Autonomous Grabbing with RGB-D

Integrated into the BrikieBot Project- A New Concept of a Robotic Bricklayer

Manar MAHMALJI

Master thesis submitted under the supervision of
ir. Michele AMBROSINO & dr. Emanuele GARONE

The co-supervision of
dr. Mehrdad TERATANI

In order to be awarded the Master's Degree in
electro-mechanical engineering- Module Robotics and Mechatronics - Construction

Academic year
2021-2022

ULB UNIVERSITÉ LIBRE DE BRUXELLES

ULB UNIVERSITÉ LIBRE DE BRUXELLES

Exemplaire à apposer sur le mémoire ou travail de fin d'études,
au verso de la première page de couverture.

Réservé au secrétariat :	**Mémoire réussi***	**OUI**
	NON	

CONSULTATION DU MEMOIRE/TRAVAIL DE FIN D'ETUDES

Je soussigné

NOM : MAHMALJI………………………………

PRENOM : MANAR………………………………

TITRE du travail : Autonomous Grabbing with RGB-D Integrated into the BrikieBot Project- A new concept of a Robotic Bricklayer………………………………

AUTORISE*

~~**REFUSE***~~

la consultation du présent mémoire/travail de fin d'études par les utilisateurs des bibliothèques de l'Université libre de Bruxelles.

Si la consultation est autorisée, le soussigné concède par la présente à l'Université libre de Bruxelles, pour toute la durée légale de protection de l'œuvre, une licence gratuite et non exclusive de reproduction et de communication au public de son œuvre précisée ci-dessus, sur supports graphiques ou électroniques, afin d'en permettre la consultation par les utilisateurs des bibliothèques de l'ULB et d'autres institutions dans les limites du prêt inter-bibliothèques.

Fait en deux exemplaires, **Bruxelles, le** 3-06-2022

Signature

Manar M.

Formulaire version 04/2015

** Biffer la mention inutile*

** Biffer la mention inutile*

ULB UNIVERSITÉ LIBRE DE BRUXELLES

ULB UNIVERSITÉ LIBRE DE BRUXELLES

Exemplaire destiné à l'étudiant.

Réservé au secrétariat : Mémoire réussi* OUI NON

CONSULTATION DU MEMOIRE/TRAVAIL DE FIN D'ETUDES

Je soussigné

NOM : MAHMALJI

PRENOM : MANAR

TITRE du travail : Autonomous Grabbing with RGB-D Integrated into the BrikieBot Project- A new concept of a Robotic Bricklayer

AUTORISE*

~~REFUSE*~~

la consultation du présent mémoire/travail de fin d'études par les utilisateurs des bibliothèques de l'Université libre de Bruxelles.

Si la consultation est autorisée, le soussigné concède par la présente à l'Université libre de Bruxelles, pour toute la durée légale de protection de l'œuvre, une licence gratuite et non exclusive de reproduction et de communication au public de son œuvre précisée ci-dessus, sur supports graphiques ou électroniques, afin d'en permettre la consultation par les utilisateurs des bibliothèques de l'ULB et d'autres institutions dans les limites du prêt inter-bibliothèques.

Fait en deux exemplaires, **Bruxelles, le** 3-06-2022

Signature Manar M.

** Biffer la mention inutile*

** Biffer la mention inutile*

Formulaire version 04/2015

Acknowledgments

I would like first to express my utmost appreciation for Michele Ambrosino for his unwavering support throughout all the academic year. Not only is he a great engineer to work with, but also a great person to know. Furthermore, I wish to extend my gratitude to the promoters: professors Emanuele Garone and Mehrdad Teratani for accepting me into this project, and for their valuable feedback all the time. I would also like to thank all the SAAS laboratory staff for providing different kinds of technical support.

Contents

List of Figures

Abstract

When robots can see, measure, analyze, and respond to their environment, a whole new world of possibilities opens up. To bring these possibilities to life, the robotics industry is increasingly adopting vision-guided robotics. In the construction business, there have been many prototypes to integrate robotic bricklayers into the industry. However, the majority of robots are somewhat restricted to specified conditions, which is primarily because they lack the perception of their environment. Integrating computer vision into a robotic bricklayer allows it to be more aware of its environment on the worksite and, thus, adds more flexibility to the achievable tasks. This thesis proposes a vision-guided approach, provided by an RGB-D camera and ArUco markers, that enables a robotic arm to autonomously perform the grabbing of a suspended block. A GitHub repository of a set of organizational tools developed for eye-to-hand calibration with OpenCV in Python is published: link. Furthermore, the results of this thesis will contribute to a conference paper that will be submitted to the IEEE International Conference on Robotics and Automation (ICRA) 2023. This thesis is part of the BrikieBot project, which lays the foundations for a new robotic bricklayer.

