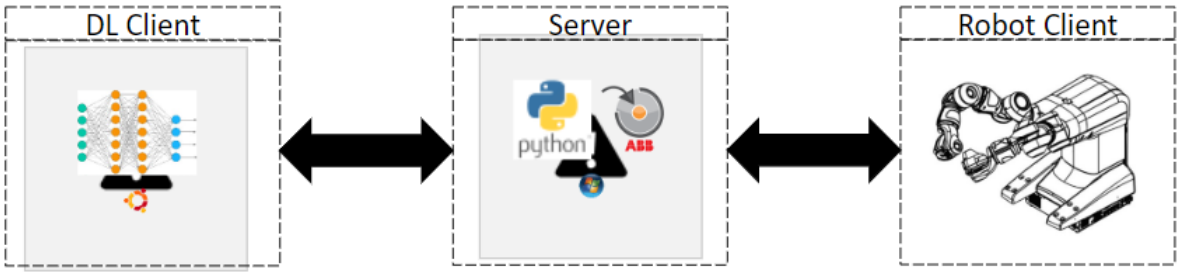


Fig. 6.2 Communication Schematic for the Proposed System

Socket programming was chosen because of its high reliability and In-order timely data delivery. The communication protocol is TCP/IP (Transmission Control Protocol) and the type of socket was chosen to be the stream socket which is available to use under the python framework under the module of the library (*socket.SOCK_Stream)*.

The IP address of the system can be viewed by typing the following command in the prompt/terminal

"*ipconfig*" for windows and the default gateway is the IP address to be used.

"*ifconfig*" for ubuntu and the default IP address will be displayed.

Table 6.1 IP Address Related Details

	Python server	Ubuntu client	Robot Controller	Gripper left	Gripper right
IP address	"*ipconfig*"	"*ifconfig"*	192.168.125.50	192.168.125.40	192.168.125.30
Port	60065	-	-	5001	5000

6.3 Data Flowgraph

Fig. 6.3 explains the sequence of data flow between different systems and the type of data being sent in the subsequent steps of communication. First, the Kinect acquires the depth image and the actual depth values are calculated based on the regressed polynomial. The information now available are the actual depth values in *NumPy.float32* format in a matrix of size (480×640). The segmentation mask of the same dimension (480×640) but of type *NumPy.unit16* and the camera intrinsic parameters which is a (3×3) matrix of type *NumPy.float32.*

This information is provided to the GQCNN 4.0-PJ pretrained model and the inference takes the form of a Rigid Transform object (autolab-core) which contains pose information such as translation, rotation, and the quaternion of the type *float32.* The pose information is encoded into a byte string in the form of *(X, Y, Z, Qw, Qx, Qy, Qz, cf1, cf2, cf3, cfx*) where, X, Y, Z are the translations in metres and *Qw, Qx, Qy, Qz* are the quaternions which essentially capture the rotational aspect of the pose.

cf1, cf2, cf3, cfx are predefined kinematic constraints to be set in Robot Studio to achieve a particular trajectory.

The python server receives the byte string decodes it to display the actual information and sends the encoded string back again to the Robot studio client. The Robot studio receives the byte string and decodes the information to execute the robot routine and a "Task Status" signal which is a simple string is sent back to the server to tell if the routine is complete or not. The information flow stops at this point in time.

