DEVELOPMENT AND EVALUATION OF A DEEP LEARNING BASED ROBOTIC BIN PICKING VISION SYSTEM USING ROS

A PROJECT REPORT

Submitted by

AVINASH SEN

TCR18MEMS04

to

the APJ Abdul Kalam Technological University

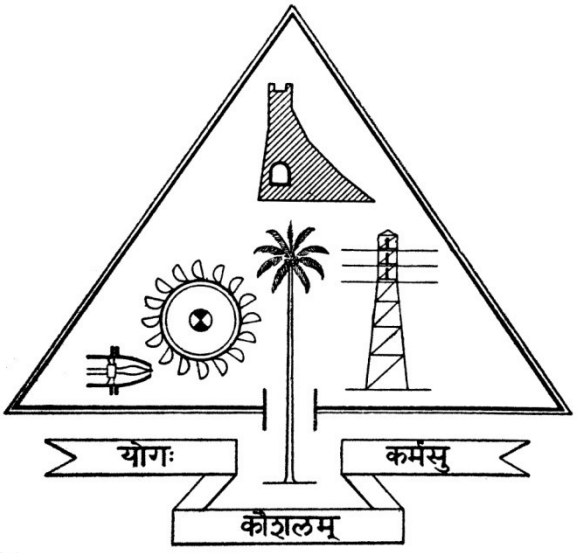
in partial fulfilment of the requirements for the award of the Degree

of

Master of Technology

In

MANUFACTURING SYSTEMS MANAGEMENT



Department of Production Engineering

Government Engineering College Thrissur

Thrissur - 680009

JUNE 2020

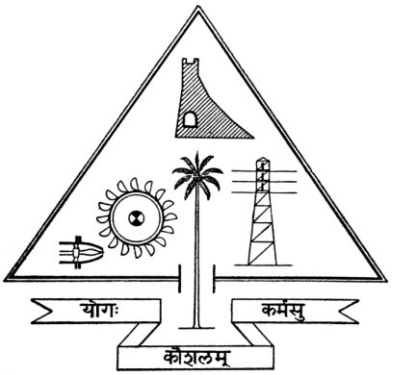
DECLARATION

I, **AVINASH SEN (Univ. Reg. No. TCR18MEMS04)** hereby declare that the project report **"DEVELOPMENT AND EVALUATION OF A DEEP LEARNING BASED ROBOTIC BIN PICKING VISION SYSTEM USING ROS"**, submitted for partial fulfillment of the requirements for the award of degree of Master of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under supervision of **Aju S S**, Associate Professor Dept. of Production Engineering, Government Engineering College, Thrissur. This submission represents my ideas in my own words and where ideas or words of others have been included; I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: GEC Thrissur

Date: 09/06/2020 AVINASH SEN

Department of Production Engineering



GOVERNMENT ENGINEERING COLLEGE THRISSUR

Certificate

This is to certify that the Project Report titled **"DEVELOPMENT AND EVALUATION OF A DEEP LEARNING BASED ROBOTIC BIN PICKING VISION SYSTEM USING ROS"** is a bonafide record of the work carried out by **AVINASH SEN (Univ. Reg. No. TCR18MEMS04)** of GOVERNMENT ENGINEERING COLLEGE, THRISSUR, in partial fulfillment of the requirements for the award of **Degree of Master of Technology** in **MANUFACTURING SYSTEMS MANAGEMENT** of **APJ Abdul Kalam Technological University, Thrissur Cluster**. The Project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

INTERNAL SUPERVISOR	Head of Department
Aju S S	Dr. Manjith Kumar
Asst. Professor	Professor & Head
Dept. of Production Engineering	Dept. of Production Engineering
.....	.....

PRINCIPAL
Dr.Sheeba V S

This project report was evaluated by us on ..../..../.....

PG Coordinator	EXTERNAL EXAMINER
Dr. Sathish K P	.....

ACKNOWLEDGEMENT

During the course of my project work several persons collaborated directly and indirectly with me. Without their support it would be impossible for me to finish my work. That is why I wish to dedicate this section to recognize their support.

I want to start expressing my thanks to my project guide **Aju S S**, Asst. Professor Dept. of Production Engineering, because of his valuable advice and guidance towards this work. I received motivation; encouragement and hold up from his during the course of work.

I am thankful to **Dr. Manjith Kumar**, Head of Production Engineering Department, and our Principal **Dr. Sheeba V.S**, for their sole co-operation.

I would like to thank our Project Coordinator **Dr. Satish K.P** for his sincere support throughout the project presentation.

I am grateful to express my thanks to all the faculty members of our Production Engineering department for their support. I articulate my gratitude to all my associates and colleagues for their support and help for this work.

Last, but not the least I wish to express my gratitude to God Almighty for His abundant blessings without which this effort would not have been successful.

AVINASH SEN

Reg. No. TCR18MEMS04

MANUFACTURING SYSTEMS MANAGEMENT (2018 Admissions)

GOVERNMENT ENGINEERING COLLEGE THRISSUR

ABSTRACT

The technique used by a robot to grab objects that are randomly placed inside a box or a pallet is called bin picking. Bin picking has evolved greatly over the years due to tremendous strides empowered by advanced computer vision technology, software development and gripping solutions. However, the creation of a versatile vision system, capable of collecting any type of object without deforming it, regardless of the disordered environment around it, remains a challenge. The solution for this problem can be achieved by learning the appearance of the object in the bin using convolutional neural network(CNN). In the dataset that is used for training has photorealistic images with precisely annotated 3D pose for all objects. Such a dataset is generated by synthetically overlaying object models using Blender tool and backgrounds with difficult composition and high graphical feature with the help of Unreal Engine(UE4) and NVidia Deep Learning Data Synthesizer(NDDS) software. By Deep Learning the created model, the object poses with adequate accuracy required for semantic grasping by any robot is obtained. The overall system is implemented using ROS framework.

Keywords: Convolutional Neural Networks, Pose Estimation, Computer Vision, Synthetic Data, ROS framework

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

SCARA	Selective Compliance Assembly Robot Arm
ROS	Robot Operating System
DOPE	Deep Object Pose Estimation
CAN	Controller Area Network
XML	eXtensible Markup Language
TF	Transform Frame
STL	Standard Tessellation Language
FBX	Filmbox File
UE4	Unreal Engine 4
NDDS	NVIDIA Deep learning Dataset Synthesizer
IR	INFRARED
YCB	Yale-CMU-Berkeley (YCB) Object
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
DOF	Degree Of Freedom
URDF	Unified Robot Description Format
PNP	Perspective-n-Point
RVIZ	ROS visualization
DR	Domain Randomization
IMU	Inertial Measurement Unit
VGR	Vision-guided robots
API	Application Program Interface
SLAM	Simultaneous Localization And Mapping
SDK	Software Development Kit
TCP	Tool-Center Point
TCP/IP	Transmission Control Protocol/Internet Protocol

NOTATIONS

μm	Micrometer
°	Degree (**deg)**
D	Dimension
K	Thousand(1000)
Hz	Hertz (Hz)
σ	Standard Deviation

Chapter 1

INTRODUCTION

As companies identified the increase in the demands on how labor-intensive tasks are performed, there was a growing need to automate as many manual processes as possible, thus ensuring improvements in quality and consistency while simultaneously decreasing production time. Due to the accelerated evolution in technology and the continuous production of sensors, instruments and more intelligent systems, the automation industry continues to evolve in a stable and solid way.

One of the fastest growing automation strands is certainly industrial robotics. According to the International Federation of Robotics, industrial robot sales rose 15% in 2015 and ABI Research estimates that sales in this industry will triple by 2025 [1]. The applications of the robots are quite diverse; however, high precision, productivity and flexibility in the tasks performed are guaranteed [2].

1.1 FRAMEWORK AND MOTIVATION

Many robots are used to perform processes where decision-making is required in order to act correctly, so that without the sensitivity of the outside world, robots are only able to repeat pre-programmed tasks. The use of auxiliary equipment, namely sensors, is essential in decision-making in order to have a flexible robotic system.

The evolution of the perception systems was crucial to promote the development of robotic systems that were more versatile and able to perform tasks of greater complexity. Machine Vision and Robot Vision, together with the Image Processing and Computer Vision techniques used to improve the quality of images and extract information from them, respectively, are the best global perception systems used since no contact with the outside world is required [3].

In an industrial environment, the objects and materials used are not always arranged in an organized and structured way, therefor there is a need to automatically manipulate objects in these situations. Before the introduction of vision systems in the industry in the 1990s [4], it was up to workers and other very complex fixed automation systems to position and guide parts before being manipulated by a robot. This was monotonous, dangerous, repetitive work with high labour costs, and there was a risk of high human error in production. It is within the scope of the search for solutions to these problems that bin picking arises.

Bin picking is the name of the technique used by a robot to grab objects that are arranged randomly inside a box or on a pallet. Figure 1.1 shows an illustration of a bin-picking installation.

Figure 1.1 A bin picking infrastructure example

Together with vision systems and intelligent sensors, the robot is able to analyse a surface, detect the objects to be grabbed and move them to the desired location without the need to spend a lot of programming time. However, the process of detecting objects and identifying their correct positioning is a trivial task when carried out by a person, but of high difficulty when performed by a machine, making this one of the greatest challenges in the area of industrial robotics, especially because of the complexity of perception [5]. This challenge is because, although robots are very advantageous when it comes to repeatability, the bin-picking process requires high accuracy in the face of an environment of disorder. The robot must be able to locate the piece in an unstructured environment where objects are constantly changing position and direction whenever something is removed from the box. This requires a delicate balance between the manipulator's dexterity, artificial vision, software, computing power to process all this information and the projection of a viable solution to the manipulator's grip in order to extract the parts without causing any damage and collision.

This technique has been researched by numerous organizations for over 3 decades, and one of the first attempts to create such a robust system was developed in 1986 at MIT [6]. Initially only 2D images were used, coupled with distance sensors, to acquire data, but the continuous technological evolution allowed this to be done with 3D devices.

In order to improve the bin-picking technique, several problems need to be overcome [6]:

- Difficulties in determining the correct geometry, position and direction of the object due to being in a three-dimensional environment surrounded by randomly distributed pieces;
- Ensure adequate and constant lighting of the workspace. If there is a large variation, there may be shadows that make it impossible to correctly detect the part to be collected;
- Cases of overlap and occlusion of parts;
- Correct determination of how to grab the part and the force to be exerted by pressing on it;
- Definition of the most adequate trajectory so as not to collide with other parts, containers and devices;
- For industrial implementation, cycle speeds must also be taken into account;

Presently there are three main forms of bin picking:

- Structured Bin picking - The parts are positioned in the container in an organised and predictable way so that they are easily detected and collected.
- Semi-Structured Bin-picking - The parts are in the container with some organisation to assist in the process of image processing and collection.
- Random Bin-picking - In this case the parts do not have any direction and specific position, having all different directions, they can be overlapped and/or interlaced, thus complicating the process of detection and collection of the desired part to the maximum.

The first two types presented are notoriously simpler processes to execute and can be quickly and easily implemented with technologies already on the market. In contrast, the Random Bin-picking process is considered the ultimate goal for creating a vision-controlled robot, also known as VGR (Vision guided robots) [6]. However, the creation of this type of versatile and precise system, capable of collecting any type of object without deforming it, regardless of the disordered environment around it, remains the main objective. Although several companies have already proposed different solutions to date, these are indicated to solve specific problems and are still not sufficiently versatile. In addition, most of the existing processes are used to grab non-fragile objects, due to the high degree of precision needed to avoid deforming sensitive objects.

1.2 PROBLEM DESCRIPTION

Currently, although there are bin-picking processes that use 2D sensors, in some tasks the use of 3D perception is unavoidable and therefore it is necessary to use 3D sensors. In more modern projects for the recreation of the Random Bin-picking process, the point-cloud survey method has been used, which replicates the outside world through low-cost sensors such as Realsense. Intel's Realsense-type sensors, although very economical, have limited accuracy within certain operating ranges (from several millimeters to several centimeters), and are not suitable for tasks where measurements have to be more accurate, or very close of the target (less than 40 or 50cm). On the other hand, there are industrial sensors for measuring distances with high precision (1 mm or better) but they have only one beam, as opposed to 3D cameras that generate a cloud of points.

In order for robots to operate safely and effectively alongside humans, they must be aware of their surroundings. One aspect of this awareness is knowledge of the 3D position and orientation of objects in the scene, often referred to as 6-DoF (degrees of freedom) pose. This knowledge is important to perform pick and place of objects, handoff from a person, or watch someone handle the object for imitation learning.

While deep neural networks have been successfully applied to the problem of object detection in 2D [7] [8] [9], they have only recently begun to be applied to 3D object detection and pose estimation [10] [11] [12]. Unlike 2D object detection, it is prohibitive to manually label data for 3D detection. Due to this difficulty of collecting sufficiently large amounts of labeled training data, such approaches are typically trained on real data that are highly correlated with the test data (e.g., same camera, same object instances, and similar lighting conditions). As a result, one challenge of existing approaches is generalizing to test data that are significantly different from the training set.

1.3 OBJECTIVES

The objective is to develop a solution for the execution of a precise bin picking vision process while taking into account the challenges associated with this process.

In this project, I present a robotic vision based bin picking system for industrial robotics applications. The developed robotic vision system performs detection and recognition functions of the target object and provides the information for sequential executions of the pick-and-place operations to the robot control system. A synthetically trained network generalizes well to a variety of real world scenarios, including various backgrounds and extreme lighting conditions. The steps to follow are thus as follows:

- Using DOPE (for "deep object pose estimation") system which works on a one-time, deep neural network-based system that infers, in near real time, the 3D poses of known objects in clutter from a single RGB image without requiring post-alignment can be achieved. This system uses a simple deep network architecture, trained entirely on simulated data, to infer the 2D image coordinates of projected 3D bounding boxes, followed by perspective-n-point (PnP) [13].
- To execute this process with high precision, the Demonstration that combining both non-photorealistic (domain randomized) and photorealistic synthetic data for training robust deep neural networks successfully bridges the reality gap for real-world applications, achieving performance comparable with state of the art networks trained on real data will be the strategy to approach.

Thus an integrated robotic system that shows the estimated poses are of sufficient accuracy to solve real world tasks such as pick and place, object handoff, and path following can be developed. This system to be developed may be placed at the end of a manipulator and become an active perception unit for the detection of objects with high accuracy for further manipulation. Therefore, the first major objective will be the installation of the Realsense sensor in the Aubo i5 manipulator and its connection to the acquisition and control system. Later, it will be necessary to integrate the global system of sensors, their controllers and the robotic manipulator into a ROS (Robot Operating System) infrastructure. Finally, an application demonstrating the accuracy of the bin picking process will be developed.

1.4 DOCUMENT STRUCTURE

This dissertation is composed of 9 chapters, including the present one, in order to meet the objectives mentioned above in a more organized way. Those chapters are arranged as follows:

1. Introduction - The current chapter in which is presented a brief framework, a description of the problem per se and an enumeration of the objectives.
2. State of the Art - Describes developed projects in the field of bin-picking and some solutions already commercialized in industrial applications;
3. Experimental Infrastructure - Introduces and describes all the hardware and software used to develop this bin-picking process;
4. Calibrations - Describes the calibration techniques and procedures performed to incorporate all the hardware and respective coordinate frames in an overall system;
5. Data Generation and Training – Describes the steps in creating dataset for training.
6. Detection and Pose Estimation - Presents a detailed explanation of the Realsense data processing;
7. System Integration - Describes the steps required for the development and organization of the entire bin-picking vision system in ROS Environment.
8. Evaluations and Results - Describes all the experiments performed to demonstrate the versatility of the developed bin-picking vision system.
9. Conclusions and Future Works - Presents conclusions of the developed project and some suggestions for future work in this field of study.

Chapter 2

STATE OF THE ART

The creation of a bin-picking vision system, especially of the random type, has been one of the focus of several companies in the robotics industry and research organizations. There have been tremendous strides empowered by advanced vision technology, software, and gripping solutions, which are in constant development. This chapter describes some recently developed projects in this field and some already commercialized solutions in industrial applications. The purpose of this chapter is not to present all types of bin-picking systems already developed, but rather explain ones that focus more in deep learning based bin-picking systems to perform a precise bin-picking process.

2.1 INDUSTRIAL IMPLEMENTATIONS OF BIN PICKING VISION SYSTEMS

The number of bin-picking systems already implemented in the industry is very large. The ones presented next are here for presenting some of the solutions with enough reliability to integrate an industrial process and to demonstrate the type of robust systems already in use in the market.

2.1.1 Solomon AccuPick 3D and SV-Series using Intel Realsense

AccuPick 3D has integrated its versatile bin picking software with RealSense technology from Intel, the best in class 3D camera using active Stereo Vision (SV). AccuPick's motion planning module is specifically targeted for applications requiring robot arms to maneuver inside of an outsized bin in industries ranging from automotive components, electronics, food packaging, machine tending, and warehouse de-palletization. The system is able to work in complex environment while remaining compatible with major robot brands, such as Universal Robots (UR+ certified), Fanuc, Staubli, Yaskawa, Kuka, ABB and more. Advanced tasks such as sealing, assembly, welding or inspection can also be equally difficult or time consuming for robots. Solomon's 3D vision guided robot solution, Solmotion, takes complexity out of these seemingly challenging tasks as it recognizes unique features to determine the three-dimensional position and orientation of a work piece with precision and speed.

All it takes is for the Solscan 3D scanner to take a snapshot of the work piece, before the software immediately matches and calculates the required routes for robot to move along. Applications include sealing of automotive parts, gluing of footwear parts, quality

inspection of metal or plastic injection objects, welding of motors, to name a few. In industrial setting, defects and features with irregular patterns such as hard to define scratches, stains, cracks and many other types of flaws, Solvision.

Figure 2.1 Solomon AccuPick 3D Bin Picking System

Solomon's AI-based machine vision software can further solve inspection problems. The software deep learning requires no tedious writing of codes to inspect various types of defects. It takes a few samples to label the defect types for machines to learn, simplifying inspections tasks and saving significant amount of engineers' time to write the programs. Another advantage of Solvision is its seamless integration with different robots, six axis or SCARA, Solomon's 3D Solscan scanner and software, so inspected objects can be automatically separated accordingly.

2.1.2 FIZYR AI software for vision guided robotics

Fizyr's vision software product is sensor independent. The client can configure the capabilities and limitations of any gripper in our software. They support multi suction grippers to apply the right (combination of) suction cups based on size, shape and material.

To make use of the 6-DoF grasp pose developed by a gripper with two extra degrees of freedom. This gripper can handle many situations making it easy to integrate with any robot in a picking cell.

Trained neural networks will provide the best 6-DoF grasp locations per object and relevant additional classifiers. They built a unique set of images representing item picking, parcel handling, de-palletizing and truck unloading in real logistics environments. One advantage of the autonomous piece picking robots from Fizyr is that it is integrated with a conveyor for transportation purposes. Fizyr software algorithm classifies boxes, plastic bags, and all kinds of items and defines the best grasp strategy for over 100 parcels per second.

Figure 2.2 Fizyr AI 3D Software

2.1.3 InPicker 3D System

InPicker allows identifying and determining the location of objects piled in a container. After selecting the best candidate, the system determines the position of the object in a 3D space, guiding the robot gripper with the most precise orientation, in order to extract the selected element. In the process of moving the robot arm, the system takes into account the position of the container and the stacked parts to avoid any possible collision. InPicker consists of a hardware solution (vision + robot) and software (image analysis + communications) that allows extracting randomly sorted parts from containers.

InPicker solves the Bin Picking application in 6 steps:

- Image capture: A 2D or 3D camera is used to obtain an image of the parts and the work environment.
- Scene 3D map creation: Interprets the environment in 3D to identify the objects of interest and determine any obstacles.
- Part detection and location: Parts are detected in the scene by determining their 3D orientation position and their different rotations.
- Grasping position: All possible perspectives in which the robot can grasp the detected parts are calculated.
- Collision prevention: Grasping positions that may pose a risk of collision in the robot's path are screened out and discarded.
- Sending coordinates: The most appropriate one is selected so that it can be extracted accurately and safely by the robot and then placed in the next production process.