Keywords: *Vision-guided robotics, RGB-D Camera, ArUco Pose estimation, Eye-to-hand calibration, Robotic Bricklayer*

Chapter 1

Introduction

In the demanding workflow of robotics in everyday industries, being resistant towards exhaustive repetition, chemical effects and other environmental hazards, or higher possible precision during maneuvering is only a fraction of all the leverage a robot may offer[7]. Robots can indeed become more intelligent by perceiving their environments and interacting with it[9]. And endowing them with such an ability has been a long-standing goal in computer vision[5]. Computer vision systems provide the robots with the ability to understand the surrounding environment, to detect objects and to classify them[12]. When robots can see, measure, analyze, and respond to their environment, a whole new world of possibilities opens up. To bring these possibilities to life, the robotics industry is increasingly adopting computer vision technology, which allows robots to sense depth, navigate landscapes, and recognize objects, people, and scenes[13].

In the construction industry, several robotic brick-laying platforms have been proposed in the course of the recent years [1]. Research on construction robotics started during the "robot boom" of the 1970s with the development of simple robotic systems that could help workers in very specific construction tasks (e.g. material manipulators for placing heavy components, or tele-operated and autonomous machines [20]). However, very few of the prototypes developed in the 70s, 80s, 90s, and early 2000s have reached the market and can be considered successes, because most of them adopt the classical assumption of 'rigid robot' which results in large weight of the robots w.r.t.[1] the loads that they are able to manipulate and therefore most of them did not pass the prototype stage [8].

[1]With respect to

1.1 Scope of Study

The BrikieBot project proposes a new robotic concept for the bricklaying of large blocks based on a "non-rigid" robot, such as a crane, in charge of the macro-movement and of holding most of the weight of the building block, and a small rigid robot mounted on an aerial work platform to achieve the desired precision during the fine placement of the block. A schematic of the envisioned solution is reported in Fig. 1.1 where (1) is the crane, (2) the aerial work platform, (3) is the robotic arm, (4) is the block to be placed, and (5) is the existing wall.

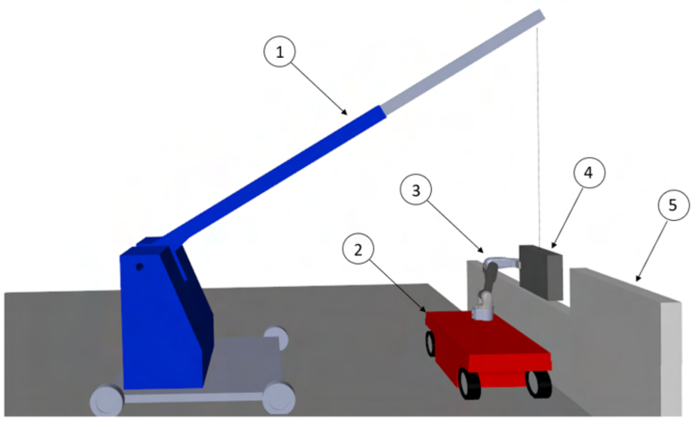


Figure 1.1: BrikieBot: Proposed Scheme for the Robotic Bricklayer, where (1) is the crane, (2) the aerial work platform, (3) is the robotic arm, (4) is the block to be placed, and (5) is the existing wall

This solution can be seen both as a way to mimic the role of masons in a current construction process (see Fig.(1.2), where a flexible arm is in charge of taking most of the weight of the load, while a rigid arm is mounted in parallel and is used to measure the exact position of the block and to guide its fine positioning.

The laying activity can be divided into three sub-tasks:

1. to move the block near the desired position with the crane;
2. to grab the hanging block with the robotic manipulator;
3. to control cooperatively the robot and the crane to carry out the fine-positioning on the wall.

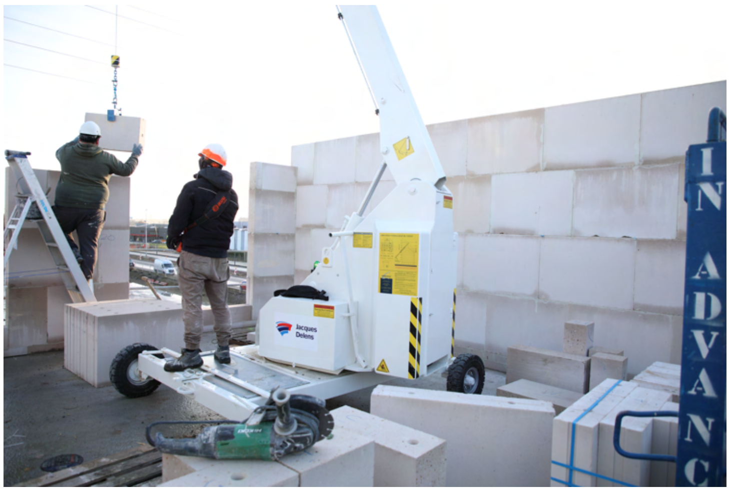


Figure 1.2: Real construction process BrikieBot aims to mimic, source: *Jacques Delens, BESIX Group*