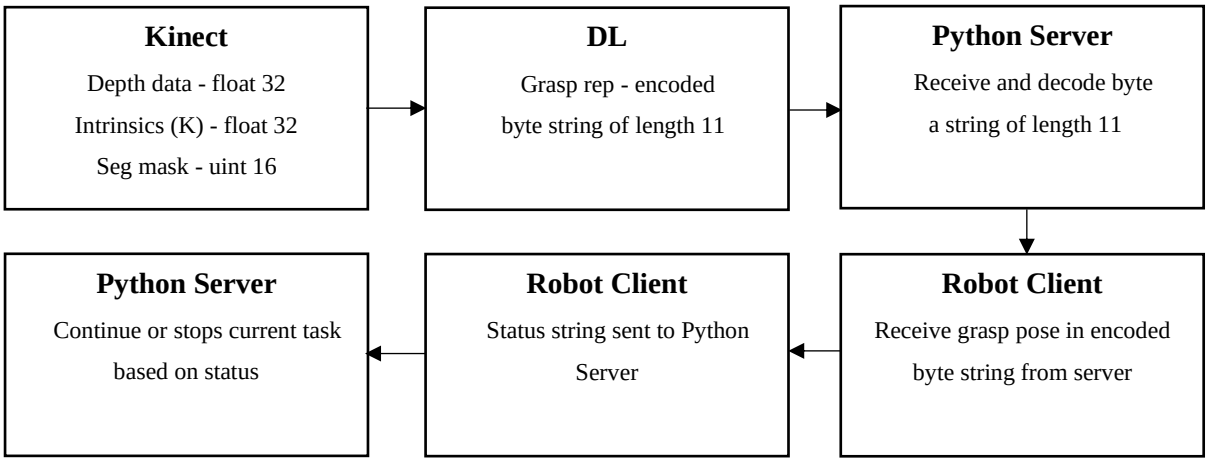


Fig. 6.3 Data Flow Sequence for the Socket Communication

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

The project has achieved all its objectives with a detailed report. Learning-based robotic grasping approaches enable the picking of a diverse set of objects and can demonstrate high grasping success rates even in cluttered scenes and non-static environments. Machine learning and simulation enable quick and easy deployment due to the automatic configuration of model-based solutions and the ability of model-free approaches to generalise to novel objects. Despite promising results, robotic grasping and manipulation still remain a challenge.

Our significant contribution to the project includes the following:

- Opened python-based communication for robot
- Offered 2 ready-available solutions for grasp localization
- A modified model for live inference grasp execution
- Primer for ease of future collaboration
- First-hand guidelines and references for robot-related troubleshooting
- Optimal robot configuration values

Having prepared a strong base for future projects/research in grasp localization will facilitate future collaborators. The following suggestions may be taken into consideration for future research/projects:

i) In the proposed work only pre-trained was used to test the grasp inference for sample objects in the dataset. There is scope to improve the deep learning model's robustness by training the network from scratch for the particular sensor robot scene or a transfer learning-based approach could also be followed where the final layers of the network are trained to give a better performance for the data taken from Kinect.

ii) For the multi-camera setup to acquire dense point clouds, the point clouds were acquired separately to avoid IR interference. The cameras were placed very close to each other to produce point cloud data for approximately the same perspective, as this increases the chances of the algorithm matching the point clouds based on the detected features (MATLAB was used for this).

iii) Dual-arm parallel-jaw grasps: Since DexNet 4.0 has a provision for both parallel-jaw and suction-cup grasps, future work may take inspiration from the same for considering dual-arm grasps.

REFERENCES

1. Kilian Kleeberger, Richard Bormann, Werner Kraus, Marco F. Huber. (2020) '*A Survey on Learning-Based Robotic Grasping*' Current Robotics Reports 1:239-249
2. Jeffrey Mahler, Matthew Matl , Vishal Satish , Michael Danielczuk , Bill DeRose , Stephen McKinley , Ken Goldberg.(2019) *'Learning ambidextrous robot grasping policies*'
3. Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, Dieter Fox.(2021) *'Contact-GraspNet: Efficient 6-DoF Grasp Generation in Cluttered Scenes'* IEEE International Conference on Robotics and Automation
4. Branko Karan.(2015) *'Calibration of Kinect-type RGB-D sensors for robotic applications'* FME Transactions**,** Vol 43, 47-54
5. Xin Zhang, Ruiyuan Wu.(2016) '*Fast depth image denoising and enhancement using a deep convolutional network*' IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)
6. Xuan Liao, Xin Zhang.(2017) '*Multi-scale mutual feature convolutional neural network for depth image denoise and enhancement'* IEEE Visual Communications and Image Processing (VCIP)
7. Junho Jeon, Seungyong Lee.(2018) *'Reconstruction-based Pairwise Depth Dataset for Depth Image Enhancement Using CNN'* Proceedings of the European Conference on Computer Vision (ECCV), pp. 422-438
8. ABM Tariqul Islam, Christian Scheel, Renato Pajarola and Oliver Staadt.(2015) *'Depth Image Enhancement using 1D Least Median of Squares'* Computer Graphics International Short Papers, Strasbourg, France, 24 June 2015 - 26 June 2015. Computer Graphics and Geometry Group of ICube Laboratory, 1-4.
9. Ming-Ze Yuan, Lin Gao, Hongbo Fu, and Shihong Xia.(2019) '*Temporal Upsampling of Depth Maps using a Hybrid Camera'* IEEE Transactions on Visualization and Computer Graphics, Vol.25, Issue. 3
10. Kwan Pang Tsuia, Kin Hong Wongb, Changling Wanga, Ho Chuen Kamb, Hing Tuen Yaub, and Ying Kin Yu.(2016) '*Calibration of Multiple Kinect Depth Sensors for Full Surface Model Reconstruction*' First International Workshop on Pattern Recognition, 100111H