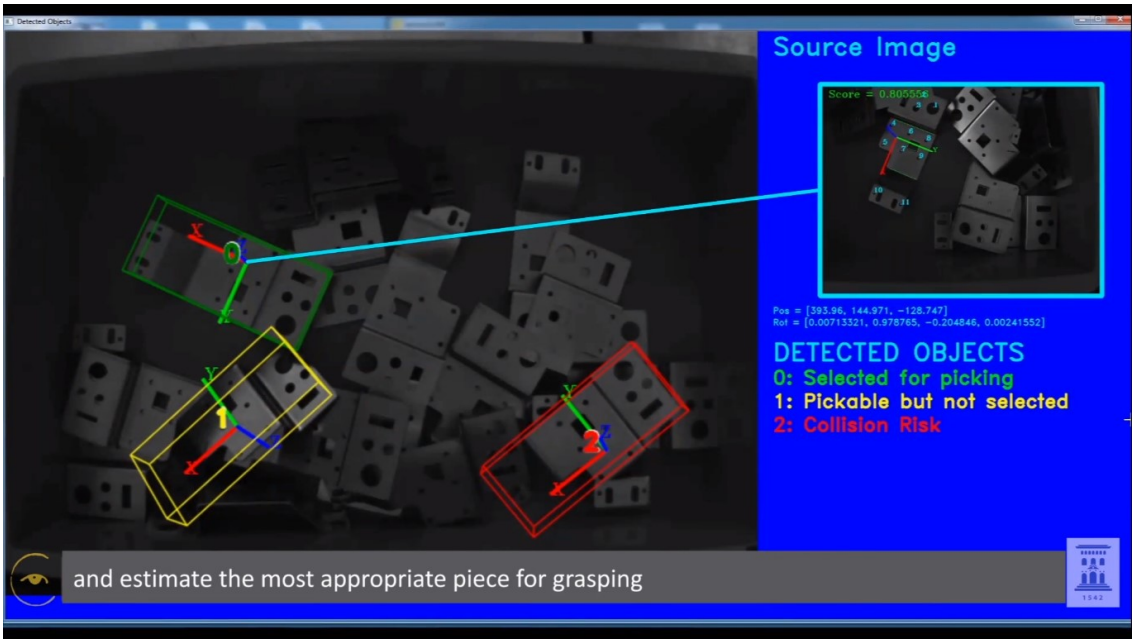


Figure 2.3: InPicker 3D System

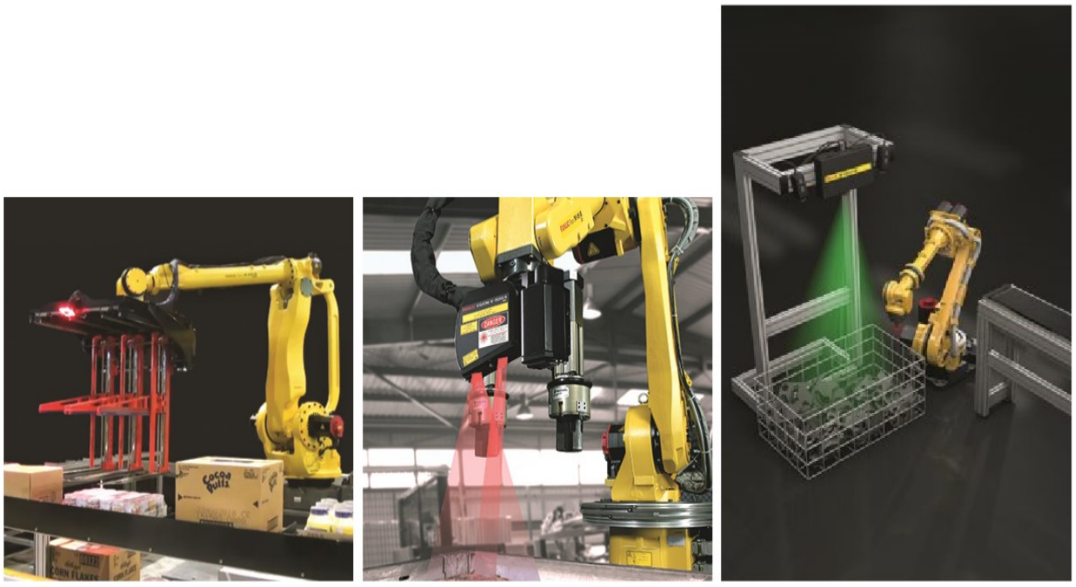
2.1.4 FANUC and its visual detection system iRVision

FANUC developed a plug and play visual detection system called iRVision. According to FANUC, this type of system is capable of locating workpieces whatever their size, shape or position by using either 2D or 3D part recognition. It can also read 1D and 2D bar codes, sort according to colour and support flexible parts feeding, high-speed visual line tracking and bin or panel picking and placing. With this system is possible to increase the overall

production flexibility and efficiency in the workplace. The iRVision can be fully integrated into an entire range of FANUC robots without complicated programming or expert knowledge [14]. This solution is suitable to various applications and industries such as Automotive, Food, Metal, Plastic, Aeromotive and Pharma industries.

FANUC iRVision supports several vision types such as 2D, 2.5D, 3D, 3D Laser and 3D Area Sensor. The 2D vision is capable of detecting objects positioned in one layer (X,Y,R) and 2.5D in two or more layers (X,Y,Z,R), both capable of picking up non-moving parts. The 2.5D vision system is represented in figure 2.4a. The 3D laser vision detects objects in position and posture by laser projection (X,Y,Z,W,P,R) and is presented in figure 2.4b .

Lastly, figure 2.4c presents the most robust system which incorporates a 3D area sensor for detection of objects by 3D map (structured light projection) (X,Y,Z,W,P,R). This last one can especially be used for high-end vision based bin picking and material handling applications despite the parts conditions, like being dirty or rusty [15].



(a) 2.5D vision system. (b) 3D laser vision system. (c) System with a 3D Area Sensor.

Figure 2.4: Different types of vision systems supported by FANUC iRVision

In order to facilitate the selection and purchase of the users' desired vision system among the ones available, FANUC offers to test the vision process with the user environment. The FANUC's simulation software ROBOGUIDE enables an evaluation of the time, effort and feasibility of the entire process, allowing the user to select and modify parts and dimensions as required, before making a purchase decision [15].

2.1.5 Pick-it

Pick-it is also a plug and play smart automation product at a fixed price. This robot vision system guides the client's robot to pick and place a wide range of products in different applications and scenarios. This system can be incorporated in different manipulators (FANUC, ABB, and KUKA). According to Pick-it, this system is capable of identifying a wide variety of products. This can be done by using their FLEX detection engine, which is capable of finding geometric shapes such as cylinders, boxes, planes and circles, as represented in figure 2.5, in which the operator chooses a shape and sets a range of sizes that need to be picked. Another solution is by using the TEACH detection engine, also provided, which can handle more complex 3D shapes after the operator takes a 3D picture of the product with the Pick-it camera so that the system can look for this shape [16]. Both vision engines offer the same tools to select parts from random bins by finding their position and orientation.

The Pick-it product includes a 3D camera, the Pick-it software and an industrial processor. The 3D camera, presented in figure 2.5a, uses structured light, which projects a known pattern onto a scene, to compute the 3D image. This way it is capable of finding overlapping products of varying sizes and materials, even in changing and poor light conditions. It also contains an RGB camera, which shows the current workspace for a better visualization of the process, but it is not used for perception. The technical data sheet of the camera can be found in [16]. The Pick-it software, figure 2.5b, has an easy to configure interface and there is no programming needed. The configuration of this type of application is done through a web interface in a Google Chrome web browser. One of the main advantages of the Pick-it solution is the capability of integrating their product in a ROS interface. ROS is explained in the Software section of chapter 3. The Pick-it software comes pre-installed on an industrial processor with 12 cores at 3.5GHz, presented in figure 2.5c.

(a) 3D Camera. (b) Pick-it Software. (c) Industrial Processor.

Figure 2.5: Pick-it product components

2.2 RELATED ACADEMIC WORK

Conversely, research organizations have been investing in cheaper alternatives to replicate this kind of robust and precise bin-picking processes. Several entities made it possible by using off-the-shelf and open source hardware and software, easy to use and integrate into an overall system. Being bin picking a process with so many variants to improve in order to create a sustainable solution, researchers have been tackling some of the main ones. Those include precise estimation of an object's position and pose, object recognition, grasp planning, among others, all in a bin/pallet full of randomly stacked objects. There are many papers and other academic work that focuses on refining each of these variants in particular. Since this project will focus on developing a precise bin-picking vision system capable of correctly estimating pose of objects by deep learning, the following applications present related solutions.

2.2.1 Robotic vision system for random bin picking with dual-arm robots - Electronics and Telecommunications Research Institute, Daejeon, Korea - 2016

Random bin picking is one of the most challenging industrial robotics applications available. It constitutes a complicated interaction between the vision system, robot, and control system. For a packaging operation requiring a pick and place task, the robot system utilized should be able to perform certain functions for recognizing the applicable target object from randomized objects in a bin. In this paper, they introduce a robotic vision system for bin picking using industrial dual-arm robots. The proposed system recognizes the best object from randomized target candidates based on stereo vision, and estimates the position and orientation of the object. It then sends the result to the robot control system. The system was developed for use in the packaging process of cell phone accessories using dual arm robots.

2.2.2 NimbRo Picking: Versatile Part Handling for Warehouse Automation - IEEE International Conference on Robotics and Automation (ICRA) - 2017

Part handling in warehouse automation is challenging if a large variety of items must be accommodated and items are stored in unordered piles. To foster research in this domain, Amazon holds picking challenges. They present their system, which achieved second and third place in the Amazon Picking Challenge 2016 tasks. The challenge required participants to pick a list of items from a shelf or to stow items into the shelf. Using two deep-learning approaches for object detection and semantic segmentation and one item

model registration method, their system localizes the requested item. Manipulation occurs using suction on points determined heuristically or from 6D item model registration. Parametrized motion primitives are chained to generate motions. They present a full-system evaluation during the APC 2016 and component level evaluations of the perception system on an annotated dataset.

2.2.2 Synthetically Trained Neural Networks for Learning Human-Readable Plans from Real-World Demonstrations - University of Southern California - 2018

Present a system to infer and execute a human readable program from a real world demonstration. The system consists of a series of neural networks to perform perception, program generation, and program execution. Leveraging convolutional pose machines, the perception network reliably detects the bounding cuboids of objects in real images even when severely occluded, after training only on synthetic images using domain randomization. To increase the applicability of the perception network to new scenarios, the network is formulated to predict in image space rather than in world space. Additional networks detect relationships between objects, generate plans, and determine actions to reproduce a real world demonstration. The networks are trained entirely in simulation, and the system is tested in the real world on the pick and place problem of stacking colored cubes using a Baxter robot.

2.2.3 PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes - NVIDIA Research - 2018

Estimating the 6D pose of known objects is important for robots to interact with the real world. The problem is challenging due to the variety of objects as well as the complexity of a scene caused by clutter and occlusions between objects. In this work, they introduce PoseCNN, a new Convolutional Neural Network for 6D object pose estimation. PoseCNN estimates the 3D translation of an object by localizing its center in the image and predicting its distance from the camera. The 3D rotation of the object is estimated by regressing to a quaternion representation. They also introduce a novel loss function that enables PoseCNN to handle symmetric objects. In addition, they contribute a large-scale video dataset for 6D object pose estimation named the YCB-Video dataset. Their dataset provides accurate 6D poses of 21 objects from the YCB dataset observed in 92 videos with 133,827 frames. They conduct extensive experiments on our YCBVideo dataset and the Occluded LINEMOD dataset to show that PoseCNN is highly robust to occlusions, can handle symmetric objects, and provide accurate pose estimation using only color images as input.

2.2.5 Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects - 2nd Conference on Robot Learning, Z¨urich, Switzerland - 2018

Using synthetic data for training deep neural networks for robotic manipulation holds the promise of an almost unlimited amount of pre-labeled training data, generated safely out of harm's way. One of the key challenges of synthetic data, to date, has been to bridge the so-called reality gap, so that networks trained on synthetic data operate correctly when exposed to real-world data. They explore the reality gap in the context of 6-DoF pose estimation of known objects from a single RGB image. They show that for this problem the reality gap can be successfully spanned by a simple combination of domain randomized and photorealistic data. Using synthetic data generated in this manner, they introduce a one-time deep neural network that is able to perform competitively against a state-of-the-art network trained on a combination of real and synthetic data. This is the first deep network trained only on synthetic data that is able to achieve state-of-the art performance on 6-DoF object pose estimation. Their network also generalizes better to novel environments including extreme lighting conditions, for which we show qualitative results. Using this network, they demonstrate a real-time system-estimating object poses with sufficient accuracy for real-world semantic grasping of known household objects in clutter by a real robot.

Chapter 3

EXPERIMENTAL INFRASTRUCTURE

In order to develop this project some devices such as a computer, a robotic handler and sensors are required. In addition to these, to develop and execute this application, the installation of software is crucial.

In this chapter, all the hardware and software used will be introduced and fully described.

3.1 HARDWARE

3.1.1 AUBO i5 Cobot

Cobot or collaborative robot arm 5 Kg payload (AUBO-i5) was specially designed with important functions from the start, combining state of the art technology with user-friendliness and high repeatability to make this collaborative robot (Cobot) user-friendly. The open-source architecture called open unit robot enables the robot operating system (ROS) to be supported through API for both industrial and academic use. AUBO robot uses the CAN bus network to communicate between harmonic drives this offers un-parallel versatility with six DOF. Low cost of ownership and high positional repeatability are some of the other criteria that make up the outstanding features of this robot. Aubo Robotics model AUBO-i5 can bear 5Kg payload collaborative robot (Cobot) with 6-DoF and 922 mm Reach. AUBO-5 works closely within environment human without safety equipment, depending on risk assessment.

Collaborative Function:

- Guide to teach (inverse kinematics motion planning), this manual operation of the robot enables you to quickly and easily program the robot by demonstration without any programming skills.
- Works side by side with human operator safety fence, laser or sensors (after a risk assessment is performed).
- Teach pendant user interface for programming (forward kinematics) enables online programming and simulation via a touch screen tablet.
- Lightweight, flexible, easy to re-purpose this robot weighs in under 24Kg.

Open Source Architecture:

- CAN bus network used in this robot for multiple microcontrollers to communicate with each other.

- ROS (Robot Operating System) is supported through API.
- Hardware adopts CAN bus protocols with open I/O interface extensions.
- Easily integrate the robot into existing production systems.

Intelligence:

- Vision systems can be integrated into the controller.
- Software system based on cloud platform management that realizes remote maintenance, fault diagnosis and online upgrading of firmware.
- This research robot platform is used widely around the world incorporate research labs and for academic robotics research.

Figure 3.1: Aubo i5 Cobot

Moreover, AUBO Robotics AUBO i5 is a collaborative industrial robot (Cobot) operating with certified safety features, Hand Drive operation, and the power and force limiting robot design type.

3.1.2 Intel RealSense Depth Camera D435i

Intel Realsense Depth Camera D435i combines the depth-sensing capabilities of the D435 with the addition of an inertial measurement unit (IMU). For great scans, an IMU provides an extra set of data allowing for better dense reconstruction. Alternately, in robotics, the Robotic Operating System (ROS) takes this input for your robot so you can understand its position in the world.

The inertial measurement unit (IMU) is used for the detection of movements and rotations in 6 degrees of freedom (6DoF). An IMU combines a variety of sensors with gyroscopes to detect both rotation and movement in 3 axes, as well as pitch, yaw, and roll. It is used in applications such as gaming and pointing devices as well as image stabilisation. Adding an IMU allows your application to refine its depth awareness in any situation where the camera moves. This opens the door for rudimentary SLAM and tracking applications allowing better point cloud alignment. It also allows improved environmental awareness for robotics and drones. The use of an IMU makes registration and calibration easier for handheld scanning system use cases, is also important in fields such as virtual/augmented reality, and drones. When using the D435i, our Intel RealSense SDK 2.0 provides IMU data that is time-stamped to align with our high-quality depth data. The Intel Realsense D435i places an IMU into our cutting-edge stereo depth camera. With an Intel module and vision processor in a small form factor, the D435i is a powerful complete package that can be paired with customisable software for a depth camera that is capable of understanding its own movement.

Figure 3.2: Intel Realsense D435i

The characteristics of the Realsense are listed below:

- Image Sensor Technology: Global Shutter, 3µm × 3µm pixel size
- Maximum Range: Approx. 10 meters. Accuracy varies depending on calibration, scene, and lighting condition.
- Minimum Depth Distance (Min-Z): 0.105 m
- Depth Field of View (FOV):86° × 57° (±3°), Depth Output Resolution: Up to 1280 × 720, Depth Frame Rate: Up to 90 fps
- RGB Sensor Resolution: 1920 × 1080, RGB Frame Rate: 30 fps

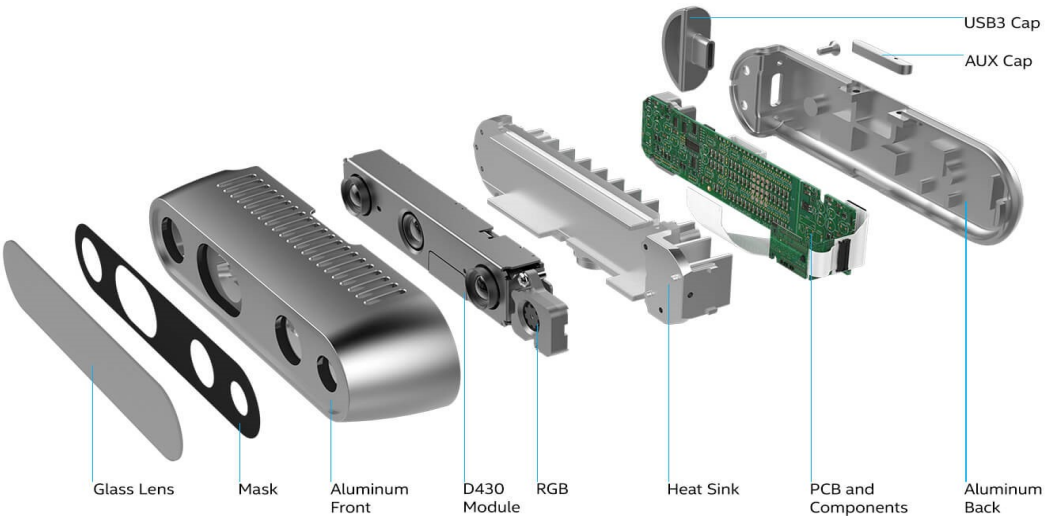


Figure 3.3: Realsense Sensor Components

Operation Mode:

The combination of a wide field of view and a global shutter sensor on the Intel D435i make it the preferred solution for applications such as robotic navigation and object recognition. The wider field of view allows a single camera to cover more areas resulting in fewer “blind spots”. The global shutter sensors provide great low-light sensitivity allowing robots to navigate spaces with the lights off. The depth camera D435i is part of the Intel Realsense D400 series of cameras, a lineup that takes Intel’s latest depth-sensing hardware and software and packages them into easy-to-integrate products. Perfect for developers, makers, and innovators looking to bring depth sensing to devices, Intel Realsense D400 series cameras offer simple out-of-the-box integration and enable a whole new generation of intelligent vision-equipped solutions.

A common representation of below type of data is a point cloud. Point clouds are a set of data points in a three dimensional space, in which each point can have additional features,

such as an associated color. Since the Realsense uses IR, it will not work in direct sunlight, for example in the outdoors.

Figure 3.4: The laser grid used by the Realsense to calculate depth

Support for the installation in the Robot

Since one of the objectives of this project is to create an active vision system unit by placing the sensor on the manipulator, a support had to be designed to attach the Realsense to the robot. By doing so, the industrial robot can be placed in different locations having always the vision system incorporated and ready to act in any occasion.

After analyzing all the possible positions, the most adequate was selected. The one in which the Realsense has 6 degrees of freedom and is attached to the side of the robot's end effector was chosen as the most suitable. This option is represented in figure 3.5 and was the one chosen for the combination of the following reasons. By having the Realsense in this location, the robot does not need to move to a position closer to the workspace limits to enable the camera to take a frame that contains the whole workspace. Besides, in this position the dimensions of the Realsense will not interfere with the movements of the manipulator.

In order to attach the Realsense to the robot's end effector, a structure, which could hold the device and be screwed to the steel sheet that protects the manipulator, was modeled. This support is represented in the figure 3.5 below.

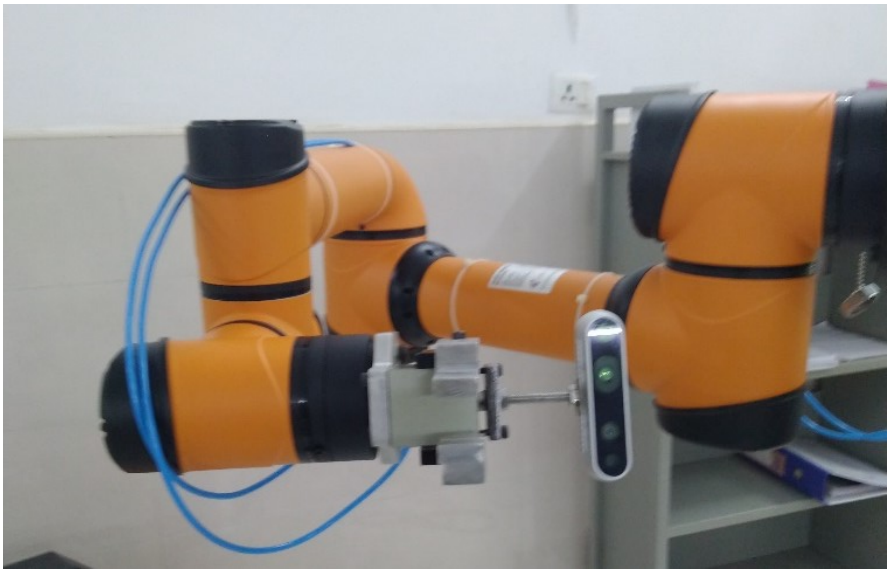


Figure 3.5: Realsense and Support fixed to the Robot

Now, with the Realsense properly installed, it is fundamental to calibrate its intrinsic and extrinsic parameters, as described in Chapter 4.

3.2 SOFTWARE

3.2.1 ROS - Robot Operating System

Creating truly robust software for robotic systems is hard. To do so, there is a need to combine all the different equipments and the communications with each other, often with programs written in different languages. In order to simplify this type of software architecture the ROS framework was created. The Robot Operating System (ROS) is a flexible framework for robot software development. It combines all the tools, libraries, and conventions with the aim of simplifying the task of creating complex and robust robot behavior across a wide variety of robotic platforms [17]. ROS contains different distributions, which are a versioned set of ROS packages. The one used in this project is the ROS Kinetic Kame.

Figure 3.6: ROS Kinetic Kame logo

Using ROS has many advantages. The primary goal of ROS is to support code reused in robotics research and development with the purpose of being possible to find a built-in package system. ROS also gives the flexibility to write nodes in different languages for the same system. It is possible to write code in Python, C++ and Lisp.