This study is not intended to explain the different challenges raised in every sub-task, as they are well-discussed in [10]. Rather, it answers to the challenges of task(2) by proposing an implementation of computer vision methods with an RGB-D camera to enable a robotic arm to autonomously perform the grabbing of a suspended block.

1.2 Outline

1. *Introduction*

   After going through a general introduction, and defining the scope of the study, a list of state-of-the-art technologies for vision-guided robotics is presented. This is followed by a statement of the study's goal.

2. *Used Equipment*

   This chapter introduces the different hardware/software used throughout the study, including the RGB-D camera and the robotic arm.

3. *Object Detection and Pose Estimation*

   This chapter discusses the proposed method for a block's detection and its pose estimation using ArUco markers.

4. *Eye to Hand Calibration*

   This chapter shows the setup, procedure and validation for the eye-to-hand calibration, along with the different challenges encountered through implementation.

5. *Grabbing Experiments*

   This chapter shows the results of grabbing experiments performed on a suspended block, as well as the proposed criterion for deciding if a block is static or oscillating.

6. *Pose Tracking*

   This chapter shows the results of tracking the grabbing pose of a moving block.

7. *Future Work and Conclusion*

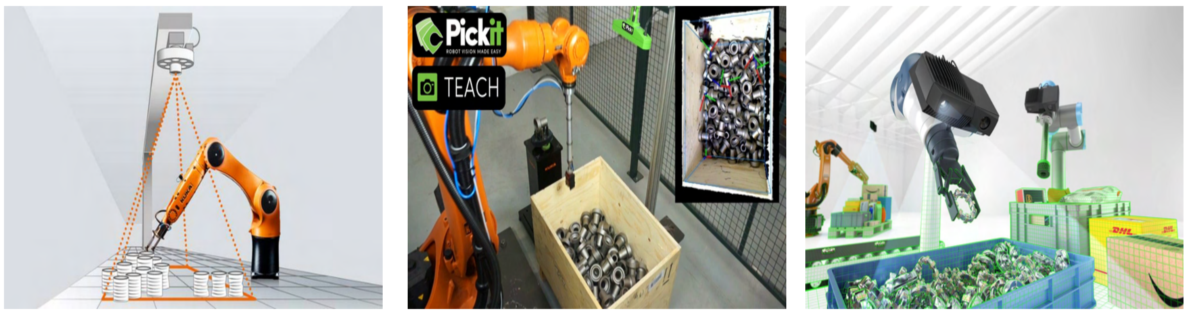
   This chapter discusses potential future aspects of the achieved work and concludes the thesis with a summary of its contribution and learning outcomes.

1.3 State-of-the-Art

The technologies involving vision-guided robotics are numerous; nonetheless, this section aims to give a sample of some achieved work both in the commercial and research stages.

1.3.1 Commercial Products

Starting by what is already available on the market, it is a common application of computer vision in industrial processes to be used in the inspection of parts before being picked by an industrial robot for the next stage of processing. In this case, the camera is fixed at some point and a robot is ready to pick an item on a planar surface, as seen in Fig.(1.3a). Normally, the inspections could be dimensional measurements, unusual colours, defect parts, counting a certain number of items, or even read a code from parts...etc. A world leader in these systems is **Cognex Corporation**, an American manufacturer of machine vision systems, software and sensors used in automated manufacturing. Cognex offers a complete family of vision products—from standalone vision systems to 2D and 3D vision software—to give a robot the power of sight. A typical vision system, as seen in Fig.(1.3a), is an industrial KUKA robot integrated with Cognex VisionPro software. The powerful vision tools locate, inspect and read codes on stationary or moving parts [3].