11. Tsuneo Yoshikawa (1985). *'Manipulability of Robotic Mechanisms'* The International Journal of Robotics Research, Vol.4. 3-9
12. Dex-Net dataset - https://berkeleyautomation.github.io/dex-net/
13. Survey analysis on Depth Frame improvement.doc (gdrive)
14. StereoLabs Zed Mini SDK - https://www.stereolabs.com/developers/release/
15. Intel Realsense D435i SDK - https://www.intelrealsense.com/sdk-2/
16. Kinect Studio SDK - https://www.microsoft.com/en-in/download/details.aspx?id=44561
17. Intel Realsense D435i SDK python library- https://github.com/IntelRealSense/librealsense
18. Kinect Python library Libfreenect - https://github.com/OpenKinect/libfreenect
19. Camera Calibration code – (project gdrive https://drive.google.com/drive/folders/1IwosYusVOeNaoTxsjr6fiXrCZtTCPjxX)
20. Image segmentation code – (project gdrive https://drive.google.com/drive/folders/1IwosYusVOeNaoTxsjr6fiXrCZtTCPjxX)
21. Vision aided Grasp localization using Computer Vision solution – (project gdrive)
22. Lorenzo Porzi, Samuel Rota Bulo, Adrian Penate-Sanchez, Elisa Ricci, and Francesc Moreno-Noguer. Learning depth-aware deep representations for robotic perception. IEEE Robotics & Automation Letters, 2016.
23. Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis survey. IEEE Trans. Robotics, 30(2):289–309, 2014.
24. Matei Ciocarlie, Kaijen Hsiao, Edward Gil Jones, Sachin Chitta,Radu Bogdan Rusu, and Ioan A S¸ucan. Towards reliable grasping and manipulation in household environments. In Experimental Robotics, pages 241–252. Springer, 2014
25. Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In Proc. IEEE Int. Conf. on Computer Vision (ICCV), pages 858–865. IEEE, 2011.
26. Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localization and mapping at the level of objects. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pages 1352–1359, 2013

27. Ziang Xie, Arjun Singh, Justin Uang, Karthik S Narayan, and Pieter Abbeel. Multimodal blending for high-accuracy instance recognition. In Proc. , IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pages 2214–2221. IEEE, 2013
28. Saurabh Gupta, Pablo Arbel´aez, Ross Girshick, and Jitendra Malik. Aligning 3d models to RGB-d images of cluttered scenes. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pages 4731–4740, 2015
29. Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker Jr, Alberto Rodriguez, and Jianxiong Xiao. Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge.
30. Lorenzo Porzi, Samuel Rota Bulo, Adrian Penate-Sanchez, Elisa Ricci, and Francesc Moreno-Noguer. Learning depth-aware deep representations for robotic perception. IEEE Robotics & Automation Letters, 2016
31. Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust RGB-d object recognition. In Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pages 681–687. IEEE, 2015.
32. Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In Proc., Advances in Neural Information Processing Systems, pages 2017–2025, 2015.
33. C. Eppner, A. Mousavian, and F. Dieter, “Acronym: A large-scale grasp dataset based on simulation,” IEEE International Conference on Robotics and Automation (ICRA), 2021
34. M. Savva, A. X. Chang, and P. Hanrahan, “Semantically-enriched 3d models for common-sense knowledge,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2015, pp. 24–31.
35. C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” Neural Information Processing Systems (NeurIPS), 2017.
36. YuMi ‘*Product Specification IRB 14000*’ Document ID: *3HAC052982-001* Revision: Q
37. YuMi ‘*Operating Manual IRB 14000*’ RobotWare 6.13 Document ID: *3HAC052986-001*
38. YuMi ‘*IRB 14000 YuMi_DualArm-datasheet*’ Release: 2021-03-31

39. IRC5 Controller *'Application manual Controller software IRC5'* RobotWare 6.13 Document ID: *3HAC050798-001*, Revision: C

40. IRC5 controller '*Operating manual Trouble shooting, IRC5'* Document ID: *3HAC020738-001*, Revision: *K*

41. FlexPendant *'Operating manual IRC5 with FlexPendant'* RobotWare 6.13, Document ID: *3HAC050941-001*, Revision: M

42. RobotWare '*Application manual RobotWare add-ins'* RobotWare 6.13 Document ID: *3HAC070207-001*, Revision: C

43. RAPID '*Technical reference manual: RAPID Instructions, Functions and Data types'* RobotWare 6.13, Document ID: *3HAC050917-001*, Revision: P

44. Smart Gripper '*Product manual IRB 14000 gripper IRC'* Document ID: *3HAC054949-001*, Revision: *M'*

45. Integrated vision '*Application manual Integrated Vision'* RobotWare 6.13 Document ID: *3HAC067707-001*, Revision: *B*

46. *Jacquard : A Large Scale Dataset for Robotics Grasp Detection* – Amaury Depierre, Emmanuel Dellandrea and Liming Chen.