At the lowest level, ROS offers a message passing interface that provides inter-process communication. However, the basic principle of a robot operating system is to run a great number of executables and enable them to exchange data synchronously or asynchronously. The instance of this executables are, in the ROS language, called nodes. ROS is a combination of independent nodes, each of which are responsible for one task and communicate with the other nodes using a publish/subscribe messaging model via logical channels called topics.

Messages are structures of data filled with pieces of information by nodes. They are a combination of primitive types such as strings, Booleans, integers and floating point and messages that are a recursive structure. Data is exchanged by two means, topics and services. A topic is a data transport system based on a subscribe/publish system which enables an asynchronous communication. One or more nodes are able to publish data to a topic and read data on that topic. A Service in contrast to a topic enables a synchronous communication between two nodes. The overall process is represented in figure 3.7.

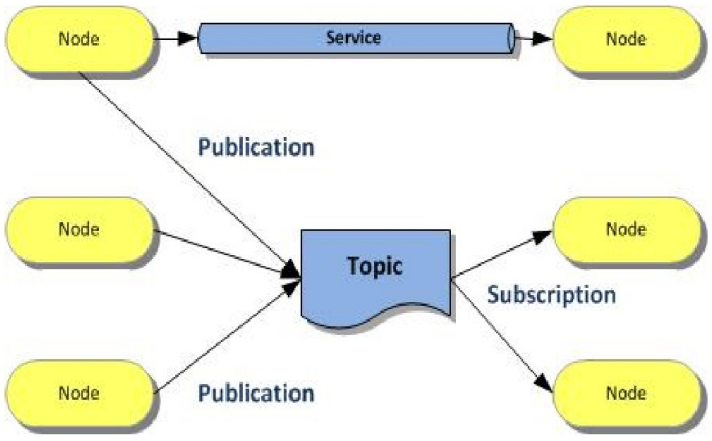


Figure 3.7: ROS basic concepts [18]

ROS also provides other facilities such as recording and playing back messages. Bags are formats for storing and playing back those messages. In order to manage this loosely coupled environment, there is a Master in ROS, which is responsible for name registration and lookup for the rest of the system. The Master is a node declaration and registration service, which enables nodes to find, each other and exchange data [18].

RViz

Rviz stands for ROS visualization and, as the name says, it is a 3D visualization tool for displaying sensor data and stating information from ROS. Using rviz, it is possible to visualize a manipulator's current configuration through its URDF described model. The Unified Robot Description Format (URDF) is an XML format for representing robot models, sensors, scenes, etc [19]. It is also possible to display a live representation of three-dimensional point clouds and sensor values coming over ROS Topics, such as sonar data, camera data and infrared distance measurement [20]. This tool is capable of using the information from the *tf* library to show all of the sensor data in a desired common coordinate frame, together with a 3D rendering of the robot. Visualizing all of this data in the same application allows the user to quickly see what the robot sees, and identify problems such as sensor misalignments or robot model inaccuracies [21].

Rqt

Rqt is a software framework for developing graphical interfaces for a robot by implementing various GUI tools in the form of plugins. With *rqt* it is possible to run all of the existing GUI tools in multiple widgets in a single window at one moment or just run it in a traditional standalone method. Users can also introduce new interface components by creating their own rqt plugins with either C++ or Python [21] [22]. However, *rqt* already has many useful plugins such as, *rqt_graph* for visualization of a live ROS system, showing nodes and the connections between them and *rqt_plot* to monitor encoders, voltages, or anything that can be represented as a number that varies over time.

tf

tf is a package, widely used in this project, that enables the tracking of multiple coordinate frames over time and is better described in chapter 4.

3.2.2 ROS Industrial

ROS-Industrial is an open-source project that extends the sophisticated capabilities of ROS software by bringing advanced robotics software to the industrial automation domain. This ROS extension contains libraries, tools, drivers and virtual models (URDF) for industrial hardware [23]. The instruction for the installation of ROS-Industrial can be found in [24].

The ROS-Industrial distribution contains metapackages for several industrial vendors, like ABB, Adept, Fanuc, Motoman, and Universal Robots. In order to manipulate AUBO i5 the ROS-Industrial relies, then, on the ROS Aubo metapackage. Additionally, the ROS

MoveIt metapackage and the installation of the drivers on the controller are also required for the correct manipulation of the robot used in this project.

3.2.3 Move*It*

MoveIt! is an open source software which includes the tools and packages required for mobile manipulation. This package incorporates dynamic motion planning, manipulation, 3D perception, kinematics, collision checking, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications [25]. The *MoveIt*! Package can be used through an easy to use interface in RViz, figure 3.8 or through programming. However, by using this package through programming there will also be an interface that displays the motion plan computed, due to the associated topics published.

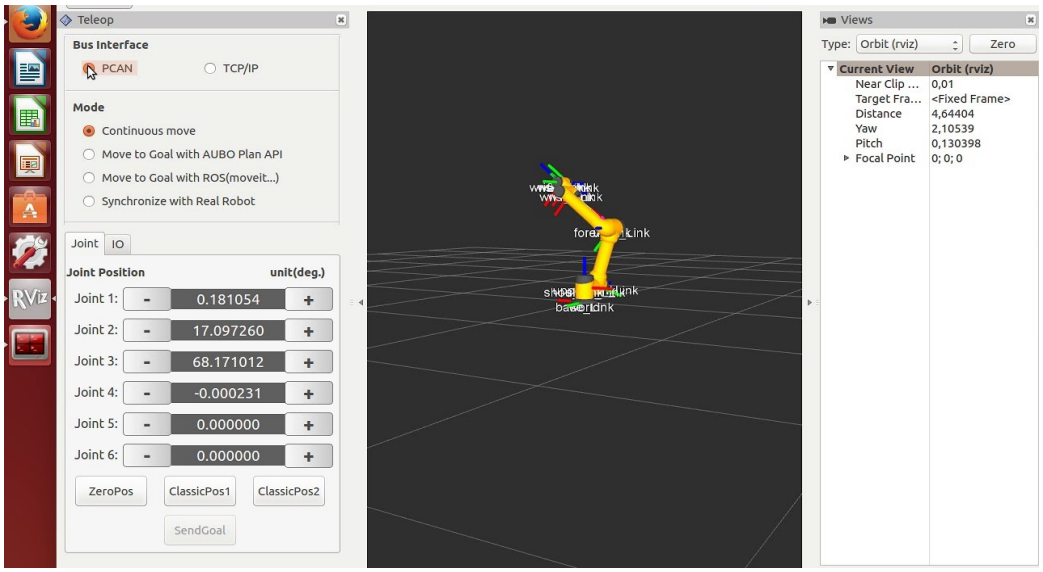


Figure 3.8: RViz interface of the MoveIt package

To use this package to manipulate a specific robot it is necessary to define some parameters, such as the type of joints and their names, their limits and the maximum velocities. To meet this need is then used the ROS Aubo Meta package.

3.2.4 ROS *aubo*

ROS Aubo is the metapackage that supports the integration of AUBO manipulators in a ROS platform. It currently contains packages that provide nodes for communication with AUBO industrial robot controllers and URDF models for various AUBO manipulators. The packages used in this project were the Aubo driver and the 3 packages that focus in the Aubo i5 manipulator. In fact, the last 3 packages are still integrated in the *aubo_experimental* metapackage which contains experimental packages that will be moved to the Aubo metapackage once they've received sufficient testing and review.

The *aubo_driver* package contains all the drivers to install in any AUBO controller for its interfacing with the ROS Industrial program. This package contains different files designed to be compiled, through a Gazebo simulator, in order to generate executable files that will be then send to an AUBO controller.

The 3 packages that focus in the AUBO i5 are a support package, a *moveit_plugins* package and a *moveit_config* package. These include all the configuration files with information about the hardware, algorithms for computation of the robot's kinematics, STL models of all the robot's components for the recreation of the URDF model, etc. Since the robot's URDF model used for this project will also include the frames and the STL models of the Realsense sensor it is necessary to create a *moveit_config* package for this specific model, thus providing collision-aware path planning for the used robot. This can be done by following the tutorial in [26].

This tutorial walks through the steps of filling the *MoveIt* Setup Assistant wizard, explains how to update various configuration/launch files to provide additional robot-specific data that allows *MoveIt* to communicate with the robot interface node and explain the process for connecting with a real robot. The second step of this tutorial also explains how to create the planning and execution launch file which will be the one used to invoke the ROS Industrial tools, to launch the planning environment and all the communications to control the manipulator. Nodes trigger this communication and topics, which are responsible for keeping track of the robot state, communicate with the controller, establishing the workspace, etc.

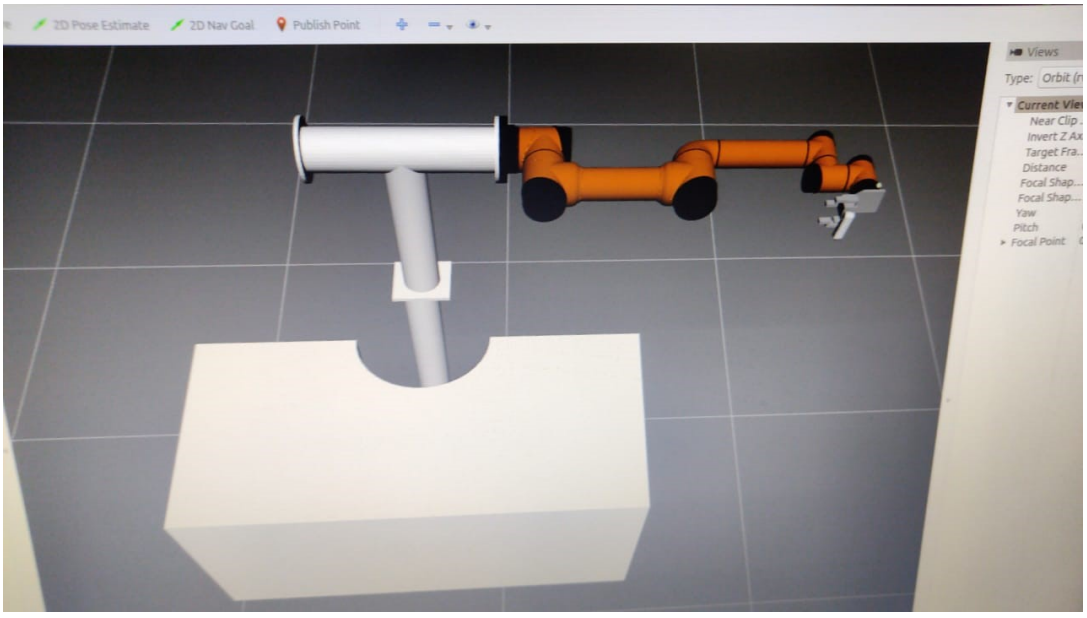


Figure 3.9: RViz interface of the ROS aubo i5

3.2.5 NVIDIA Deep learning Dataset Synthesizer (NDDS) and UE4

NDDS is a UE4 plugin from NVIDIA to empower computer vision researchers to export high-quality synthetic images with metadata. NDDS supports images, segmentation, depth, object pose, bounding box, keypoints, and custom stencils. In addition to the exporter, the plugin includes different components for generating highly randomized images. This randomization includes lighting, objects, camera position, poses, textures, and distractors, as well as camera path following, and so forth. Together, these components allow researchers to easily create randomized scenes for training deep neural networks.

Training and testing deep learning systems is an expensive and involved task due to the need for hand-labeled data. This is problematic when the task demands expert knowledge or not-so-obvious annotations (e.g., 3D bounding box vertices). In order to overcome these limitations we have been exploring the use of simulators for generating labeled data. We have shown in [7] [8] that highly randomized synthetic data can be used to train computer vision systems for real-world applications, thus showing successful domain transfer.

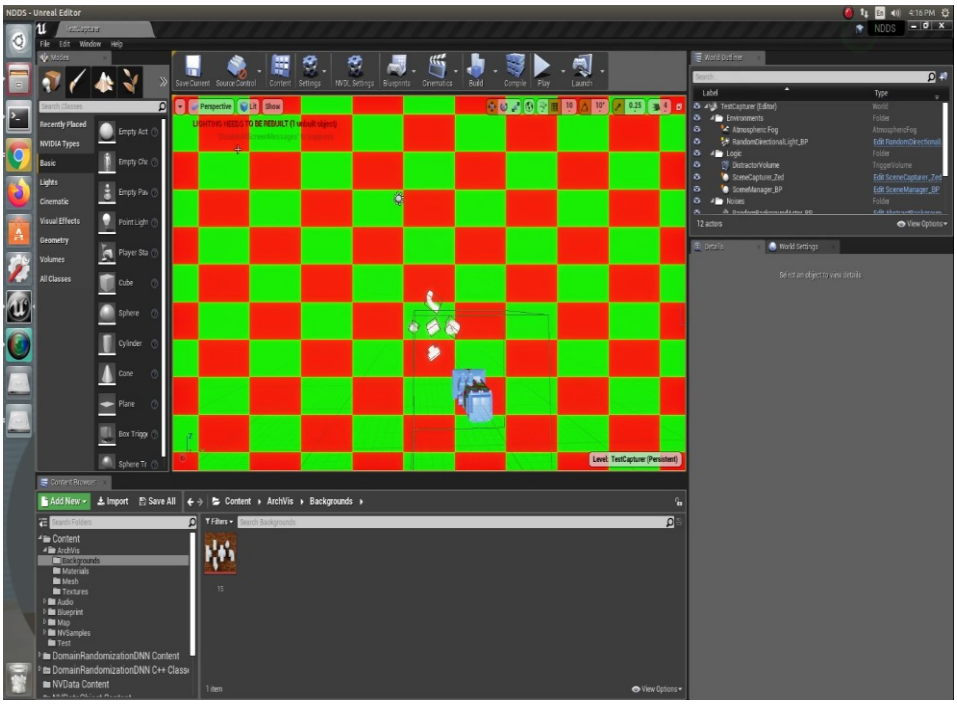


Figure 3.10: NDDS interface of the UE4 [28]

Unreal Engine 4 (UE4)

Unreal Engine is the world's most open and advanced real-time 3D creation tool. Continuously evolving to serve not only its original purpose as a state-of-the-art game engine, today it gives creators across industries the freedom and control to deliver cutting-edge content, interactive experiences, and immersive virtual worlds.

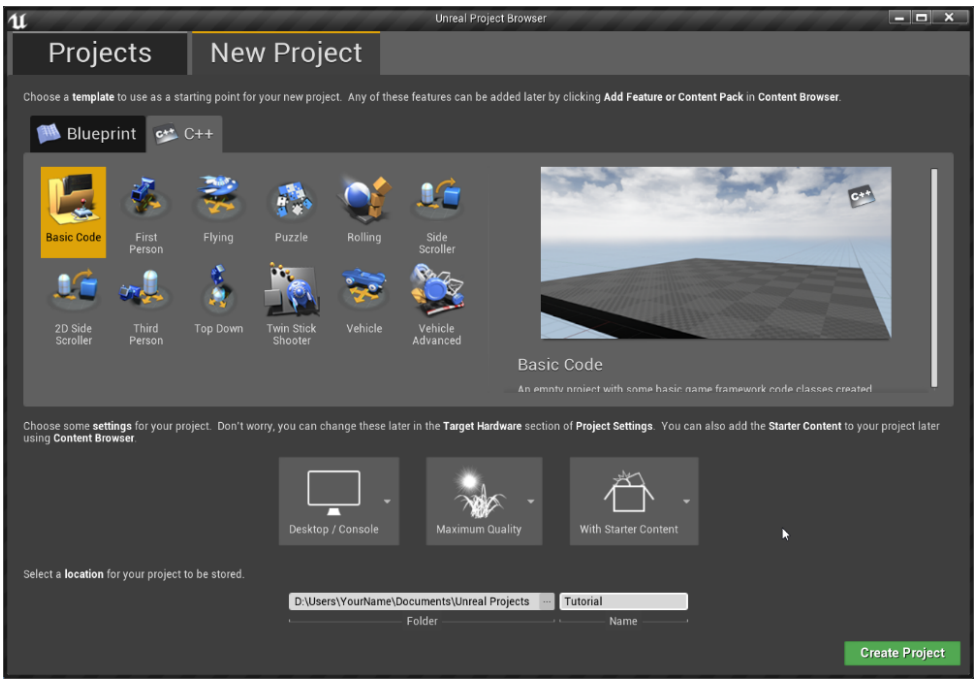


Figure 3.11: Interface of the UE4 Editor

Unreal Engine includes the Unreal Editor, an integrated development environment available on Linux, MacOS, and Windows for content authoring. With support for multi-user editing, artists, designers, and developers can simultaneously make changes to the same Unreal Engine project in a safe and reliable way.

Chapter 4

CALIBRATIONS

This chapter describes the calibration techniques and procedures performed between each pair coordinate frames of the involved system.

Until now, the hardware of this project was all used separately. In order to integrate the robotic manipulator, the sensors and their controllers into a global system, it is important to determine the transformations between all the components. The robot, the Realsense sensor and the gripper all contain 3D coordinate frames that change over time. To keep track of all these frames the *tf* package, from ROS, was used. *tf* maintains the relationship between coordinate frames in a tree structure buffered in time and lets the user transform points and vectors between any two coordinate frames at any desired point in time [27].

4.1 AUBO ROBOT AND END EFFECTOR

In order to include the coordinate frames of all the robot's joints, the URDF specification of the AUBO-i5 was included to launch the global system. The specifications of the AUBO manipulators can be downloaded from the git repository in [28].

The frame which represents the base of the robot (*robot_base_link*) was defined as the global fixed frame. Obviously, the robot's specification does not incorporate the description of the gripper used in this project. To do so, the coordinate frame of the tool center point (TCP) has to be incorporated in the global system. The TCP constitutes the origin of the tool coordinate system and is the one which will be moved to the programmed target position.

In order to incorporate the frame of the tooltip (*eef_tool_tip*), the gripper's exact length had to be determined. This was done by moving the end effector closer to the table and by visualizing the coordinates of the frames sent from the robot through a TCP/IP connection. By doing so, it was possible to determine the difference on the OZ axis between the base of the robot (*robot_base_link*) and the last frame of the robot (*eef_base_link*). This difference was later used to do the transformation to represent the frame of the tool tip (*eef_tool_tip*). This process is represented in figure 4.1, it is possible to see that after the transformation, the *eef_tool_tip* is almost equal to zero on the Z direction, as desired.

Figure 4.1: Visualization of the robot with the end eff ector and is coordinate frame

The frame of tip of the gripper was represent the real one, by incorporating the mesh of the gripper and the description of the last coordinate system (*eef_tool_tip*) in the URDF specifications. The visualization of the robot with the end effector, in the respective position, and all coordinate frames, so far, is now possible using the following command:

4.2 REALSENSE

The calibration of a RGB-D sensor consists of two parts, the calibration of the intrinsic and extrinsic parameters. The intrinsic parameters represent a projective transformation from the 3D camera's coordinates into the 2D image coordinates. The extrinsic parameters represent a rigid transformation from 3D world coordinate system to the 3D camera ĂŹs coordinate system. The camera calibration parameters are represented in figure 4.2 for a better understanding.

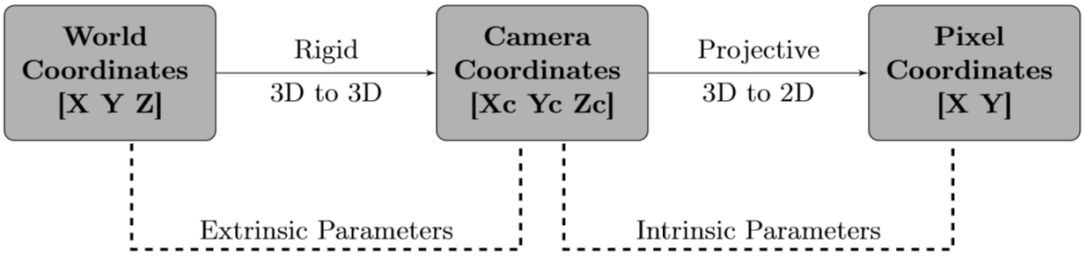


Figure 4.2: Camera calibration parameters

4.2.1 Calibration of the Intrinsic Parameters

The intrinsic calibration is responsible for describing the internal parameters of both RGB and IR (depth) cameras. Those parameters are the focal length, the optical center, also known as the principal point, the skew coefficient and the coefficients that describe the lens distortion [29] [30].

Actually, this process is not truly required, since the *realsense_viewer* driver already provides default camera models with reasonably accurate focal lengths and other parameters definitions. Additionally, even though the driver does not model the lens distortion, the *realsense*, fortunately, uses low-distortion lenses, so even the edges of the image will not be displaced by more than a few pixels. However, if the application in which the *realsense* is used requires maximum accuracy, performing a rigorous calibrations is helpful [31]. Since the project requires as much accuracy as possible to create a precision bin picking vision process, this calibration was performed.

The calibration procedure used for the purpose of calibrating both RGB and IR cameras was the one fully described in the tutorials [31] and [32]. In these tutorials the *camera_calibration's cameracalibrator.py* node is used to calibrate a monocular camera with a raw image. For this, a large checkerboard with known dimensions is required. The one used was a 8x6 checkerboard with 105.3mm squares, which can be seen in figure 4.3.