47. Y. Jiang, S. Moseson, and A. Saxena. *Efficient grasping from RGBD images: Learning using a new rectangle representation*. In ICRA, 2011

48. Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea and Ken Goldberg '*Dex-Net 2.0: Deep Learning to Plan RobustGrasps with Synthetic Point Clouds and Analytic Grasp Metrics'*

49. Berkeley Autolab YumiPy documentation *https://berkeleyautomation.github.io/yumipy/*

APPENDIX – I

DEEP LEARNING

I.1 Implementation Details

The pre-trained Dex Net 4.0 model for Parallel Jaw (GQCNN-4.0-PJ) is used for getting the grasp candidates concerning the camera. The model weights and checkpoints are obtained from the "GQCNN" official repository. There are models available for the suction type of gripper too if that's of any interest in the future with the pre-trained model being termed GQCNN-4.0-SUCTION. A step-by-step procedure to implement the code for GQCNN-4.0-PJ is given below:

- Download installer setup for anaconda 5.2.0 which ships with Python version 3.6 (Link Windows - https://repo.continuum.io/archive/Anaconda3-5.2.0-Windows-x86_64.exe)
  (Link Linux - https://repo.continuum.io/archive/Anaconda3-5.2.0-Linux-x86_64.sh)
- Download or clone the repository for the GQCNN version 1.3.0 (.zip,.tgz) from the link (https://github.com/BerkeleyAutomation/gqcnn/releases/tag/v1.3.0) and unzip it in the directory you wish to put it.
- It is advised to have everything installed in (base) of the Anaconda package, however, a new environment can be created with python version 3.6.
- Install Tensorflow-GPU or Tensorflow-CPU in the environment where you are planning to use GQCNN version 1.3.0

  Version Info for Tensorflow-CPU: tensorFlow-1.15.0

  Version Info for Tensorflow-GPU: tensorflow_gpu-1.15.0 , cuDNN = 7.4 , CUDA = 10

  The packages can be installed using "*pip*" or using "*conda install*".

  (For reference, "*pip install tensorflow==1.15.0*" or "*conda install tensorflow=1.15.0*")

  It is advisable to use conda install as it takes care of dependencies in the activated environment.

- Make the working directory to (gqcnn-1.3.0) as all the commands will be executed with this as a base folder. Run the command "*pip install .",* to install gqcnn and other requirements to run the gqcnn package.

- autolab-core, autolab-perception, visualization, numpy, opencv-python, scipy, matplotlib, tensorflow-gpu<=1.15.0, scikit-learn, scikit-image, gputil, psutil
- Make sure the above-mentioned packages are installed as part of the pip installation before proceeding further.
- The ROS dependencies require you to install the autolab-core package as a catkin package in the catkin workspace to utilize its functionality (If ROS is used in the project kindly look into ROS installation in gqcnn documentation)
- The pre-trained model is tested for inference in Ubuntu 18.04.6 LTS (Bionic Beaver).

I.2 MODIFICATIONS TO RUN PRE-TRAINED GQCNN-4.0-PJ:

- Go to the directory (gqcnn-1.3.0/scripts/downloads) execute the command **"***chmod + x download_models.sh***"** to enable access to run shell commands.
- Run the shell file using the command **"***sh download_models.sh***"**. This should start the installation of pre-trained models inside a ("model") directory with files including GQCNN-2.0, GQCNN-2.1, GQCNN-3.0, GQCNN-4.0-PJ, GQCNN-4.0-SUCTION, FC-GQCNN-4.0-PJ, FC-GQCNN-4.0-SUCTION.
- Cut the "**scripts**" folder which is present inside the gqcnn-1.3.0 directory and paste it into the "**examples**" folder.

I.3 MODIFICATIONS IN PYTHON SCRIPTS:

- Go to ***policy.py*** inside "gqcnn-1.3.0/examples" and change line 121-122 to the following,

  " os.path.join(os.path.dirname(os.path.realpath(__file__)),
  "scripts/downloads/models/models") "

  This is to set the model directory to your pre-trained model from which the configuration file, and checkpoints file will be loaded for your pre-trained model.
- Make sure you have the following files in, "gqcnn-1.3.1/examples/scripts/downloads/model/model/GQCNN-4.4-PJ" architecture.json, checkpoint, config.json, im_mean.npy, im_std.npy, model.ckpt.data, 00000-of-00001, model.ckpt.index, model.ckpt.meta, pose_mean.npy, pose_std.npy
- These files are required to get inference from ***network_tf.py*** which is present in "gqcnn-1.3.0/gqcnn/model/tf".
- Go to the python script ***network_tf.py*** and change line 42 to "import TensorFlow.train as tcf" , as the original contrib package is now deprecated and the functionality for (list_variables) is switched to train module.

- Add the following lines at the start of the ***network_tf.py*** script to suppress all warnings during runtime of python policy,
  import warnings
  warnings.filterwarnings("ignore")
- Run the sample python policy as given in the documentation for the sample dataset present in "gqcnn-1.3.0/data/examples"
- To execute the inference go-to base folder which is gqcnn-1.3.0 then run the following python policy in the command prompt as given below,
- *python examples/policy.py GQCNN-4.0-PJ --depth_image data/examples/clutter/phoxi/dex-net_4.0/depth_0.npy --segmask data/examples/clutter/phoxi/dex-net_4.0/segmask_0.png --camera_intr data/calib/phoxi/phoxi.intr*
- The policy can be modified to run inference for a single object as well by changing the depth image and segmask directory to the following,
  "data/examples/single_object/primesense"
- To view other grasp parameters add the following lines after line 255 of ***policy.py***

```
print("grasp pose details : {}".format(action.grasp.pose()))
print("Approach axis: {} , Approach angle:{}".format(action.grasp.approach_axis,action.grasp.approach_angle))
print("Contact Points: {}".format(action.grasp.contact_points))
print("Width: {} , Width_px: {}".format(action.grasp.width,action.grasp.width_px))
print("Centre: {}".format(action.grasp.center))
print("Endpoints: {}".format(action.grasp.endpoints))
logger.info("Planning took %.3f sec" % (time.time() - policy_start))
```

- The training dataset of size 8.0 GB can be downloaded for GQCNN-4.0-PJ (parallel jaw) in the following link,

  https://berkeley.app.box.com/s/6mnb2bzi5zfa7qpwyn7uq5atb7vbztng

As part of the deep learning network exploration significant time was also invested in installing the dependencies of PointNetGPD which is again a point cloud based network which works on ros as well as python interface. The step by step procedure during the installation process is given below.

1. Create new folder named code in home directory

2. Create anaconda environment pointnetgpd with python 3.8+ as the first installation when proceeded according to the instruction threw up a error
3. Install pcl-tools via sudo apt install pcl-tools.
4. Install pytorch separately acc to machine dependency
   - conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
5. Clone the repository in code folder and create anaconda env variable as it is being used as home_dir variable in sub routines.
6. Go to location of conda environment “pointnetgpd” and create the following files in the path (“home/harik/anaconda3/envs/pointnetgpd/etc/conda…..”)
   mkdir -p ./etc/conda/activate.d
   mkdir -p ./etc/conda/deactivate.d
   touch ./etc/conda/activate.d/env_vars.sh
   touch ./etc/conda/deactivate.d/env_vars.sh
7. export PointNetGPD_FOLDER = ‘/home/harik/code/PointNetGPD’
   - To be added in activate.d/env_vars.sh
8. unset PointNetGPD_FOLDER
   - To be added in deactivate.d/env_vars.sh
9. When the environment is activated (PointNetGPD_FOLDER) is available to use and is erased when “pointnetgpd” environment is deactivated
10. The environment variable was tested for path in home ./bashrc and after sourcing The path variable didn’t seem to update because conda uses it’s own bash , the Existence of path variable was tested using “conda env config vars list”.
    Meshpy and Dex-net setup.py develop ran successfully in env
11. The gripper configuration was set to default as the values were hard to find to test out on a sample.
12. Clone ycb-tools in “$PointNetGPD_FOLDER/PointNetGPD/data”
13. Modify the script download_ycb_dataset.py to make the code download only Specific objects as each object file is around 600MB (no space in Vm to download the entire dataset due to memory constraints)
14. Downloaded 3 object categories and the folder structure was changed.
15. The python-pcl has been outdated to be used in point net gpd and the code needs to be updated by contributors to support python 3.x.