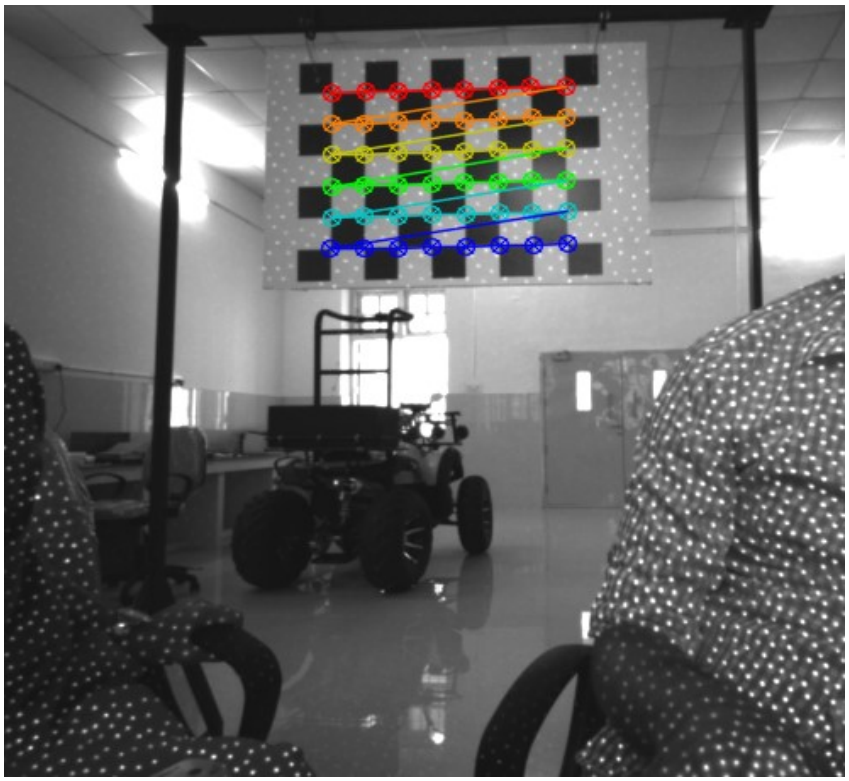


Figure 4.3: Calibration window with the highlighted checkerboard

In the depth calibration, the speckle pattern emitted by the IR projector makes it impossible to accurately detect the checkerboard corners, because of the clarity the speckles create on the surfaces. The solution is to cover the IR light source, thus blocking the speckles, and provide a separate IR light source, such as the sunlight or incandescent lamps [31].

After the intrinsic calibration, the automatically saved parameters, which are stored in the *rgb_deviceID.yaml* and the *depth_deviceID.yaml* files, were used in the launch file which configures the camera info URLs. The URL (Uniform Resource Locator) expresses the location for getting and saving calibration data [33].

4.2.2 Calibration of the Extrinsic Parameters

The extrinsic parameters indicate the external position and orientation of the camera in the 3D world. Mathematically, the position is defined by a translation (t) which is a 3*1 vector and the orientation is defined by a 3*3 rotation matrix (R).

In this case, the camera is mounted on the robot's 6th joint, therefore it is necessary to compute the static transformation from the frame of the robot's joint to the optical frame of the camera. This is referred to as an eye-in-hand type of calibration. To do this calibration, the *visp_hand2eye_calibration* ROS package [34] was used. This package is used to estimate the camera position with respect to its effector (in this circumstance the robot's arm) using the ViSP library.

To compute the relative transformation between the camera and the hand/arm, it is necessary to feed the calibrator node with the */world_effector* and the */camera_object* transformations, which are known. The */world_effector* represents, in this specific case, the dynamic transformation from the global frame (*robot_base_link*) to the frame of the robot's 4th joint (*robot_link_4*). The */camera_object* transformation is calculated using an ArUco marker and the *aruco_detect* package [35] and represents the dynamic transformation between the camera and the ArUco frames. To use this last package, the installation of the fiducial software is necessary. This provides a system that allows a robot to determine its position and orientation by looking at a number of fiducial markers, detected by the *aruco_detect* node, that are fixed in the robot's environment [36]. The process of the ArUco detection can be visualized in figure 4.4.

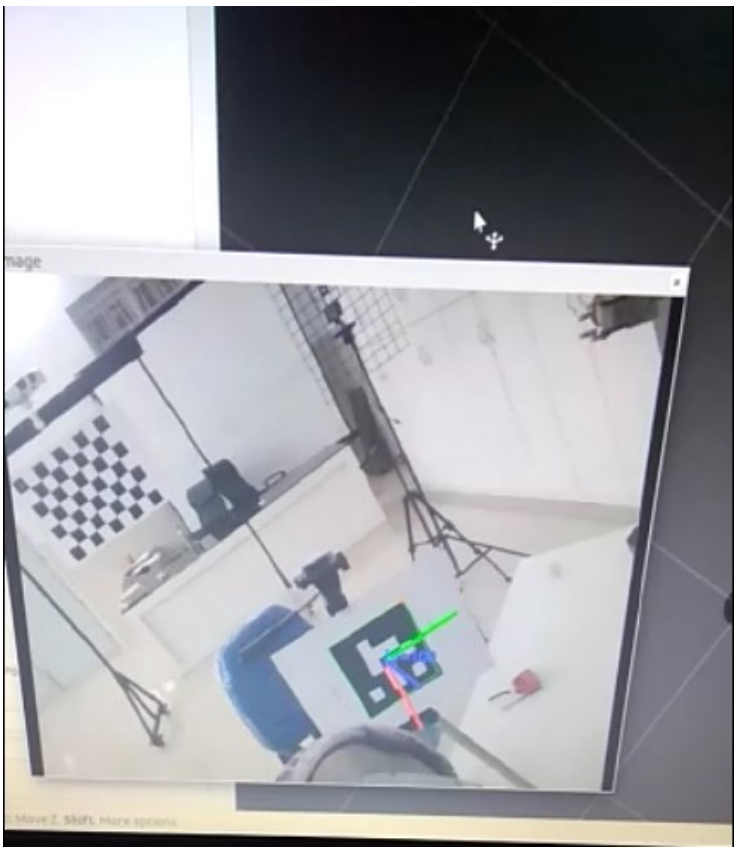


Figure 4.4: ArUco detection

Figure 4.5 presents a simple schematics of the calibration process. This calibration computes the desired static transformation (*/effector_camera*) through the two known dynamic transformations published in the topics */world_effector* and */camera_object* acquired in different poses.

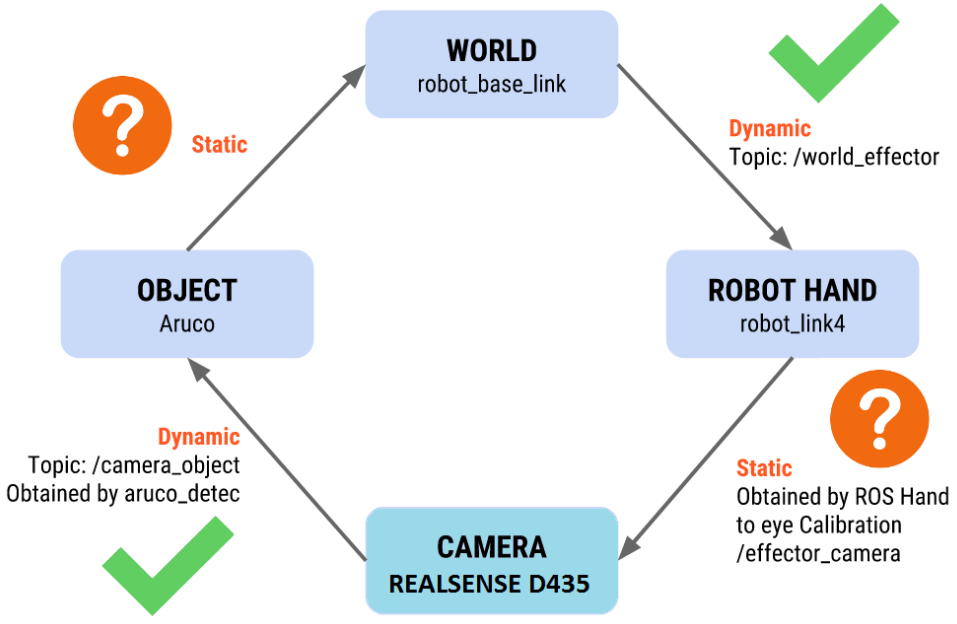


Figure 4.5: The Realsense's extrinsic calibration process

For this calibration to work, the creation of a ROS client is fundamental. This is responsible for feeding the required transformations to the calibrator from a few poses, to compute the relative transformation between the camera and the arm. The *camera_calibration_client.py* program, present in the calibration file, is responsible for executing this process and printing the */effector_camera* transformation in the translation (xyz) and rotation (rpy) format. This is done by running the *kinect_extrinsic_calibration.launch* file in one terminal and the *camera_calibration_client.py* program in another. With this client it is possible to see and record the instantaneous transformations.

Lastly, the */effector_camera* transformation and Realsense *xacro* were included in the *binpicking_macro.xacro* file to create the relation between the robot's 6th joint and the *camera_link* frame, which represents the Realsense.

After all of these calibrations, the robot, the Realsense, the laser sensor and all the associated coordinate frames, can be visualized and analyzed live by running the following command:

```
$ roslaunch bin_picking global_state_visualize.launch
```

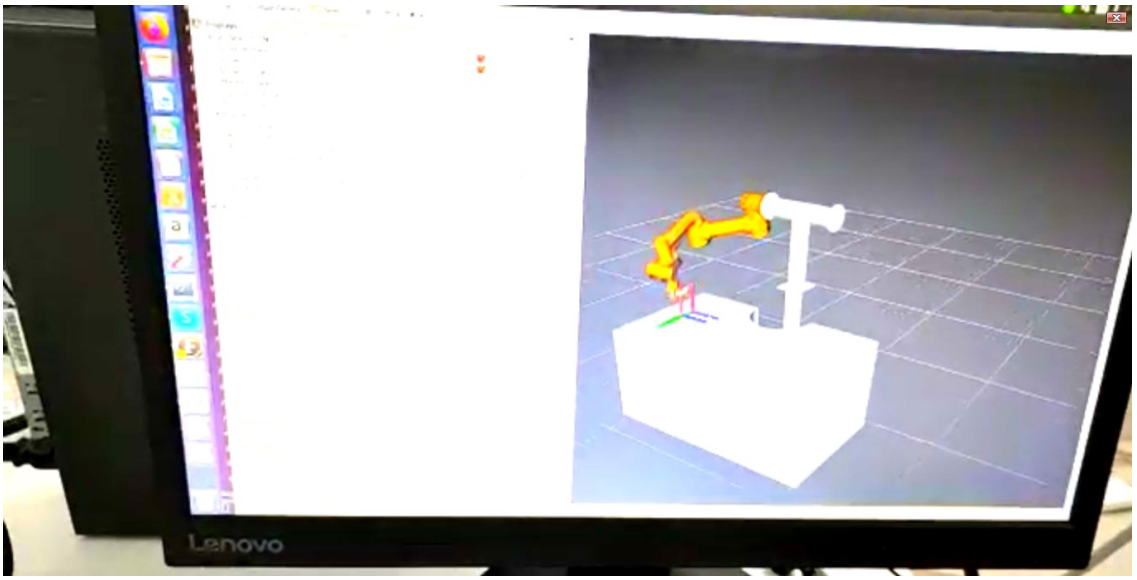


Figure 4.6: Complete system with all the components properly calibrated

It is also possible to visualize the *tf* tree with the frame coordinates of the entire system in figure 4.7. The frames that belong to the Realsense, the end effector and are properly highlighted in the figure. The *tfs* with the prefix robot are all launched through the AUBO driver.

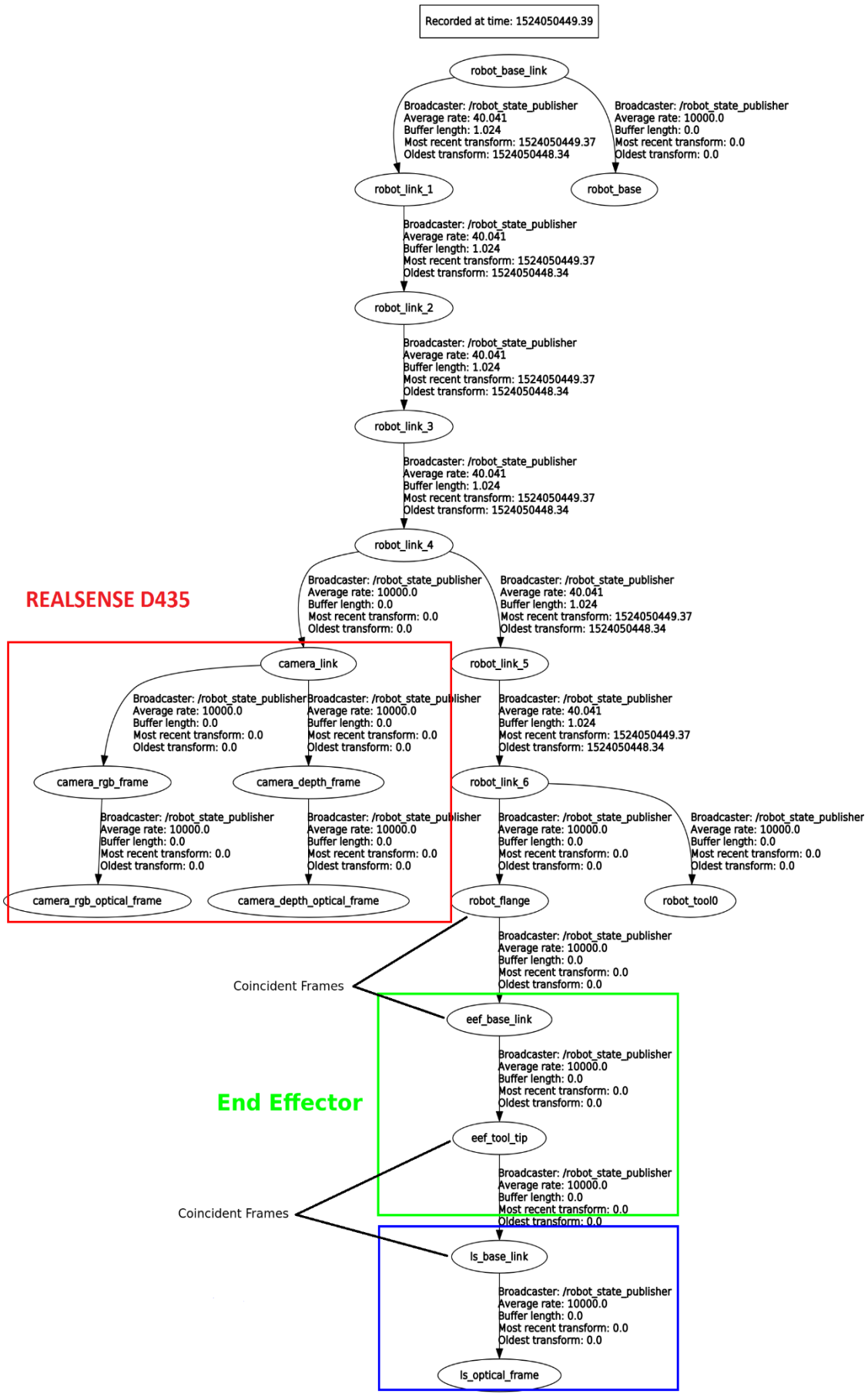


Figure 4.7: *tf* tree of all the coordinate frames

Chapter 5

DATA GENERATION AND TRAINING

A key question addressed by this research is how to generate effective training data for the network. In contrast to 2D object detection, for which labeled bounding boxes are relatively easy to annotate, 3D object detection requires labeled data that is almost impossible to generate manually. Although it is possible to semi-automatically label data (using a tool such as LabelFusion [37]), the labor-intensive nature of the process nevertheless impedes the ability to generate training data with sufficient variation. For example, we are not aware of any real-world training data for 6-DoF object pose estimation that includes extreme lighting conditions or poses.

To overcome these limitations of real data, I turn to synthetically generated data. Specifically, I use a combination of non-photorealistic domain randomized (DR) data and photorealistic data to leverage the strengths of both which is the state of the art method published by NVidia. Synthetic data has an additional advantage in that it avoids overfitting to a particular dataset distribution, thus producing a network that is robust to lighting changes, camera variations, and backgrounds.

Since my goal is precise bin picking vision process of the objects whose 6-DoF pose has been estimated, I used the popular YCB object (Cracker Box) [38]. Both the domain randomized and photorealistic data were created by placing the YCB object model in different virtual environments. All data were generated by a custom plugin we developed for Unreal Engine 4 (UE4) called NDDS [39]. By leveraging asynchronous, multithreaded sequential frame grabbing, the plugin generates data at a rate of 50–100 Hz, which is significantly faster than either the default UE4 screenshot function or the publicly available Sim4CV tool [40]. The steps of the process used to creation of Dataset are listed below [41]:

1. Creation of a 3D CAD models in solid-works (.stl format).
2. Importing the CAD model in the Blender to give color, material, unit scales etc. to object and export as (.fbx) file.
3. Importing the Blender file in the Unreal Engine (UE4) and Creation of the Dataset by using NDDS Software.
4. Traing the Dataset.

5.1 CREATION OF A 3D CAD MODEL IN SOLIDWORKS

SOLIDWORKS focuses on quickly creating 3D solid models of your design, rapidly creating both complex parts and assemblies on screen in 3D as oppose to flat 2D drawings. With all drawing views generated from the original 3D model, SOLIDWORKS ensures any amendments made to the model are automatically updated within the drawing. This automatic associativity guarantees your solid model is always accurately reflected within in your drawings.

Key SOLIDWORKS 3D solid modelling features enable you to:

- Produce 3D solid models of any part and assembly, regardless of size and complexity.
- Synchronise all 3D models, 2D drawings and other design and manufacturing documents thanks to inbuilt associativity, which automatically tracks for any changes and makes updates.
- Create surfacing for any 3D geometry regardless of complexity or stylisation.
- Produce in depth 3D model analysis instantly on an extensive range of properties: mass, density, moments of inertia.

Figure 5.1: 3D model of the Cracker box

Finally we need to analyze and clean our 3D model to remove any faces, edges, and vertices that we don't need. One way to do this is to remove doubles, which will automatically clean the model by minimizing the polygon count. If you want to analyze the model even deeper, you should consider getting the Print Tool Box add-on. It provides a lot of information that's useful for the purposes of making a model printable. Apart from telling you the volume and area of a model, it can correct a design, indicating where the errors are. Other features include checking if the model is solid, if the faces are well oriented, if thicknesses are correct, and if overhangs will be an issue during printing. Finally, in order to transfer your model to a Blender, you will need to export it as an *STL* file. To do so, simply select "Export" in the File menu, and choose .*STL*.

5.2 IMPORTING MODEL IN THE BLENDER

Blender is a free, open-source 3D modeling software used in many industries, including filmmaking, animation, and video games. It's popularity stems from its capacity to handle complex geometries and different modeling styles. When it comes to 3D printing, Blender performs exceptionally well. You only have to be aware of a few issues during the modeling process, and that's exactly what we cover in this article. Important to have in mind when modeling is that your 3D model should be one solid mesh in order to be printed. (A mesh is a set of points, edges, and surfaces that describe a 3D object.) Otherwise, your 3D printer won't be able to correctly interpret your model and it will fail.

Blender has a number of modifiers that make the modeling process much easier. They can be accessed through the Properties bar in Object mode. *Booleans modifier*, with which you can unify, intersect, or subtract one solid with or from another. A very simple way to assure that your 3D model is thick enough is to use the *solidify modifier*. The *Remesh modifier* remodels the topography of an existing mesh, simplifying the way surfaces are represented. The *Decimate modifier* also adjusts the topology, but with a big difference. While remesh respects the shape of the model, maintaining the vertices as much as possible, decimate will reduce the polygon count, removing "unnecessary" vertices and edges.

Finally, in order to transfer your model to a UE4, you will need to export it as an *FBX* file. To do so, simply select "Export" in the File menu, and choose *.fbx*.

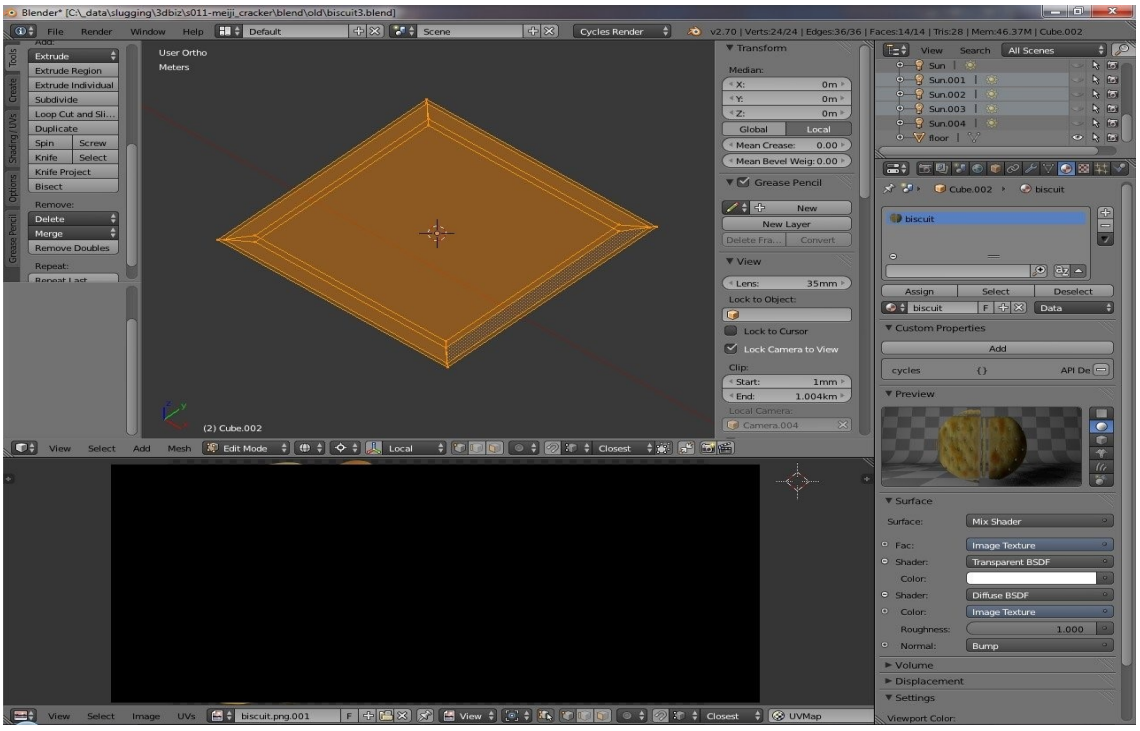


Figure 5.2: Blender interface to give color, material, unit scales

If you want to analyze the model even deeper, you should consider getting the Print Tool Box add-on. It provides a lot of information that is useful for the purposes of making a model printable. Blender is a formidable 3D modeling software, but it does require a bit of practice to get used to. Nevertheless, its many useful tools make it worth the effort, and with these tips, designing your 3D prints should be a breeze.