APPENDIX – II

ROBOTICS AND CONTROL

II.1 Commonly used RAPID programming features

Table II.1 Commonly Used Programming Features

DATA TYPE			
	Name	**Description**	**Example**
Basic	bool	Logical true/false	VAR bool reachable:=TRUE;
	dnum	Numeric value	<TYPE> dnum dist:= 60.17;
	errnum	Error number	IF ERRNO=ERR_ROBLIMIT THEN …..
	num	Numeric number	<TYPE> num count:= 1;
	string	String	<TYPE> string name:= “Stanley”;
Robot Motion	confdata	Robot configuration	confdata:=[-1,0,1,4];
	robconf	Specify robot configuration	<robTarget>.robconf.cf1 := -1;
	jointtarget	Robot joint values	CONST jointtarget Home:=[];
	trans	Cartesian translation	<robTarget>.pos.x:= 150
	robjoint	Joint position	(Refer RAPID Instruction Manual)
	robtarget	Cartesian pose of robot	<TYPE> robTarget Pick:= [[x ,y, z],[qw,qx,qy,qz],[-cf1,cf4,cf6,cfx],[-extAxis ,9E+09,9E+09,9E+09 ,9E+09,9E+09]];
	tooldata	Characteristics of tool	PERS tooldata TGripL:=[TRUE,[[-0.233031,-0.111434,135.858],[1,0,0,0]],[0.244,[7.5,11.8,52.7],[1,0,0,0],0.00021,0.00023,8E-05]];
Com	socketdev	Socket device	VAR socketdev serverSocket;
	socketstatus	Socket communication status	(Refer RAPID Instruction Manual)
INSTRUCTIONS			
Basic	IF ELSE		IF stat="CONTINUE" THEN …. ELSE ….ENDIF
	FOR	For loop	FOR <condition> ….. ENDFOR
	EXIT	Exits routine	(Refer RAPID Instruction Manual)
Rob	ConfJ	Configuration supervision for MoveJ	ConfJ\On; ConfJ\Off;

	ConfL	Configuration supervision for MoveL	ConfL\On; ConfL\Off;
	GripLoad	Set gripper load	(Refer RAPID Instruction Manual)
	MoveAbsJ	Move command by setting joint angles	MoveAbsJ Home,v100\T:=5,z50,tool_manual;
	MoveC	Move in circular path	(Refer RAPID Instruction Manual)
	MoveJ	Non-linear cartesian-space move command	MoveJ Axis,v100,z50,tool_manual\WObj:=wobj0;
	MoveL	Linear cartesian-space move command	MoveL Pick,v20,z50,tool_manual\WObj: =wobj0;
	g_Init	Initialize gripper	g_Init\maxSpd:=20,\holdForce:=8,\Calibrate,\Grip;
	g_GripIn	Retract parallel jaw	g_GripIn;
	g_GripOut	Extend parallel jaw	g_GripOut;

Communication	SocketAccept	Accept incoming connection request	(Refer RAPID Instruction Manual)
	SocketBind	Bind to a IP address and port	(Refer RAPID Instruction Manual)
	SocketClose	Close a socket	SocketClose serverSocket;
	SocketConnect	Connect request to a server	SocketConnect clientSocket ,"192.168.125.50", 60064\Time:=WAIT_MAX;
	SocketCreate	Create a socket	SocketCreate clientSocket;
	SocketListen	Server listens for incoming connection requests	(Refer RAPID Instruction Manual)
	SocketReceive	Receives a byte string from a remote client/server	SocketReceive clientSocket \Str := data;
	SocketSend	Sends a byte string from a remote client/server	SocketSend clientSocket\Str:="Received data";
FlexPen	TPErase	Clear FlexPendant screen	TPErase;
	TPRead	Get user input	(Refer RAPID Instruction Manual)

	TPWrite	Display on FlexPendant	TPWrite "Task completed";
	WaitRob	Wait for robot to complete task	WaitRob\InPos;
	WaitTime	Wait for a specified time	WaitTime 2;
	WaitUntil	Wait until a condition is met	(Refer RAPID Instruction Manual)
FUNCTIONS			
	Abs	Absolute value	(Refer RAPID Instruction Manual)
	AND	Logical AND	IF x=1 AND y=1 THEN …
	OR	Logical OR	IF x=1 OR y=1 THEN …
	DnumToStr	Converts double numerical value to string	(Refer RAPID Instruction Manual)
	NumToStr	Converts numerical value to string	(Refer RAPID Instruction Manual)
	StrToVal	Converts string to value	(Refer RAPID Instruction Manual)
Robot	CalcJointT	Compute joint target from robot target	joints:=CalcJointT(pose,tool\Wobj:=wobj);
	CalcRobT	Compute robot target from joint target	pose:=CalcRobT(joints,tool\Wobj:=wobj);
	CRobT	Get current robot's Cartesian position	(Refer RAPID Instruction Manual)
	SocketGetStatus	Get status of socket	(Refer RAPID Instruction Manual)

II.2 Robot Configuration data

As specified earlier, the robot configuration depends on the quadrants achieved by the respective axis i.e 1,4 and 6. Cfx however is determined by the position of the wrist center w.r.t the lower arm and the sign of axis-2 and axis-5. Robot configuration can also be calculated from jointtarget by using CalcRobT function.

Table II.2 Robot Configuration Data

Angle	Configuration data	Angle	Configuration data
-45° to +45°	0	-45° to -135°	-1
45° to 135°	1	-135° to -225°	-2
135° to 225°	2	-225° to -315°	-3
225° to 315°	3	-315° to -360°	-4

Note: Robot Configuration data gets revised in every RobotWare version. This data is for RobotWare v6.13 only. Incase of a different version, refer *Technical reference manual – RAPID instructions, Functions and Data types* for the specific version of RobotWare.

II.3 Tools and Frameworks Used

Given is a list of all the tools/modules used for robotics and control.

Table III.1 Tools and Frameworks Used

Name	Module	Version	Purpose
Anaconda	Spyder IDE	3.0	Python-based server
MATLAB	Robotics Toolbox	2021b	Kinematic analysis: Reachability, Manipulability, Inverse Kinematics
	Rotation Viewer	2021b	Rotation visualizer
RobotStudio	-	21.4.97	RAPID programming
	RobotWare	6.13	FlexPendant OS
	Smart Gripper	3.56	Firmware: 5.x

II.4 Troubleshooting guide

Each fault or error is first detected as a symptom, for which an error event log message may or may not be created. It could be an error event log message on the FlexPendant, an observation that the robot isn't behaving consistently or that the controller cannot be started. Common faults displaying an event log message along with common observational inconsistencies are listed with the recommended solutions.

RobotStudio And Controller Side:

1) <u>Communication related:</u>

- SOCKET_TIMEOUT
  Cause: Controller has specific default listening time
  Recommended solution: 1) Set maximum waiting time for accepting connection

Example: SocketAccept serverSocket, clientSocket, \Time:=WAIT_MAX

- Socket error (41569)
  Cause: The socket is already connected and cannot be used to listen for incoming connections.
  Recommended solution: 1) Close the socket and retry
  Example: SocketClose <socket>
  2) Use another socket
- Socket error (41570)
  Cause: The socket cannot accept incoming connection requests since it is not set to listen state. SocketAccept is used before SocketListen
  Recommended solution: 1) Set socket to listen for incoming connections before trying to accept
  2) If in While loop, keep SocketAccept outside loop
- Socket error (41575)
  Cause: The specified address is invalid.
  Recommended solution: Set correct controller service port IP address i.e., 192.168.125.1
- Socket error (41597)
  Cause: The socket has already been bound to an address and cannot be bound again.
  Recommended solution: Close socket and recreate before trying to bind socket to a new address.
- Value error (40205)
  Cause: String value exceeds the maximum allowed length.
  Recommended solution: Rewrite the program to use strings of less length.

2) <u>Manipulator related:</u>

- 10052/10053, Regain start/Regain ready
  Cause: Robot arms in very odd position
  Recommended solution: 1) Jog to a reasonable configuration
  2) If jogging causes motion supervision/jogging error, enable lead-through and move the arm manually joint by joint starting from base
- 50024, Corner path failure
  Cause: Time delay, closely programmed points, System requires high CPU-load.
  Recommended solution: 1) Reduce speed (worked in our case)
  2) Reduce number of instructions between consecutive move instructions
- 50026, Close to singularity
  Cause: Robot configuration too close to singularity
  Recommended solution: 1) Modify path away from singularity point
  2) Change to joint interpolation
- 50028/50028, Joint out of Range/Jog in wrong direction
  Cause: Specified target is out of working range for joint
  Recommended solution: 1) Specify within working range
  2) Jog to bring to working range if stuck
- 50056, Joint collision
  Cause: Actual torque on joint value is higher than ordered while at low or zero speed. Might be caused by jam error (the arm has got stuck) or hardware error.

Recommended solution: 1) Jog arm manually if arm is stuck
2) Calibrate the arm again

- 50060, Incorrect tool
  Cause: Toodata identifier not specified
  Recommended solution: 1) Create toolData and objectData
  Example: PERS tooldata TGripR: =
  [TRUE,[[0,0,0],[1,0,0,0]],[0.1,[0,150,150],[1,0,0,0],0,0,0]];
- 50065, Kinematics error
  Cause: The destination of the movement is outside the reach of the robot or too close to a singularity
  Recommended solution: 1) Change the destination position
- 50076, Orientation not correct
  Cause: Orientation is incorrectly defined
  Recommended solution: 1) Make an accurate normalization of the quaternion elements
- 50076, Kinematic limitation
  Cause: Position outside reach, Joint 1,2 or 3 out of range, long segment
  Recommended solution: 1) Use MoveAbsJ
  2) Insert another via point to reduce length of segment
- 50200, Torque error
  Cause: High load data, High speed
  Recommended solution: 1) Lower load data (For ABB Yumi < 0.5)
  Example: PERS tooldata
  TGripR:=[TRUE,[[0,0,0],[1,0,0,0]],[0.1,[0,150,150],[1,0,0,0],0,0,0]];
  2) Reduce tool/joint speed
- 50204, Motion supervision
  Cause: Collision, incorrect load definition, External force on arm
  Recommended solution: 1) If exposed to forces, use RAPID command/system parameters to raise the supervision level
  2) Correct load definitions
- 50436, Robot configuration error
  Cause: Cannot reach robot configuration or must pass through singular point
  Recommended solution: 1) Use SingArea\Wrist and ConfL\Off
  2) Replace MoveL by MoveJ
- 50474, Target in a singularity
  Cause: Robot target is near singular (some joints might also be close to 0 deg)
  Recommended solution: 1) Use SingArea instruction or MoveAbsJ
  2) During jogging, use axis by axis
- 80001: Error IOEnable Hand_R