Figure 5.3: Photorealistic image for all sides of the box in blender

5.3 IMPORTING IN THE UNREAL ENGINE (UE4) AND CREATION OF THE DATASET BY USING NDDS PLUG-IN

The NVidia Deep Learning Dataset Synthesizer comes with demo content of assets to capture simulation data from the Unreal Game Engine. Once you open the Unreal Editor with the *NDDS.uproject*, a default level called *TestCapturer* will load as indicated at the top left hand corner of the 3D view port. This level has a sample scene with a basic simulation capture set up.

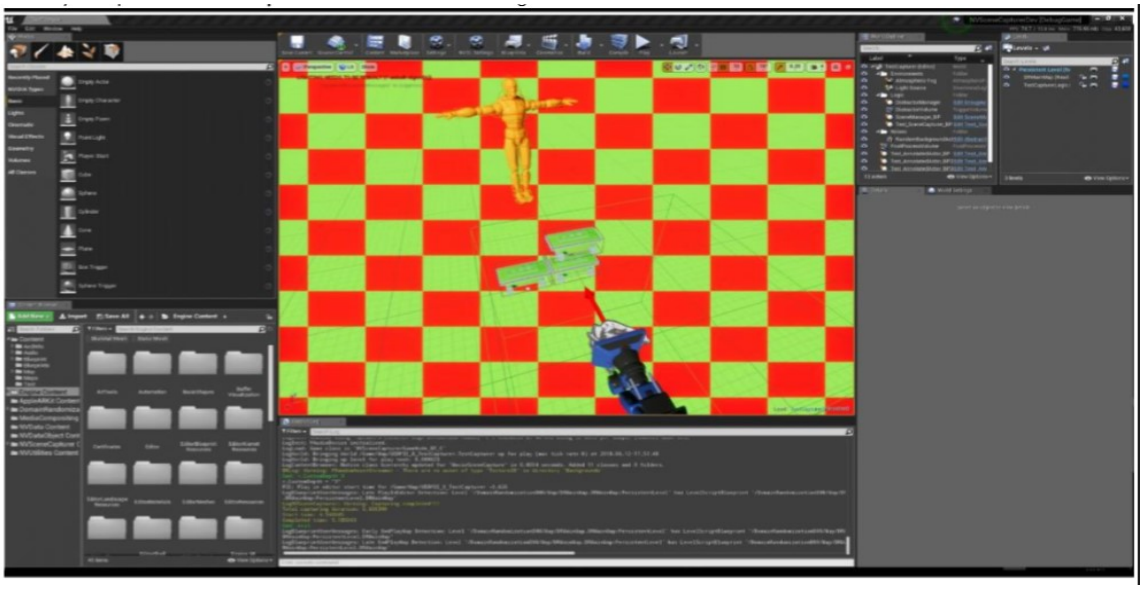


Figure 5.4: Importing the model in UE4 interface [41]

NVSceneCapturer Content

Once plugin content is visible, you will see a *DomainRandomization* DNNCONTENT and *NVSceneCapturer* CONTENT folder in the Content directory, as well as the source files.

- *Capturers*: This folder contains capturer camera objects (configuration + intrinsics) that can be placed in the scene.
- *SceneCapturer_Simple*: is a premade capturer (like a camera) with default feature extractors. It captures the following:
  (1)*ObjectData* (2)*TrueColor* (3)*Depth* (4)*Instance Segmentation* (5)*ClassSegmentation*
- *Feature Extractors:* This folder contains difference capturer Absolute and Quantized depth settings to be used on the Capturer Cameras.
- *Segmentation:* This folder contain feature extractor which segmenting the objects in the scene.
- *Game*: This folder contains Game Mode object and UI actors for the Capturer runtime interface.
- *Material*: This folder contains the Materials used in the Feature Extractors.

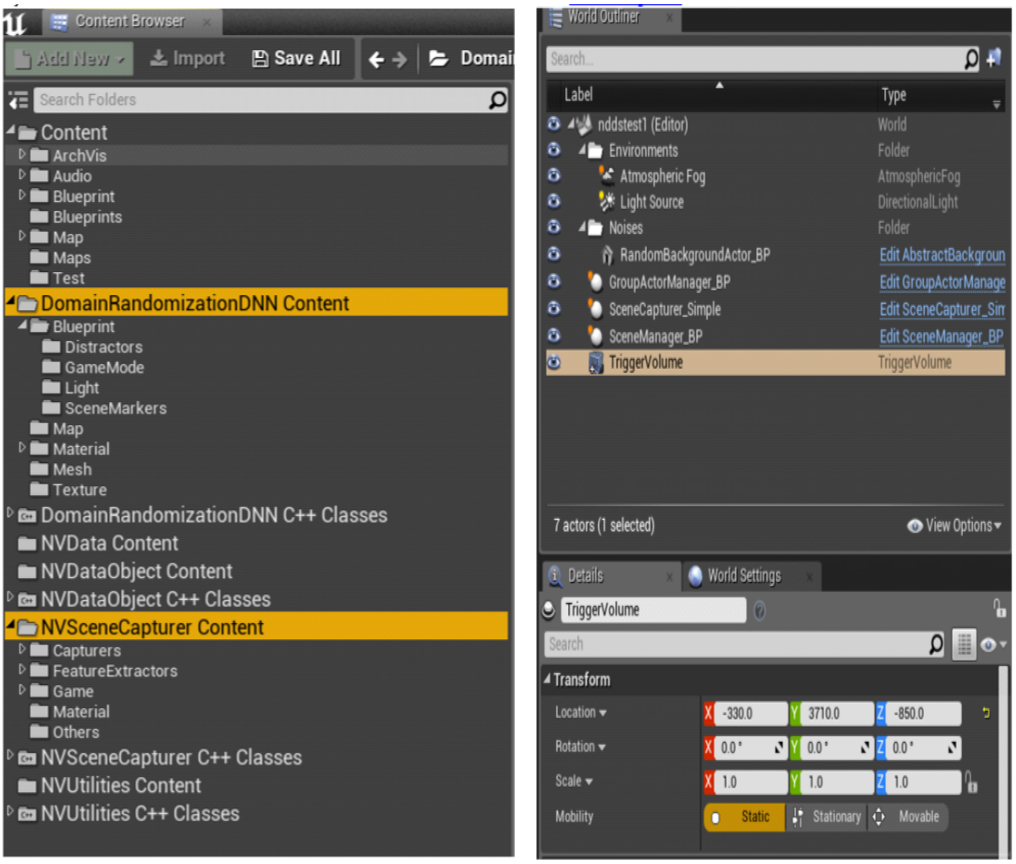


Figure 5.5: Customizing Parameters window [41]

Data Generation using NVidia Deep Learning Data Synthesizer (NDDS)

Domain randomization: The domain-randomized images were created by placing the foreground objects within virtual environments consisting of various distractor objects in front of a random background. Randomly varying distractors, overlaid textures, backgrounds, object poses, lighting, and noise generated images. More specifically, the following aspects of the scene were randomized, similar to [42]: number and types of distractors, selected from a set of 3D models (cones, pyramids, spheres, cylinders, partial toroids, arrows, etc.); texture on the object of interest, as well as on distractors (solid, striped); solid colored background, photograph (from a subset of 10,000 images from the COCO dataset [43]),or procedurally generated background image with random multicolored grid pattern; 3D pose of objects and distractors, sampled uniformly; directional lights with random orientation, color, and intensity; and visibility of distractors.

Photorealistic images: The photorealistic data were generated by placing the foreground objects in 3D background scenes with physical constraints. Backgrounds were chosen from standard UE4 virtual environments such as a kitchen, sun temple, and forest. These environments were chosen for their high-fidelity modeling and quality, as well as for the variety of indoor and outdoor scenes. Allowed to fall under the weight of gravity, and to collide with each other and with surfaces in the scene, these objects interact in physically plausible ways. While the objects were falling, the virtual camera system was rapidly teleported to random azimuths, elevations, and distances with respect to a fixation point to collect data. Azimuth ranged from –120◦ to +120◦ (to avoid collision with the wall, when present), elevation from 5◦ to 85◦, and distance from 0.5 m to 1.5 m. We used this dataset, known as Falling Things (FAT), publicly available [44].

Figure 5.6: Example from our domain randomized (left) and photorealistic (right) [45]

5.4 TRAINING AND TESTING THE DATASET

For training, we used *60k* photorealistic image frames. For data augmentation, Gaussian noise, random contrast and random brightness were added. For PoseCNN [11], we used the publicly available weights, which were trained using an undisclosed synthetic dataset and fine-tuned on images from separate videos of the YCB-Video dataset.

To avoid the vanishing gradients problem with our network, a loss was computed at the output of each stage, using the L2 loss for the belief maps and vector fields. The ground truth belief maps were generated by placing 2D Gaussians at the vertex locations. The ground truth vector fields were generated by setting pixels to the normalized x and y components of the vector pointing toward the object's centroid. Only the pixels within a 3 pixel radius of each ground truth vertex were set in this manner, with all other pixels set to zero. Wherever more than one vertex resided with in this radius, one of them was selected at random to be used for generating these components.

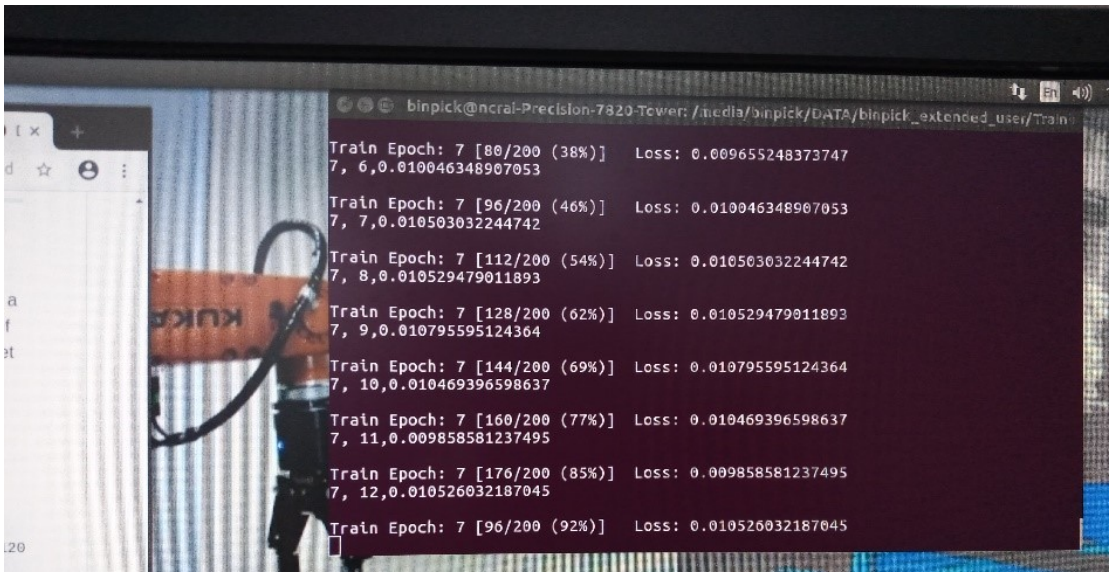


Figure 5.7: Training using GPU

There are several natural ways to split the dataset for training and testing. One approach would be to hold out one location per scene as the test sets, and leave the other data for training. Another approach would be to hold out one environment for testing, leaving the others for training. Finally, the single object images could be used for training, while using the remaining images for testing that is, assuming that occlusion is artificially introduced in the data augmentation process during training. Further details regarding training/testing methodology can be found with the dataset.

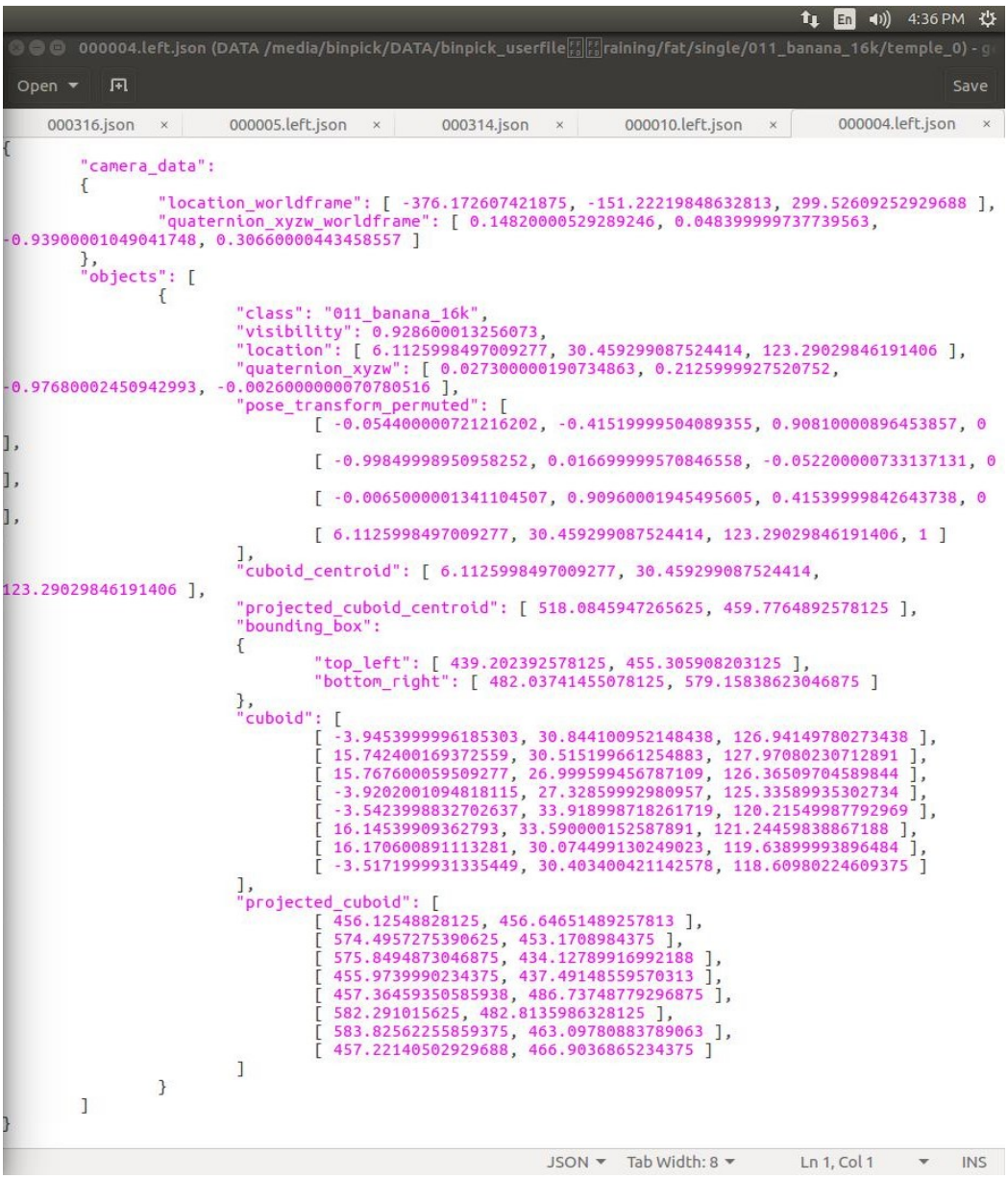


Figure 5.8: JSON file preview

This network was implemented using PyTorch v0.4 [45]. The VGG-19 feature extractions were taken from publicly available trained weights in torchvision open models. The networks were trained for 60 epochs with a batchsize of 16. Adam [46] was used as the optimizer with learning rate set at 0.0001. The system was trained on a DELL PRECISION 7820 workstation (containing NVIDIA P4000 GPU), and testing used same.

Chapter 6

DETECTION AND POSE ESTIMATION

Having all the hardware correctly calibrated, and being already capable of acquiring Realsense data, the next important step is processing the camera data. Therefore, this chapter presents a detailed explanation of the Realsense data processing to detect and estimate the pose of objects. The camera data used to develop this process was recorded into a bag through the *rosbag* package, so that the code could be written and tested without being always connected to the Realsense sensor. A bag is a file format in ROS for storing ROS message data [27].

I find a two-step solution to address the problem of detecting and estimating the 6-DoF pose of all instances of a set of known household objects from a single RGB image. First, a deep neural network estimates belief maps of 2D keypoints of all the objects in the image coordinate system. Secondly, peaks from these belief maps are fed to a standard perspective-n-point (PnP) algorithm [13] to estimate the 6-DoF pose of each object instance. That is, a one shot, deep neural network-based system that infers, in near real time, the 3D poses of known objects in clutter from a single RGB image without requiring post-alignment. This system is called DOPE (for “deep object pose estimation”) created by NVidia [47].

6.1 NETWORK ARCHITECTURE

This sytem developed by NVida is inspired by convolutional pose machines (CPMs) [48] [49], their one-shot fully convolutional deep neural network detects keypoints using a multistage architecture. The feed forward network takes as input an RGB image of size of w×h×3 and branches to produce two different outputs, namely, belief maps and vector fields. There are nine belief maps, one for each of the projected 8 vertices of the 3D bounding boxes, and one for the centroids. Similarly, there are eight vector fields indicating the direction from each of the 8 vertices to the corresponding centroid, similar to [11], to enable the detection of multiple instances of the same type of object. (In our experiments, w = 640, h = 480)

The network operates in stages, with each stage taking into account not only the image features but also the outputs of the immediately preceding stage. Since all stages are convolutional, they leverage an increasingly larger effective receptive field as data pass

through the network. This property enables the network to resolve ambiguities in the early stages due to small receptive fields by incorporating increasingly larger amounts of context in later stages.

Image features are computed by the first ten layers from VGG-19 [50] (pretrained on ImageNet), followed by two 3×3 convolution layers to reduce the feature dimension from 512 to 256, and from 256 to 128. These 128-dimensional features are fed to the first stage consisting of three 3×3×128 layers and one 1×1×512 layer, followed by either a 1×1×9 (belief maps) or a 1×1×16 (vector fields) layer. The remaining five stages are identical to the first stage, except that they receive a 153 dimensional input (128+16+9 = 153) and consist of five 7×7×128 layers and one 1×1×128 layer before the 1 × 1 × 9 or 1 × 1 × 16 layer. All stages are of size w/8 and h/8, with ReLU activation functions interleaved throughout.

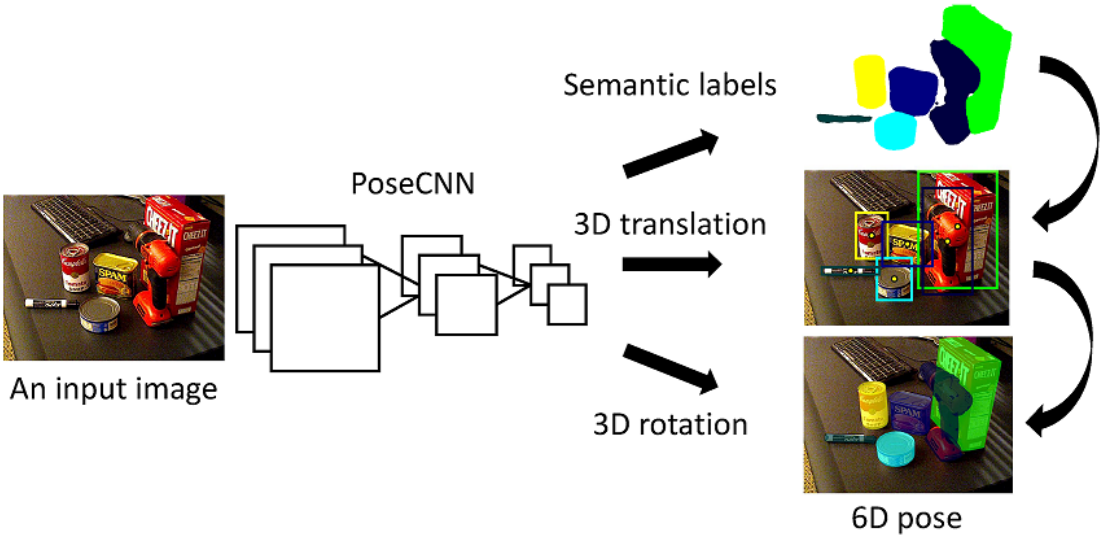


Figure 6.1: Network diagram [11]

6.2 DETECTION

After the network has processed an image, it is necessary to extract the individual objects from the belief maps. A one-shot, deep neural network-based system that infers, in near real time, the 3D poses of known objects in clutter from a single RGB image without requiring post-alignment. This system uses a simple deep network architecture, trained entirely on simulated data, to infer the 2D image coordinates of projected 3D bounding boxes, followed by perspective-n-point (PnP) [13].