3) Robot Routine related:

1. **Inconsistent path accuracy:** A robot TCP's pose varies from time to time and is sometimes accompanied by noise emerging from bearings, gearboxes, or other locations. Hence, the path of the robot's TCP may not be consistent.
   Possible causes:
   i. Robot arm not calibrated correctly
   ii. TCP calibration/definition problem
   iii. Brakes not releasing correctly

iv. Mechanical coupling between link and joints damaged
v. Bearing damaged or worn (accompanied by harsh noises)
vi. Configuration supervision switched off
vii. Wrist singularity check switched on
viii. Wide zone region in move command(s)

4) FlexPendant/UI related:

- 41617, Too intense frequency of Write Instructions
  Cause: A high usage frequency of user interface write instructions, such as TPWrite, has forced the program execution to slow down.
  Recommended solution: 1) Use WaitTime
- 40708: I/O Device Is Not Enabled
  Recommended solution: Recovery: ERR_IOENABLE.
- Problem jogging the robot
  Cause: Joystick damaged or robot not in manual mode.
  Recommended solution: 1) Switch to manual mode
  2) Make sure flex pendant is connected properly to controller
  3) Reset the Flex Pendant if none of these work
  Note: Resetting the flexpendant doesn't reset the system on the controller.

Python-client side:

1) Communication related:

- [WinError 10061] No connection could be made because the target machine actively refused it
  Cause: Socket not open to listening, Server socket not created, Client ran before server
  Recommended solution: 1) Start server before running python-client
  Example: SocketClose <socket>
  2) Create a server socket, listen and accept connections

Software related:

1) Gripper firmware related:

Version	HandDriver	EIO	Firmware
3.56	1.12.6	1.4	5.x

2) Gripper communication related

- 71058, Lost communication with I/O device
  Cause: Gripper isn't sending signal to the controller due to connection failure.
  1) Wrong IP address
  2) Gripper to flange loose connection
  3) Gripper's communication module damaged
  Recommended solution: 1) Correct the IP address
  2) De-commission the gripper and re-attach
- 71367, No communication with I/O device
  Cause: Gripper isn't communicating with the controller

1) Robot has just booted
2) Wrong IP address

Recommended solution: 1) Wait for about 10 seconds after the robot has booted
2) Correct the IP address

3) Gripper software commissioning

When installing grippers for the first time or changing grippers on a robot, the Left and Right identity (chirality) has to be set up. The chirality is controlled by different IP addresses. For Right and Left, these are 192.168.125.30 and 192.168.125.40, respectively. These are the only possible IP addresses for a gripper. The default IP address for a gripper is 192.168.125.30. That is, grippers are delivered to users as Right grippers.

Procedure for changing chirality:

i) Shut off power to the Left flange using the configuration option under Smart Gripper view.

ii) Turn on the power to the Right flange and verify that the LED on the Right gripper is ON.

iii) Verify that the gripper on the Right flange is set up as the Right gripper.

• If yes, proceed to step v).
• If no, switch the IP address of the gripper using the FlexPendant Gripper interface.

iv) Reboot the gripper by shutting power OFF and then ON.

v) Shut off the power to the Right flange using the FlexPendant gripper interface.

vi) Turn on the power to the Left flange and verify that the LED on the Left gripper is ON.

vii) Verify that the gripper on the Left flange is set up as the Left gripper.

• If yes, the setting ends.

• If no, switch the IP address of the gripper using the FlexPendant.

viii) Reboot the gripper by shutting the power OFF and then ON

4) RobotWare version related

Note that ABB revises RAPID functions and definitions for every new version of RobotWare. Hence it is vital to have the RobotWare compatible SmartGripper version and firmware. RAPID instructions are also subject to change, hence refer to the manual for the corresponding version of RobotWare only.

505

ORIGINALITY REPORT

14%	12%	6%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	serioussurvivor.com Internet Source	2%
2	arxiv.org Internet Source	1%
3	idoc.pub Internet Source	1%
4	digitalassets.lib.berkeley.edu Internet Source	1%
5	v1.overleaf.com Internet Source	1%
6	downloads.ctfassets.net Internet Source	1%
7	export.arxiv.org Internet Source	1%
8	elib.dlr.de Internet Source	1%
9	library.e.abb.com Internet Source	1%