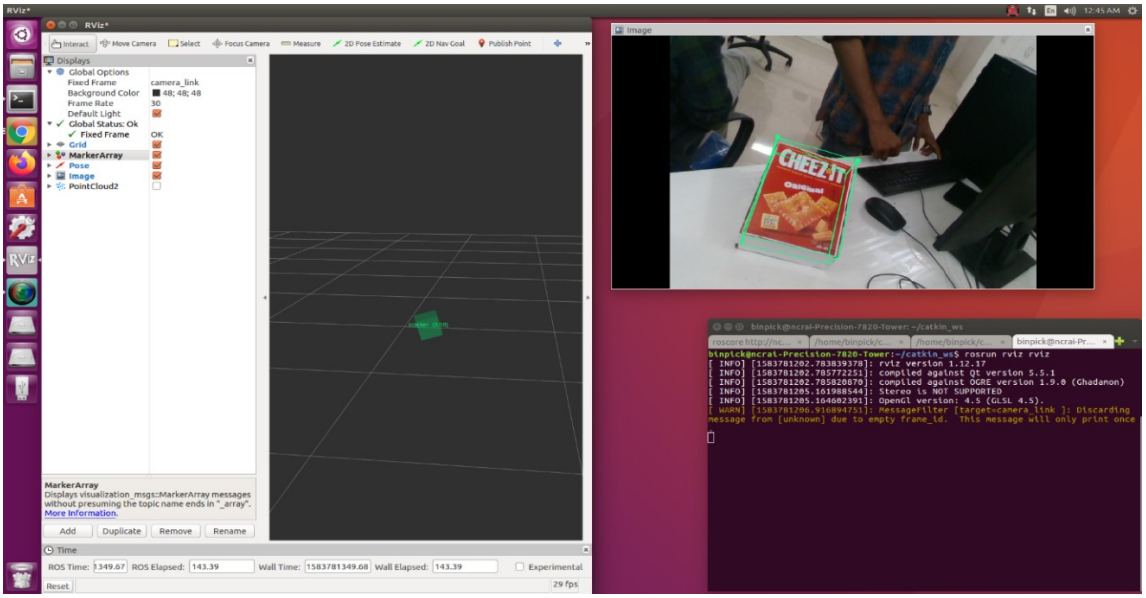


Figure 6.2: Detection of the Cracker Box Front Side

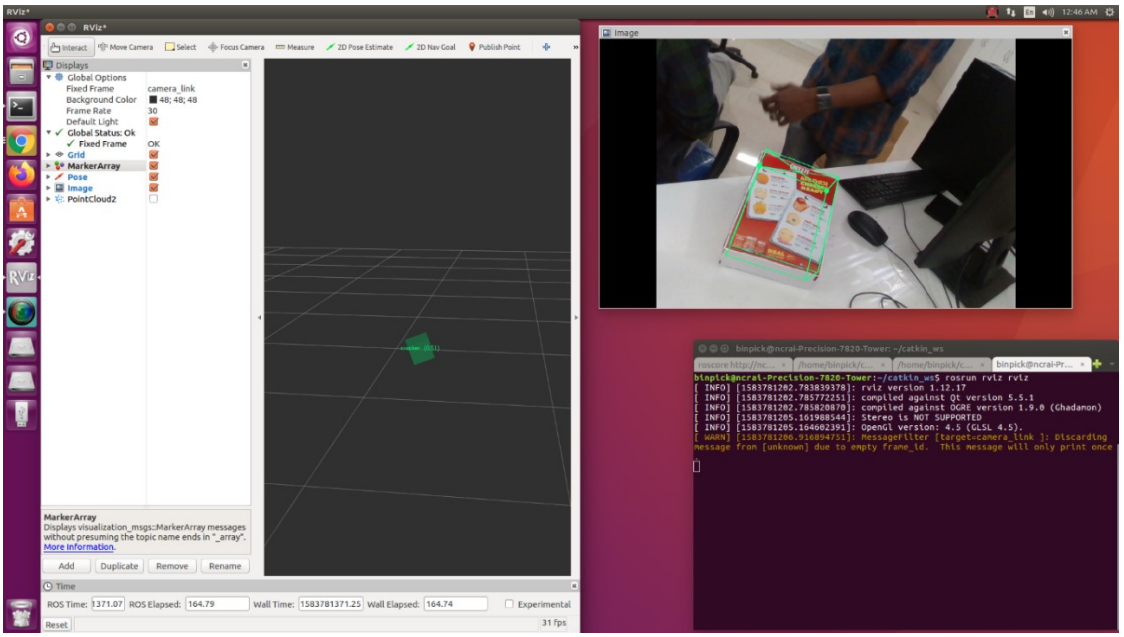


Figure 6.3: Detection of the Cracker Box Back Side

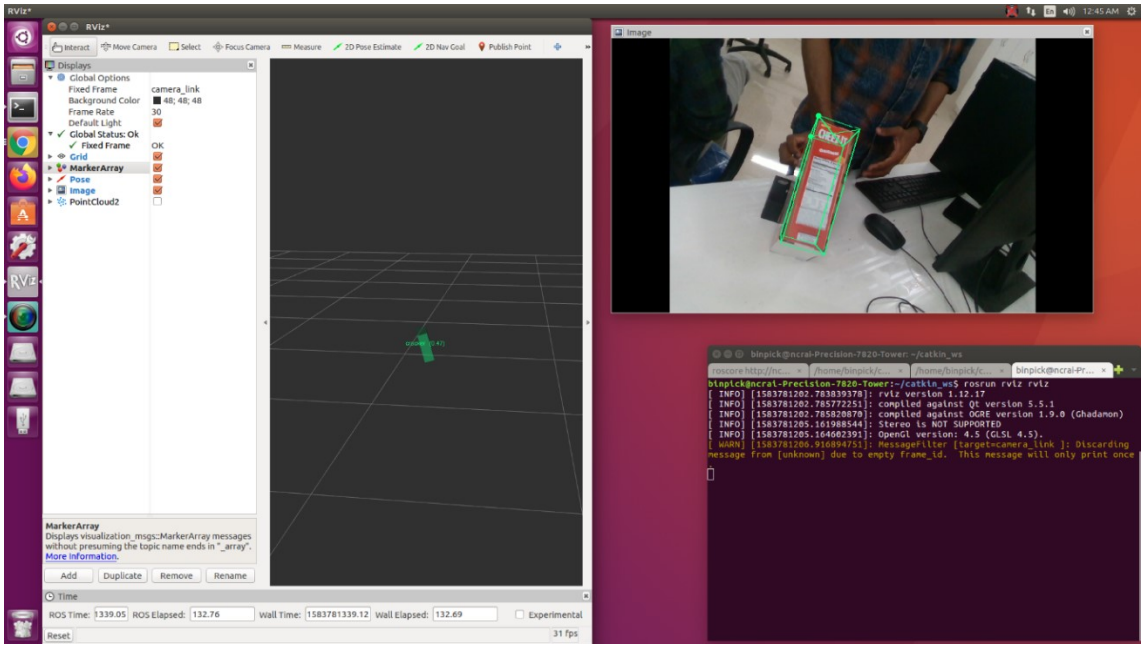


Figure 6.4: Detection of the Cracker Box Left Side

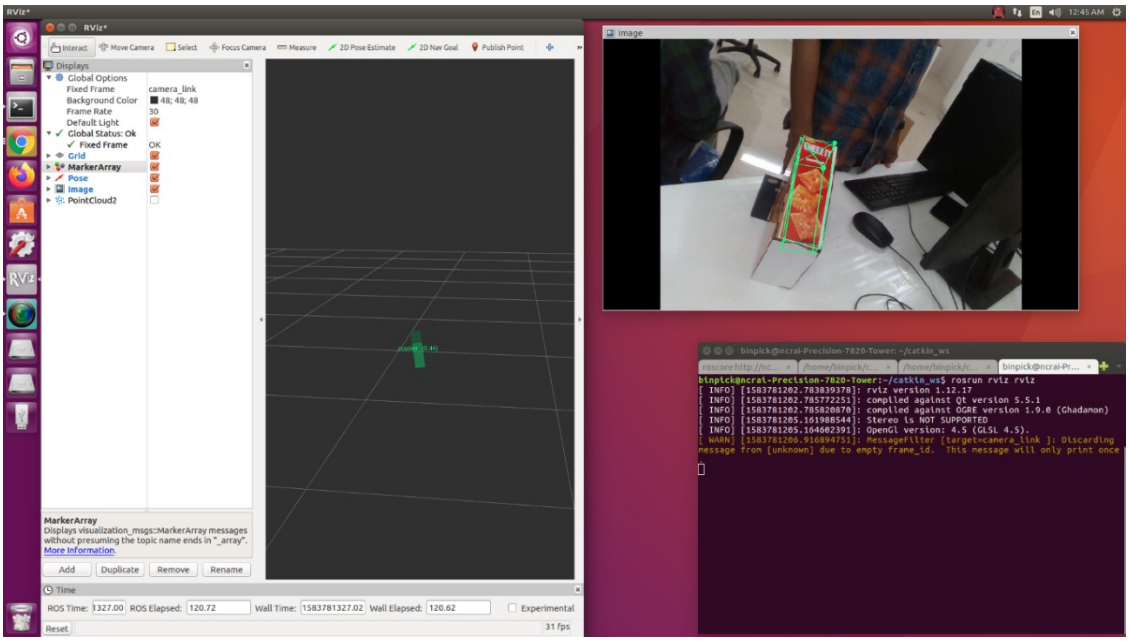


Figure 6.5: Detection of the Cracker Box Right Side

Demonstration that combining both non-photorealistic (domain randomized) and photorealistic synthetic data for training robust deep neural networks successfully bridges the reality gap for real-world applications, achieving performance comparable with state-of the-art networks trained on real data.

6.3 POSE ESTIMATION

In contrast to other approaches in which complex architectures or procedures are required to individuate the objects [10] [11] [51] [12], our approach relies on a simple post processing step that searches for local peaks in the belief maps above a threshold, followed by a greedy assignment algorithm that associates projected vertices to detected centroids. For each vertex, this latter step compares the vector field evaluated at the vertex with the direction from the vertex to each centroid, assigning the vertex to the closest centroid within some angular threshold of the vector.

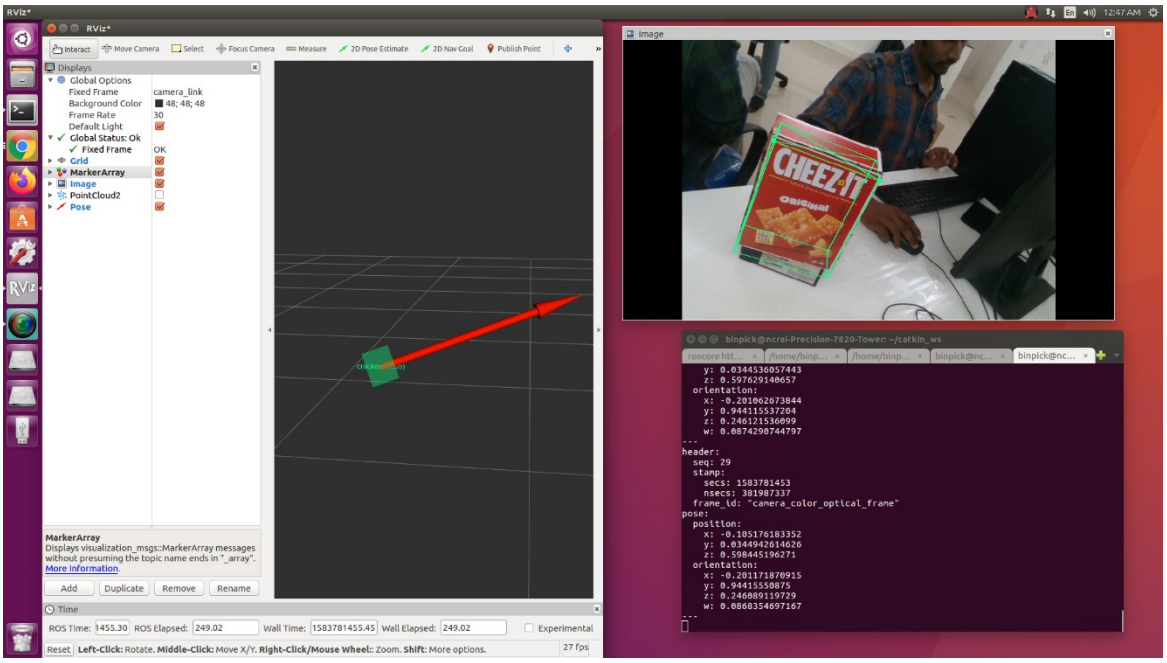


Figure 6.6: Pose Estimation of the Cracker Box Front Side

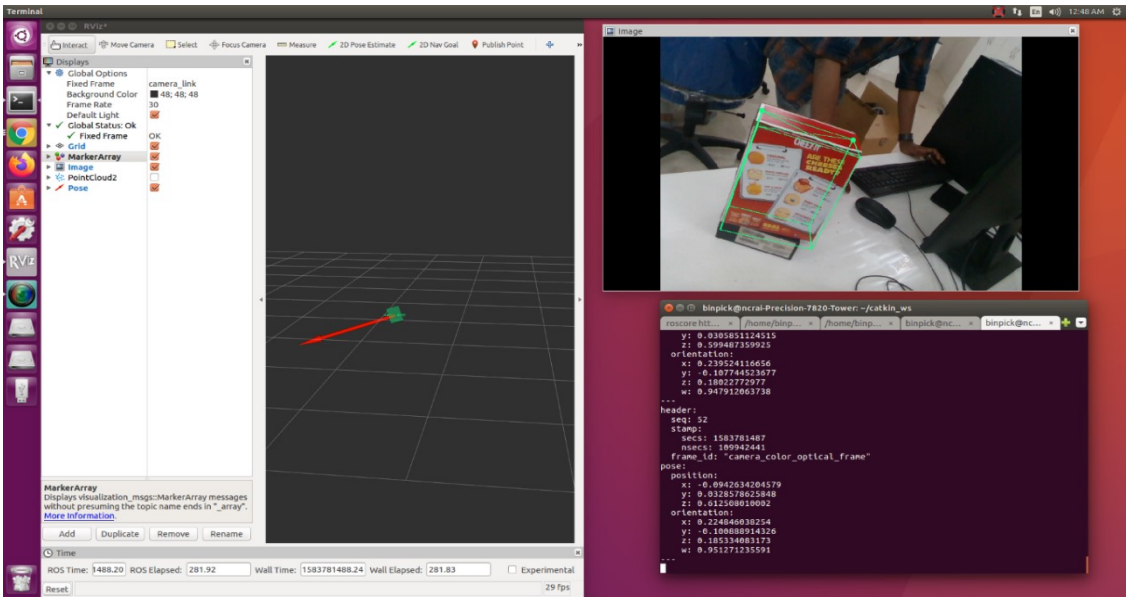


Figure 6.7: Pose Estimation of the Cracker Box Back Side

Once the vertices of each object instance have been determined, a PnP algorithm [13] is used to retrieve the pose of the object, similar to [12] [10]. This step uses the detected projected vertices of the bounding box, the camera intrinsics, and the object dimensions to recover the final translation and rotation of the object with respect to the camera. All detected projected vertices are used, as long as at least the minimum number (four) are detected.

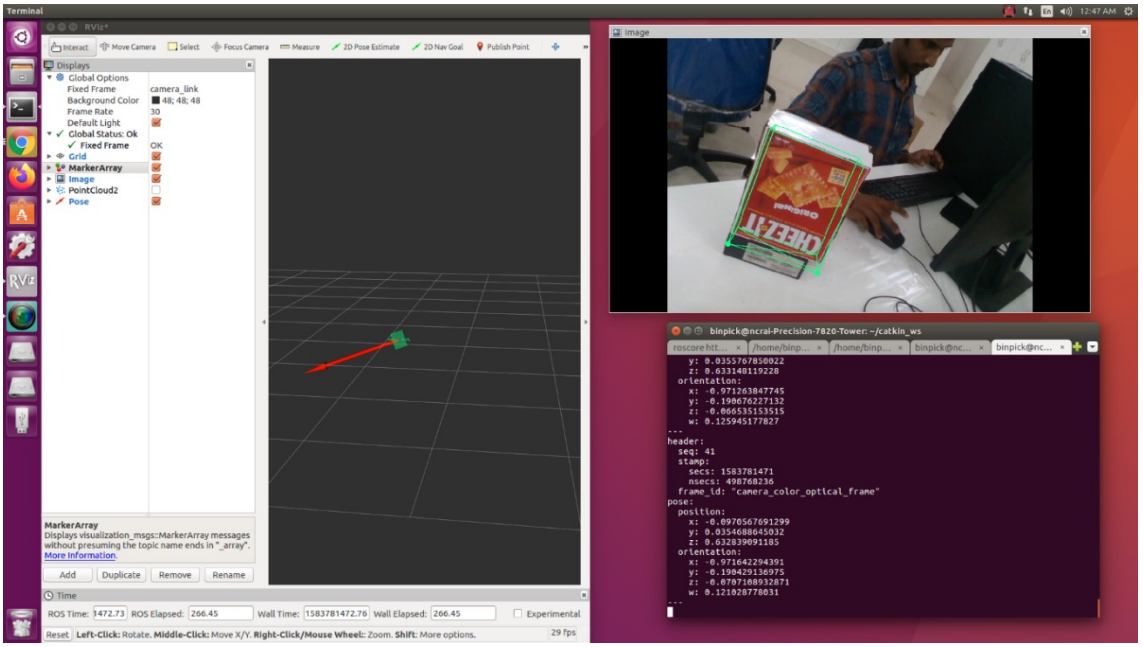


Figure 6.8: Pose Estimation of the inverted Cracker Box

Chapter 7

SYSTEM INTEGRATION

With the complete hardware calibrated, and having the object in the workspace successfully detected, it is now possible to develop the precise bin-picking vision system in ROS Environment. The description of the steps required for the development and organization of the entire process is presented in this chapter.

As a part of open sourcing the project work for more feedback and suggestions, I have tried to publishing in GitHub Open source [52]. GitHub offers plans free of charge and professional and enterprise accounts. Free GitHub accounts are commonly used to host open source projects.

Also for documentation of the Project work Diary and preparation of final report, I have tried to publishing in GitHub Pages (Blog) [53]. GitHub Pages is a static site hosting service that takes HTML, CSS, and JavaScript files straight from a repository on GitHub, optionally runs the files through a build process, and publishes a website.

7.1 GITHUB REPOSITORY FOR THE ROS PACKAGE

A repository is like a folder for your project. My project's repository contains all of my project's files and stores each file's revision history. We can also discuss and manage our project's work within the repository. We can own repositories individually, or we can share ownership of repositories with other people in an organization. For user-owned repositories, we can give other people collaborator access so that they can collaborate on our project. With GitHub Free for user accounts and organizations, we can work with unlimited collaborators on unlimited public repositories with a full feature set, or unlimited private repositories with a limited feature set. To get advanced tooling for private repositories, we can upgrade to GitHub Pro, GitHub Team, or GitHub Enterprise Cloud.

Repository : AUBOi5-D435-ROS-DOPE (Aubo i5 Dual Arm Collaborative Robot - RealSense D435 - 3D Object Pose Estimation - ROS)

A package for detecting and estimating the 6-DoF pose of known objects using a novel architecture and data generation pipeline using the state of the art algorithm DOPE in Aubo i5 collaborative robot with Intel Realsense D435i camera. The neural network consists several steps to refine and estimates of the 2D coordinates of projected vertices of each object’s 3D bounding cuboid. These vertices are then used to output the final pose using PnP, with known camera intrinsic and object dimensions. The neural network trained only

on photorealistic data can attain state of the art results compared with a neural network trained on real world data and resulting poses are with much needed accuracy for robotic pick and place.

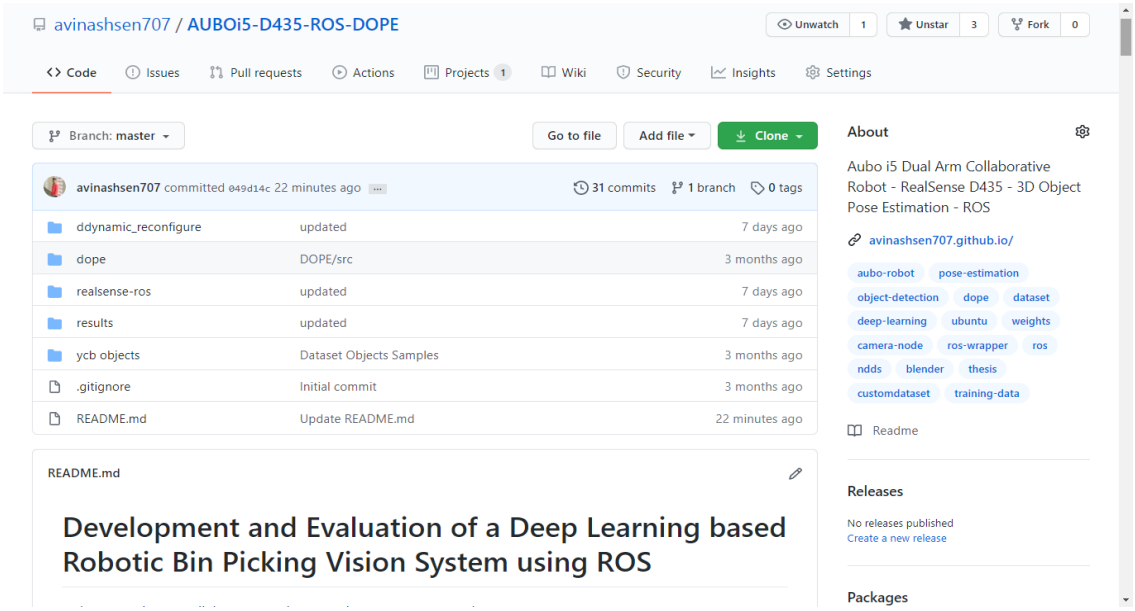


Figure 7.1: Published Repository in GitHub Platform

Installing

I have tested my work on Ubuntu 16.04 with ROS Kinetic with an NVIDIA Quadro P4000 in python 2.7. The following steps describe the native installation.

Step 1: Download the DOPE code

```
cd ~/catkin_ws/src
git clone https://github.com/avinashsen707/AUBOi5-D435-ROS-DOPE
```

Step 2: Install python dependencies

```
cd ~/catkin_ws/src/dope
pip install -r requirements.txt
```

Step 3: Install ROS dependencies

```
cd ~/catkin_ws
rosdep install --from-paths src -i --rosdistro kinetic
sudo apt-get install ros-kinetic-rosbash ros-kinetic-ros-comm
Build

cd ~/catkin_ws
catkin_make
```

Step 4: Download the weights and save them to the weights folder, i.e., *~/catkin_ws/src/dope/weights/.*

ROS Wrapper for Intel® RealSense D435 - Ubuntu 16.04_ROS Kinetic

Step 1: Install the latest Intel® RealSense™ SDK 2.0

Install from Debian Package follow the instructions to also install *librealsense2-dev* and *librealsense-dkms* packages. OR Build from sources by downloading the latest Intel® RealSense™ SDK 2.0 and follow the instructions under Linux Installation

Step 2: Install the ROS distribution Install ROS Kinetic, on Ubuntu 16.04

Step 3: Install Intel® RealSense™ ROS from Sources

```
cd ~/catkin_ws/src/
```

Clone the latest Intel® RealSense™ ROS from here into *'catkin_ws/src/'*

```
git clone https://github.com/IntelRealSense/realsense-ros.git
cd realsense-ros/
git checkout `git tag | sort -V | grep -P "^\d+\.\d+\.\d+" | tail -1`
cd ..
```

Verified all dependent packages are installed. You can check *.travis.yml* file for reference. Specifically, make sure that the ros package *dynamic_reconfigure* is installed. If *dynamic_reconfigure* cannot be installed using APT, you may clone it into your workspace *'catkin_ws/src/'* from here (Version 0.2.0)

```
catkin_init_workspace
cd ..
catkin_make clean
catkin_make -DCATKIN_ENABLE_TESTING=False -DCMAKE_BUILD_TYPE=Release
catkin_make install
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Running

1. Start ROS master

```
cd ~/catkin_ws
source devel/setup.bash
roscore
```

2. Start camera node (or start your own camera node)

Realsense D435 & usb_cam node (./dope/config/config_pose.yaml):

```
topic_camera: "/camera/color/image_raw"            #"/usb_cam/image_raw"
topic_camera_info: "/camera/color/camera_info"     #"/usb_cam/camera_info"
```

Start camera node:

```
roslaunch realsense2_camera rs_rgbd.launch  # Publishes RGB images to
`/camera/color/image_raw`
```

3. Start DOPE node

```
roslaunch dope dope.launch [config:=/path/to/my_config.yaml]  # Config file is
optional; default is `config_pose.yaml`
```

4. Start rviz node

```
rosrun rviz rviz
```

Debugging

The following ROS topics are published (assuming topic_publishing == 'dope'):

```
    /dope/webcam_rgb_raw       # RGB images from camera
    /dope/dimension_[obj_name] # dimensions of object
    /dope/pose_[obj_name]      # timestamped pose of object
    /dope/rgb_points           # RGB images with detected cuboids overlaid
    /dope/detected_objects # vision_msgs/Detection3DArray of all detected
objects
    /dope/markers              # RViz visualization markers for all objects
*Note:* `[obj_name]` is in {cracker}
```

To debug in RViz, run rviz, then add one or more of the following displays:

- `Add > Image` to view the raw RGB image or the image with cuboids overlaid
- `Add > Pose` to view the object coordinate frame in 3D.
- `Add > MarkerArray` to view the cuboids, meshes etc. in 3D.
- `Add > Camera` to view the RGB Image with the poses and markers from above.

In order to have a coordinate frame set up, you can run this static transformation: *rosrun tf2_ros static_transform_publisher* world <*camera_frame_id*>, where <*camera_frame_id*> is the frame_id of your input camera messages.

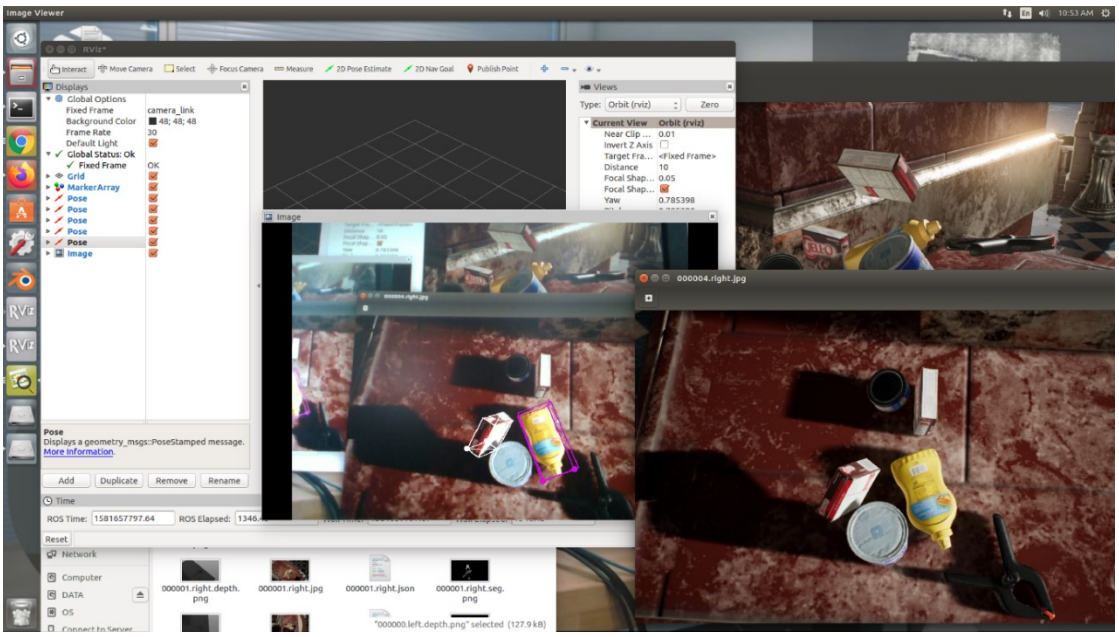


Figure 7.2: ROS interface of the Package

7.2 GITHUB PAGE FOR THE PROJECT BLOG

GitHub Pages are public webpages hosted and easily published through GitHub. The quickest way to get up and running is by using the Jekyll Theme Chooser to load a pre-made theme. We can then modify your GitHub Pages' content and style remotely via the web or locally on our computer. There are three types of GitHub Pages sites: project, user, and organization. Project sites are connected to a specific project hosted on GitHub, such as a JavaScript library or a recipe collection. User and organization sites are connected to a specific GitHub account.

To publish a user site, we must create a repository owned by our user account that's named *<user>.github.io*. To publish an organization site, we must create a repository owned by an organization that's named *<organization>.github.io.* Unless we are using a custom domain, user and organization sites are available at *http(s)://<username>.github.io* or *http(s)://<organization>.github.io.*

The publishing source for our GitHub Pages site is the branch or folder where the source files for our site are stored. All sites have a default publishing source, and project sites have additional publishing sources available.

The default publishing source for user and organization sites is the *master* branch. If the repository for our user or organization site has a *master* branch, our site will publish automatically from that branch. We cannot choose a different publishing source for user or organization sites.

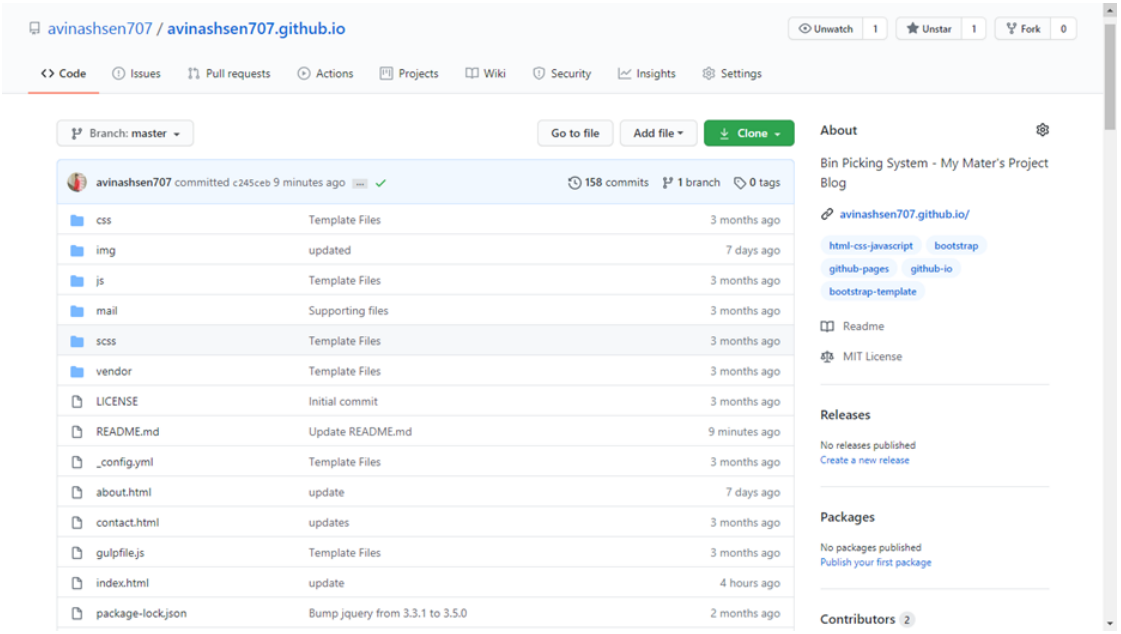


Figure 7.3: Published Repository of GitHub Page

The default publishing source for a project site is the *gh-pages* branch. If the repository for our project site has a *gh-pages* branch, our site will publish automatically from that branch. Unlike user and organization Pages, Project Pages are kept in the same repository as the project they are for, except that the website content is stored in a specially named *gh-pages* branch or in a *docs* folder on the *master* branch. The content will be rendered using Jekyll, and the output will become available under a subpath of our user pages subdomain, such as *username.github.io/project* (unless a custom domain is specified).

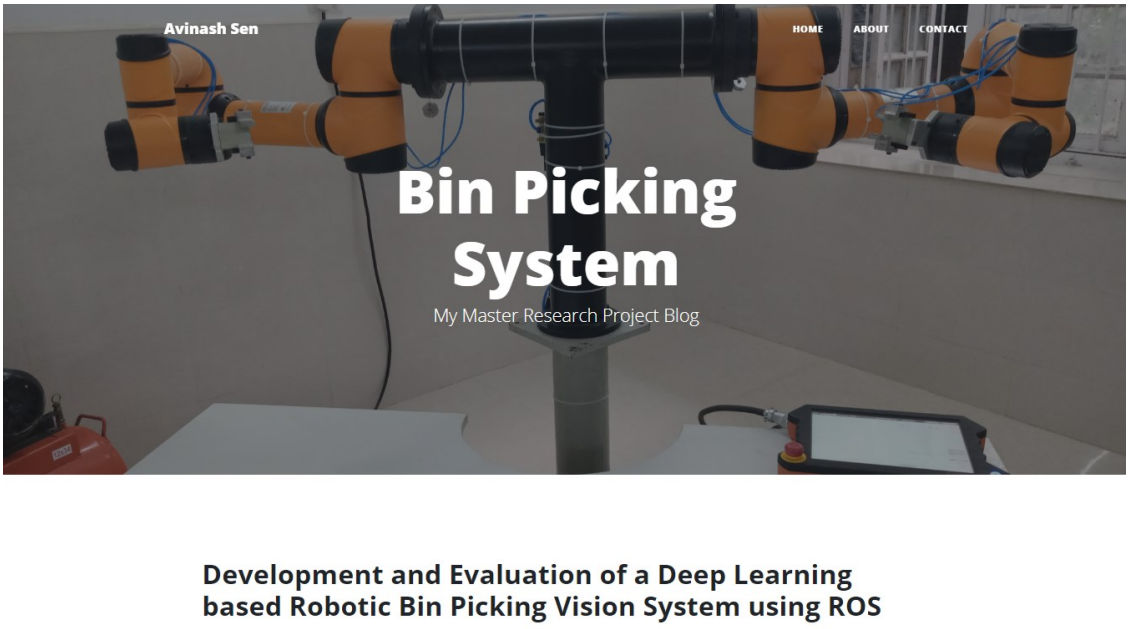


Figure 7.4: Webpage of the Project Blog

Chapter 8

EVALUATIONS AND RESULTS

In order to evaluate and demonstrate the performance of the developed bin-picking vision system, several tests were taken and a demonstration was held to present the versatility of the system. Therefore, this chapter presents the description of all the experiments performed to accomplish the previously mentioned task.

For this particular project, this sort of assessment could be done by evaluating the number of successes in the identification and pose estimation of object. Consequently, eight tests were taken with the object cracker box. All of these tests consist of detecting and pose estimating the object, following the steps presented in the chapter 7 of the System Integration.

8.1 PERFORMED TESTS

I set up a random bin picking test environment using a robotic vision system, and conducted experiments on the proposed method. In the experimental study, cracker box was used for the objects. The object were located approximately 800 mm away from the camera.

I applied the object detection and pose estimation functions, and compared the measured distance and actual distance values. In the distance measurements taken from the input images, the offset value is applied to correct the actual distance. Samples of the experimental results for the charging cradles and travel adapters are shown in Table 8.1.

Table 8.1: Evaluation results

Distance (mm)	**Measure (x_i mm)**	**Error**
807.2	807.1	0.1
828.4	828.8	0.4
837.2	836.6	0.6
829.6	828.3	1.3
810.4	811.2	0.8
827.1	824.9	2.2
812.2	812.1	0.1
810.8	809.8	1.0
Avg.		*0.81*

In the experiment on the cracker box, the mean distance error was 0.81, and the standard deviation was 0.7, for eight set trails. We repeated the experiment on the cracker box five times. The results of this experiment are shown in Table 8.2.

N = total number of trials x_i = measured distance **μ** = mean of the measured values

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

Table 8.2: Distance accuracy

Test Case	Avg.Err	Std.Dev (σ)
1	0.81	0.70
2	0.76	0.59
3	0.67	0.33
4	0.93	0.62
5	1.09	0.43
Avg.	*0.85*	*0.53*

8.2 DEMONSTRATION

As demonstrated through the experiment results above, the average error rate in the distance to the target object was around 1 mm. Therefore, even when assuming that an error of up to 2 mm may occur, there was no problem for the robot to pick up the object in our bin picking environment. The developed bin picking vision system with a robot is shown in Figure 8.1.

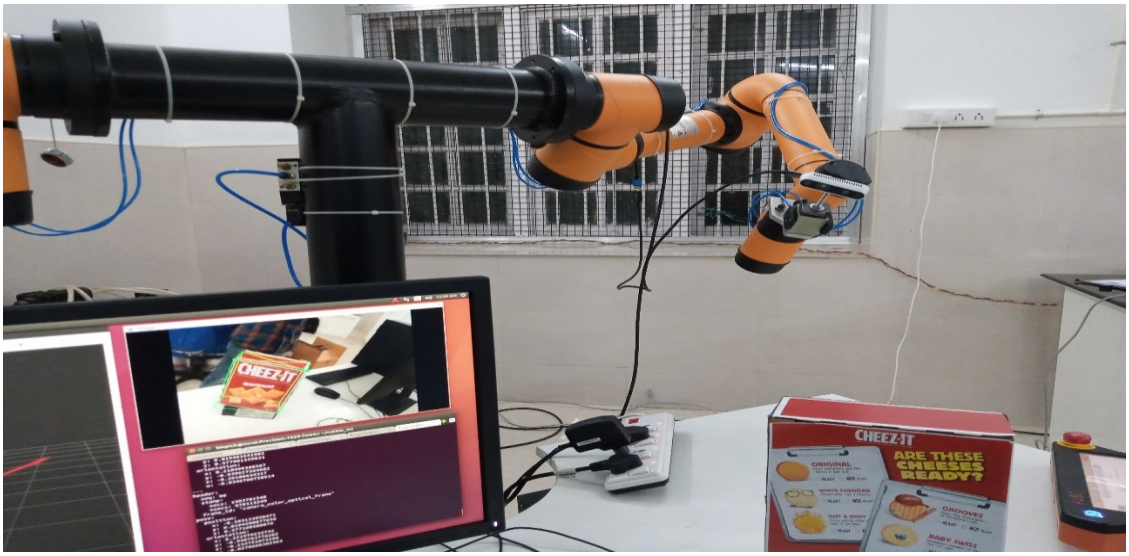


Figure 8.1: Bin Picking Vision System

Chapter 9

CONCLUSIONS AND FUTURE WORKS

During the course of this project, a solution for the execution of a precise bin picking vision system capable of detecting and estimating pose of objects using deep learning was accomplished, thus achieving the main objective.

9.1 CONCLUSIONS

This process began with the setup and installation of all necessary hardware and software, with specific reference to the modulation of support developed to install sensor in the manipulator for the creation of an active perception unit.

After having everything correctly installed, the calibration was performed in order to incorporate the manipulator and the camera sensor in a global system. Then it was possible to replicate a URDF model of the entire system, thus keeping track of all the frames throughout the bin-picking process. The methods used in the Realsense's extrinsic and intrinsic calibration are both available online.

Upon obtaining and processing the Realsense's data, one of the main objectives of this project was achieved through using a simple deep network architecture, trained entirely on Synthetic data, capable of generating an almost unlimited amount of pre-labeled training data with little effort. The network employs multiple stages to refine ambiguous estimates of the 2D locations of projected vertices of each object's 3D bounding cuboid. These points are then used to predict the final pose using PnP, assuming known camera intrinsics and object dimensions. We have shown that a network trained only on synthetic data can achieve state of the art performance.

The Robotic system that shows the estimated poses are of sufficient accuracy and are integrated into a ROS (Robot Operating System) infrastructure computed in the second stage, represents the most prominent and innovative feature of this project. At last, the successfully performed tests demonstrated the viability and the reliability of the developed bin picking vision system.

8.2 FUTURE WORKS

The developed system results from the working combination of different simple tasks, therefore these should be individually improved to create an even more robust process.

Further research should be aimed at increasing the number of objects, handling symmetry, and incorporating closed-loop refinement to increase grasp success. Moreover, it is needed to apply practical solutions for multiple object localization to many different real-world situations. The robotic manipulation of the project is in the future scope stage and it will be fully made into grasp the respective object from the randomly ordered bin.

As a another future scope of this project, Creating a custom dataset for the objects in any problem domain using NDDS (NVidia Deep Learning Data Synthesizer) and testing in DOPE algorithm can be achieved.

REFERENCES

[1] The Economist, "The growth of industrial robots - Daily chart," 2017. [Online]. Available: https://www.economist.com/blogs/graphicdetail/2017/03/daily-chart-19. [Accessed 03 July 2019].

[2] RobotWorx, "Advantages and Disadvantages of Automating with Industrial Robots," [Online]. Available: https://www.robots.com/blog/viewing/advantages-and-disadvantages-of-automating-with-industrial-robots. [Accessed 6 July 2019].

[3] Alex Owen-Hill, "Robot Vision vs Computer Vision: What's the Difference?," 2016. [Online]. Available: https://blog.robotiq.com/robot-vision-vs-computer-vision-whats-the-difference. [Accessed 09 July 2019].

[4] IncEPICSystems, "Quick History of Machine Vision," 2017. [Online]. Available: https://www.epicsysinc.com/blog/machine-vision-history. [Accessed 09 July 2019].

[5] teqrma, "Vision Guided Robotics," Bin Picking - The revolutionary technique for industry 4.0, [Online]. Available: https://teqram.com/solutions/bin-picking/. [Accessed 09 July 2019].

[6] RIA - Robotic Industries Association, "Robotics Industry Insights - The Pervasive Relevance of Bin Picking in Nature and Business," 2011. [Online]. Available: https://www.robotics.org/content-detail.cfm?content_id=3080. [Accessed 09 July 2019].

[7] J. Dai, Y. Li, K. He, and J.Sun, "R-FCN: Object detection via region-based fully convolutional networks," *NIPS,* 2016.

[8] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *CVPR,* 2017.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, " SSD: Single shot multibox detector," *ECCV,* 2016.

[10] M. Rad and V. Lepetit. BB8, "A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth," *ICCV,* 2017.

[11] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," *RSS,* 2018.

[12] B. Tekin, S. N. Sinha, and P. Fua , "Real-time seamless single shot 6D object pose prediction," *CVPR,* p. , 2018.

[13] V.Lepetit, F.Moreno-Noguer,and P.Fua.E PnP, "An accurate O(n) solution to the PnP problem," *International Journal Computer Vision,* 2018.

[14] FANUC, "Vision functions for industrial robots," [Online]. Available: https://www.fanuc.eu/es/en/robots/accessories/robot-vision. [Accessed 12 July 2019].

[15] FANUC THE FACTORY AUTOMATION COMPANY, "iRVision - Fully integrated plug & play vision system - 2D, 2.5D, 3D, 3D Laser, 3D Area Sensor," 2017. [Online]. Available: https://www.fanuc.eu/uk/en/robots/accessories/robot-vision.

[16] Pick-it, "Bin picking with Pick-it," 2017. [Online]. Available: https://www.pickit3d.com/applications/bin-picking. [Accessed 15 July 2019].

[17] Open Source Robotics Foundation, "About ROS," 2016. [Online]. Available: http://www.ros.org/about-ros/. [Accessed 17 July 2019].

[18] Generation Robots, "Robot Operating System (ROS)," http://dx.doi.org/10.1007/978-3-319-54927-9, 2017. [Online]. Available: https://www.generationrobots.com/blog/en/2016/03/ros-robot-operating-system-2/. [Accessed 19 July 2019].

[19] Dave Coleman, "urdf - ROS Wiki," 2014. [Online]. Available: http://wiki.ros.org/urdf. [Accessed 19 July 2019].

[20] Rethink Robotics, "Rviz - sdk-wiki," 2015. [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Rviz. [Accessed 21 July 2019].

[21] Open Source Robotics Foundation, "ROS.org | Core Components," 2015. [Online]. Available: http://www.ros.org/core-components/.

[22] Dirk Thomas, Dorian Scholz, and Aaron Blasdel., "rqt - ROS Wiki," [Online]. Available: http://wiki.ros.org/rqt. [Accessed 21 July 2019].

[23] Matthew Robinson, "Industrial - ROS Wiki," 2018. [Online]. Available: http://wiki.ros.org/Industrial. [Accessed 22 July 2019].

[24] "Industrial/Install," ROS Wiki, 2017. [Online]. Available: http://wiki.ros.org/Industrial/Install. [Accessed 30 August 2019].

[25] Ioan A. Sucan and Sachin Chitta, "MoveIt! Motion Planning Framework," 2017. [Online]. Available: https://moveit.ros.org/. [Accessed 23 July 2019].

[26] GvdHoorn, "Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot - ROS Wiki," 2017. [Online]. Available: http://wiki.ros.org/Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot. [Accessed 24 July 2019].

[27] Isaac Saito, "Bags - ROS Wiki," 2015. [Online]. Available: http://wiki.ros.org/Bags. [Accessed 03 January 2019].

[28] liuxinwust, "Aubo_Robot," [Online]. Available: https://github.com/liuxinwust/aubo_robot. [Accessed 16 September 2019].

[29] Yannick Morvan, "Pinhole Camera Model". In: Computer Vision," 2014. [Online]. Available: http://www.epixea.com/research/multi-view-coding-thesisse8.html. [Accessed 20 August 2019].

[30] MATLAB, "What Is the Genetic Algorithm? - MATLAB & Simulink," [Online]. Available: https://www.mathworks.com/help/vision/ug/camera-calibration.html. [Accessed 25 August 2019].

[31] Alexander Reimann, "openni_launch/Tutorials/IntrinsicCalibration - ROS Wiki," 2015. [Online]. Available: http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration. [Accessed 2 September 2019].

[32] Adam Allevato, "camera_calibration/Tutorials/MonocularCalibration - ROS Wiki," 2017. [Online]. Available: http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration. [Accessed 4 September 2019].

[33] Jack Quin, "camera_info_manager_py - ROS Wiki," 2013. [Online]. Available: http://wiki.ros.org/camera_info_manager_py. [Accessed 6 September 2019].

[34] Fabien Spindler, "visp_hand2eye_calibration - ROS Wiki," 2016. [Online]. Available: http://wiki.ros.org/visp_hand2eye_calibration. [Accessed 10 October 2019].

[35] Jim Vaughan, "aruco_detect - ROS Wiki," 2018. [Online]. Available: http://wiki.ros.org/aruco_detect. [Accessed 14 October 2019].

[36] Saw Yer, "fiducials - ROS Wiki," 2018. [Online]. Available: http://wiki.ros.org/fiducials. [Accessed 20 October 2019].

[37] P.Marion,P.R.Florence,L.Manuelli,andR.Tedrake. LabelFusion, " Apipelineforgenerating ground truth labels for real RGBD data of cluttered scenes," *ICRA,* 2018.

[38] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and model set: Towards common benchmarks for manipulation research," *Intl. Conf. on Advanced Robotics (ICAR),* 2015.

[39] T. To, J. Tremblay, D. McKay, Y. Yamaguchi, K. Leung, A. Balanon, J. Cheng, and S. Birchfield, "NDDS: NVIDIA deep learning dataset synthesizer, 2018," 2018. [Online]. Available: https://github.com/NVIDIA/Dataset_Synthesizer. [Accessed 20 November 2019].

[40] M. Mueller, V. Casser, J. Lahoud, N. Smith, and B. Ghanem, "Sim4CV: A photo-realistic simulator for computer vision applications," *International Journal of Computer Vision,* pp. 1-18, 2018.

[41] Thang To and Jonathan Tremblay and Duncan McKay and Yukie Yamaguchi and Kirby Leung and Adrian Balanon and Jia Cheng and William Hodge and Stan Birchfield, *{NDDS}: {NVIDIA} Deep Learning Dataset Synthesizer,* 2018.

[42] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," *CVPR Workshop on Autonomous Driving (WAD),* 2018.

[43] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. L. Zitnick, "Microsoft COCO: Common objects in context," *CVPR,* 2014.

[44] J.Tremblay, T.To, and S.Birchfield, "Falling things: A synthetic dataset for 3D object detection and pose estimation," *CVPR Workshop on Real World Challenges and New Benchmarks for Deep Learning in Robotic Vision,* June 2018.

[45] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," *NIPS Autodiff Workshop,* 2017.

[46] D. P. Kingma and J. Ba. Adam, "A method for stochastic optimization," *ICLR,* 2015.

[47] Jonathan Tremblay and Thang To and Balakumar Sundaralingam and Yu Xiang and Dieter Fox and Stan Birchfield, "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects," in *tremblay2018corl:dope*, 2018.

[48] S.-E.Wei,V.Ramakrishna,T.Kanade,andY.Sheikh, "Convolutional pose machines," *InCVPR,* 2016.

[49] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2D pose estimation using part affinity fields," *CVPR,* 2017.

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR,* 2015.

[51] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again," *ICCV,* pp. 1521-1529, 2017.

[52] Avinash Sen, "github.com/avinashsen707/AUBOi5-D435-ROS-DOPE," GitHub, 20 March 2020. [Online]. Available: https://github.com/avinashsen707/AUBOi5-D435-ROS-DOPE.

[53] Avinash Sen, "avinashsen707.github.io," GitHub Pages, 20 March 2020. [Online]. Available: https://avinashsen707.github.io/.

APPENDICES

APPENDIX A

EXECUTION OF THE DEVELOPED BIN PICKING VISION SYSTEM

A.1 INITIAL SETUP

1. Turn on the robot's controller in AUTO mode, which is used to set the manipulator in continuous cycle;
2. Run the ROS server and press the cycle start button;
3. Power up the Realsense;
4. Connect to the switch on the Router, which makes the Ethernet connection with the robot's controller.

A.2 LAUNCH FILES

A.2.1 realsense_bag.launch

Launch file used to play a bag of a point cloud previous recorded. This was used with the purpose of writing the code for the segmentation of the camera data.

```
<launch>
            <arg name="camera"              default="camera"/>
            <arg name="tf_prefix"           default="$(arg camera)"/>
            <arg name="external_manager"    default="false"/>
            <arg name="manager"             default="realsense2_camera_manager"/>

          <!-- Camera device specific arguments -->
            <arg name="serial_no"           default=""/>
            <arg name="json_file_path"      default=""/>
            <arg name="fisheye_width"       default="640"/>
            <arg name="fisheye_height"      default="480"/>
            <arg name="enable_fisheye"      default="true"/>


            <arg name="depth_width"         default="640"/>
            <arg name="depth_height"        default="480"/>
            <arg name="enable_depth"        default="true"/>


            <arg name="infra_width"         default="640"/>
            <arg name="infra_height"        default="480"/>
            <arg name="enable_infra1"       default="true"/>
            <arg name="enable_infra2"       default="true"/>
```

```
    <arg name="color_width"          default="640"/>
    <arg name="color_height"         default="480"/>
    <arg name="enable_color"         default="true"/>


    <arg name="fisheye_fps"          default="30"/>
    <arg name="depth_fps"            default="30"/>
    <arg name="infra_fps"            default="30"/>
    <arg name="color_fps"            default="30"/>
    <arg name="gyro_fps"             default="400"/>
    <arg name="accel_fps"            default="250"/>
    <arg name="enable_gyro"          default="true"/>
    <arg name="enable_accel"         default="true"/>


    <arg name="enable_pointcloud"    default="false"/>
    <arg name="enable_sync"          default="true"/>
    <arg name="align_depth"          default="true"/>
    <arg name="filters"              default=""/>


    <!-- rgbd_launch specific arguments -->


    <!-- Arguments for remapping all device namespaces -->
    <arg name="rgb"                              default="color" />
    <arg name="ir"                               default="infra1" />
    <arg name="depth"                            default="depth" />
    <arg name="depth_registered_pub"             default="depth_registered"
/>
    <arg name="depth_registered"                 default="depth_registered"
unless="$(arg align_depth)" />
    <arg name="depth_registered"
default="aligned_depth_to_color" if="$(arg align_depth)" />
    <arg name="depth_registered_filtered"        default="$(arg
depth_registered)" />
    <arg name="projector"                        default="projector" />
    <!-- Disable bond topics by default -->
    <arg name="bond"                             default="false" />
    <arg name="respawn"                          default="$(arg bond)" />
    <!-- Processing Modules -->
    <arg name="rgb_processing"                   default="true"/>
    <arg name="debayer_processing"               default="false" />
    <arg name="ir_processing"                    default="false"/>
    <arg name="depth_processing"                 default="false"/>
    <arg name="depth_registered_processing"      default="true"/>
    <arg name="disparity_processing"             default="false"/>
    <arg name="disparity_registered_processing" default="false"/>
```

```
    <arg name="hw_registered_processing"         default="$(arg align_depth)"
/>
    <arg name="sw_registered_processing"         default="true" unless="$(arg
align_depth)" />
    <arg name="sw_registered_processing"         default="false" if="$(arg
align_depth)" />
    <group ns="$(arg camera)">


      <!-- Launch the camera device nodelet-->
      <include file="$(find
realsense2_camera)/launch/includes/nodelet.launch.xml">
        <arg name="external_manager"          value="$(arg
external_manager)"/>
        <arg name="manager"                   value="$(arg manager)"/>
        <arg name="tf_prefix"                 value="$(arg tf_prefix)"/>
        <arg name="serial_no"                 value="$(arg serial_no)"/>
        <arg name="json_file_path"            value="$(arg json_file_path)"/>


        <arg name="enable_pointcloud"         value="$(arg
enable_pointcloud)"/>
        <arg name="enable_sync"               value="$(arg enable_sync)"/>
        <arg name="align_depth"               value="$(arg align_depth)"/>


        <arg name="fisheye_width"             value="$(arg fisheye_width)"/>
        <arg name="fisheye_height"            value="$(arg fisheye_height)"/>
        <arg name="enable_fisheye"            value="$(arg enable_fisheye)"/>


        <arg name="depth_width"               value="$(arg depth_width)"/>
        <arg name="depth_height"              value="$(arg depth_height)"/>
        <arg name="enable_depth"              value="$(arg enable_depth)"/>


        <arg name="color_width"               value="$(arg color_width)"/>
        <arg name="color_height"              value="$(arg color_height)"/>
        <arg name="enable_color"              value="$(arg enable_color)"/>


        <arg name="infra_width"               value="$(arg infra_width)"/>
        <arg name="infra_height"              value="$(arg infra_height)"/>
        <arg name="enable_infra1"             value="$(arg enable_infra1)"/>
        <arg name="enable_infra2"             value="$(arg enable_infra2)"/>


        <arg name="fisheye_fps"               value="$(arg fisheye_fps)"/>
```

```
            <arg name="depth_fps"               value="$(arg depth_fps)"/>
            <arg name="infra_fps"               value="$(arg infra_fps)"/>
            <arg name="color_fps"               value="$(arg color_fps)"/>
            <arg name="gyro_fps"                value="$(arg gyro_fps)"/>
            <arg name="accel_fps"               value="$(arg accel_fps)"/>
            <arg name="enable_gyro"             value="$(arg enable_gyro)"/>
            <arg name="enable_accel"            value="$(arg enable_accel)"/>
            <arg name="filters"                 value="$(arg filters)"/>
        </include>
        <!-- RGB processing -->
        <include if="$(arg rgb_processing)"
                 file="$(find rgbd_launch)/launch/includes/rgb.launch.xml">
            <arg name="manager"                     value="$(arg manager)" />
            <arg name="respawn"                     value="$(arg respawn)" />
            <arg name="rgb"                         value="$(arg rgb)" />
            <arg name="debayer_processing"          value="$(arg
debayer_processing)" />
        </include>


        <group if="$(eval depth_registered_processing and
sw_registered_processing)">
          <node pkg="nodelet" type="nodelet" name="register_depth"
                args="load depth_image_proc/register $(arg manager) $(arg
bond)" respawn="$(arg respawn)">
            <remap from="rgb/camera_info"           to="$(arg
rgb)/camera_info" />
            <remap from="depth/camera_info"         to="$(arg
depth)/camera_info" />
            <remap from="depth/image_rect"          to="$(arg
depth)/image_rect_raw" />
            <remap from="depth_registered/image_rect" to="$(arg
depth_registered)/sw_registered/image_rect_raw" />
          </node>


          <!-- Publish registered XYZRGB point cloud with software registered
input -->
          <node pkg="nodelet" type="nodelet"
name="points_xyzrgb_sw_registered"
                args="load depth_image_proc/point_cloud_xyzrgb $(arg manager)
$(arg bond)" respawn="$(arg respawn)">
            <remap from="rgb/image_rect_color"      to="$(arg
rgb)/image_rect_color" />
            <remap from="rgb/camera_info"           to="$(arg
rgb)/camera_info" />
            <remap from="depth_registered/image_rect" to="$(arg
depth_registered_filtered)/sw_registered/image_rect_raw" />
```

```
            <remap from="depth_registered/points"     to="$(arg
depth_registered)/points" />
          </node>
        </group>
        <group if="$(eval depth_registered_processing and
hw_registered_processing)">
          <!-- Publish registered XYZRGB point cloud with hardware registered
input (ROS Realsense depth alignment) -->
          <node pkg="nodelet" type="nodelet"
name="points_xyzrgb_hw_registered"
                args="load depth_image_proc/point_cloud_xyzrgb $(arg manager)
$(arg bond)" respawn="$(arg respawn)">
            <remap from="rgb/image_rect_color"        to="$(arg
rgb)/image_rect_color" />
            <remap from="rgb/camera_info"             to="$(arg
rgb)/camera_info" />
            <remap from="depth_registered/image_rect" to="$(arg
depth_registered)/image_raw" />
            <remap from="depth_registered/points"     to="$(arg
depth_registered_pub)/points" />
          </node>
        </group>
      </group>
    </launch>
```

A.2.2 dope.launch

This is the official DOPE ROS package for detection and 6-DoF pose estimation of known objects from an RGB camera. The network has been trained on the following YCB objects cracker box.

```
<launch>
      <arg name="config" default="$(find dope)/config/config_pose.yaml"
doc="Path to parameter config file"/>


      <node name="dope" pkg="dope" type="dope" output="screen"
clear_params="true">
        <rosparam file="$(arg config)"/>
      </node>
    </launch>
```

APPENDIX B

LIST OF PUBLICATIONS

1. **Avinash Sen, Aju S S, Dr. Lalu P P, Dr. Sudeesh R S** (2020) *'Detection and Pose Estimated Grasping in Industrial Robot for Bin Picking Operations'*. Published in the proceedings of the International Colloquium organized by Mar Athanasius College of Engineering in association with The Institution of Engineers (India) Kochi Local Centre.

IC@MACE-2020

INTERNATIONAL COLLOQUIUM ON RECENT TRENDS IN ENGINEERING

MAR ATHANASIUS COLLEGE OF ENGINEERING, KOTHAMANGALAM, KERALA, INDIA

13-14 March 2020

CERTIFICATE OF APPRECIATION

This is to certify that the paper entitled

Detection and Pose Estimated Grasping in Industrial Robot for Bin Picking Operations

authored by

Avinash Sen, Aju S S, Dr. Lalu P P, Sudeesh R S

is published in the proceedings of the International Colloquium organized by Mar Athanasius College of Engineering in association with The Institution of Engineers (India) Kochi Local Centre.

Dr. Kurian John
Secretary, IC@MACE

Dr. Brijesh Paul
Chairman, IC@MACE

Dr. Mathew K.
Principal, MACE

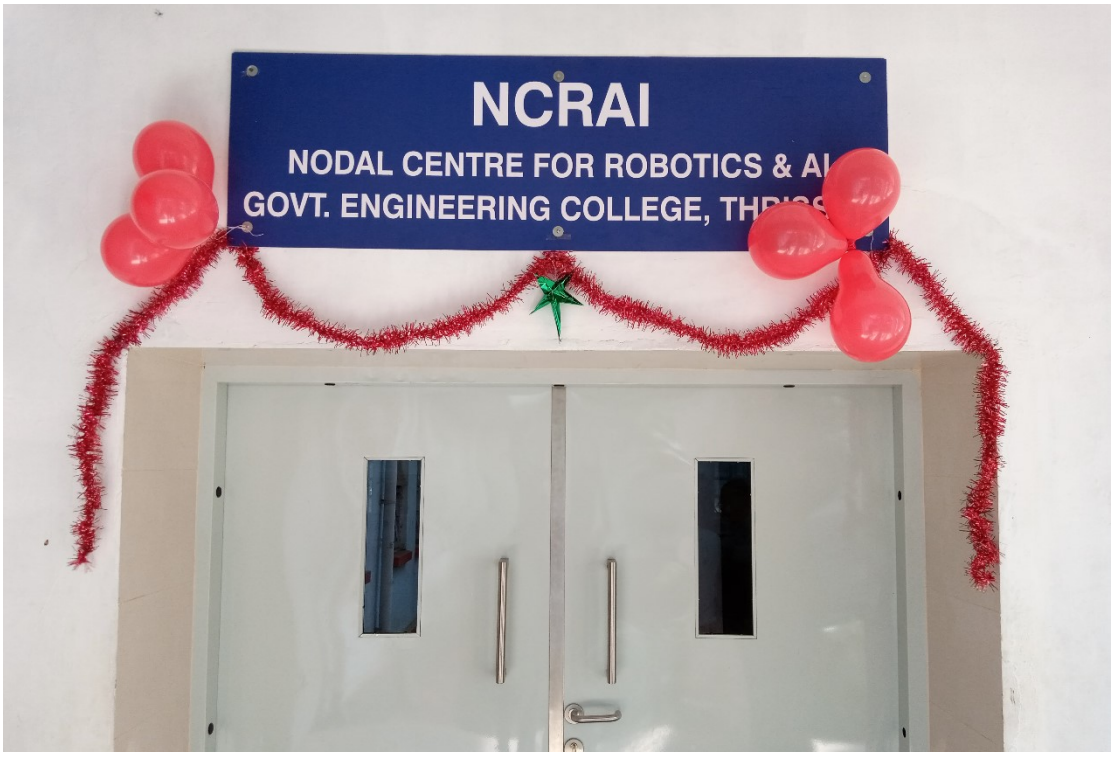
APPENDIX C

ENDORSEMENTS

C.1 NODAL CENTRE FOR ROBOTICS AND ARTIFICIAL RESEARCH (NCRAI)

This work has been supported by **Nodal Centre for Robotics and Artificial Research (NCRAI) of** *Government Engineering College, Thrissur*. Project development, subsequent testing and Evaluations are held on this laboratory.

Nodal Center for Robotics and Artificial Intelligence is being established at Government Engineering College Thrissur with two fold objective of 1) Providing facilities for frontline research in Robotics and AI 2) Organizing and Training and skill development programs for Technical Institutes in the field of Robotics and AI. The facility is envisaged to be shared by various Colleges, Polytechnics and Technical High Schools under Department of Technical education Kerala and function as center of excellence in Robotics and AI. The center was started functioning in June 2019 is funded by Higher Education Department, Government of Kerala.



C.2 ROBOTICS AND AUTOMATION SOCIETY OF IEEE SB GEC

Received First Prize for the *Technical Paper Competition* organized by **Robotics and Automation Society of IEEE SB GEC** Thrissur.

IEEE

IEEE SB GECT
GOVERNMENT ENGINEERING COLLEGE THRISSUR

IEEE Robotics & Automation Society

GOVERNMENT ENGINEERING COLLEGE THRISSUR

CERTIFICATE

OF EXCELLENCE

This is to certify that

Avinash Sen

of Government Engineering College Thrissur has secured first position in Roboto - technical paper competition for his paper "DETECTION AND POSE ESTIMATED GRASPING IN INDUSTRIAL ROBOT FOR BIN PICKING OPERATIONS" conducted by RAS chapter IEEE SB GEC Thrissur

Prof. Joseph KD
Student Branch Counsellor
IEEE SB GECT

Vishnu S
Chairman
IEEE SB GECT