(a) (b) (c)

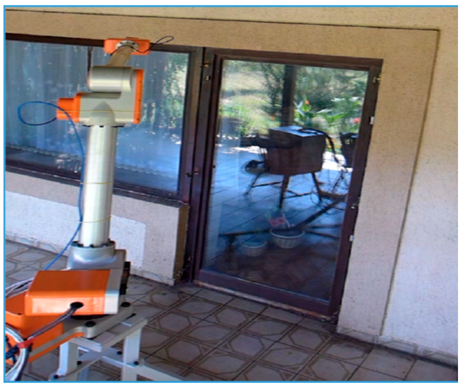
Figure 1.3: Different Commerical Products of Vision-Guided Robotics. (a) KUKA Visiontech with Cognex VisionPro [3], (b) Pickit L 3D camera, bin picking[15], (c) Zivid Two 3D camera, bin picking[24]

Going further, the Belgian startup Pick-it™, founded in 2012, has taken the quest to make robot vision easy by developing a product of a 3D camera and accompanying software that guides the robot to see, pick and place a wide range of products, not necessarily on a flat place as seen in Fig.(1.3a). The product is suited for various service applications: machine tending, bin picking, order picking, palletizing, kitting and assembly[15]. An example of a bin picking process can be seen in Fig.(1.3b), in which the camera is fixed, and it is able to determine the poses of each of the items in the bin(seen on the top right of the figure) and communicate them to the robot to pick an item. Standard features of the Pick-it product include easy bin collision avoidance and tool modeling. The pick frame can be chosen to adjust the gripper geometry[15]. More importantly, Pick-it™ TEACH makes it possible to define the product model simply by taking a photo with the Pick-it™ camera[15]. Which makes it flexible and easy to use for several items during the same working day with seamless effort.

On the same wavelength as Pick-it is Zivid. The Norwegian startup, founded in 2015, is also making 3D cameras for similar service applications, but with a very high resolution and extreme precision native color 3D point clouds with minimal occlusion, and excellent noise suppression, significantly improving object recognition and increasing the number of detectable parts[24].In Fig.(1.3c), it can be seen that the camera is mounted on the robot, which gives it a lot more flexibility than what was seen in Fig.(1.3b). However, it is worth noting that Pick-it can also mount the camera on the robot. In fact, the two companies have recently started to collaborate to combine the longer expertise of Pick-it and the more accurate cameras of Zivid[24].

1.3.2 Research Projects

[19] describes a robust and simple computer vision algorithm to detect and extract windows and coarse obstacles on walls using depth images recorded with a 3D camera. The goal is to allow a robot to paint a wall on its own while evading the obstacles like windows, paintings...etc.The paint spraying unit is a 5DOF robotic arm and the 3D camera is an Intel Realsense D435 stereo camera. The scheme is shown in Fig.(1.4a). The project deals with plane detection and segmentation, a famous problem in computer vision, using a random sample consensus model (RANSAC), which is an iterative method for estimating a mathematical model from a data set that contains outliers[19]. In this case, a wall is a huge set of points in the same plane, and RANSAC iteratively classifies all the points away from the wall by a certain threshold as outliers. The final result of the extracted wall is shown in Fig.(1.4b). However, the main disadvantage of this algorithm is that there is no upper limit on the number of iterations it takes to find the plane, and a higher number of iterations increases the probability of a reasonable model being produced is increased. Hence, there is a trade-off in performance. Furthermore, the algorithm assumes the wall is the greatest planar object in the image and the obstacles are always parallel to the wall.


(a)

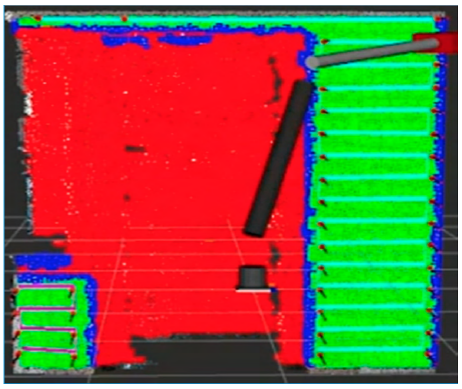

(b)

Figure 1.4: (a) 5DOF robotic arm using a D435 stereo camera to detect and extract a wall with obstacles using a model-based method, RANSAC. The robot is then able to paint the extracted wall (b) Extracted wall from RANSAC. The area in red are obstacles and the area in green is to be painted

A similar work is done in [2] in terms of plane segmentation. [2] develops a pipeline for extracting horizontal contact surfaces from point-cloud data, also from an Intel Realsense D435 stereo camera, for autonomous placing of rigid objects. It basically indicates a horizontal surface for a robot to place a certain load in a roller-container, as shown in Fig.(1.5a), where the camera is fixed on top of the container. The proposed pipeline uses a region-growing algorithm, tailored for horizontal surfaces extraction, to determine good surfaces for placing, as shown in Fig.(1.5b). A a region-growing clustering algorithm means that a neighborhood of point is admitted to a cluster if they fulfill some proximity condition[2]. Finally, it compares the results with an off-the-shelf RANSAC implementation.

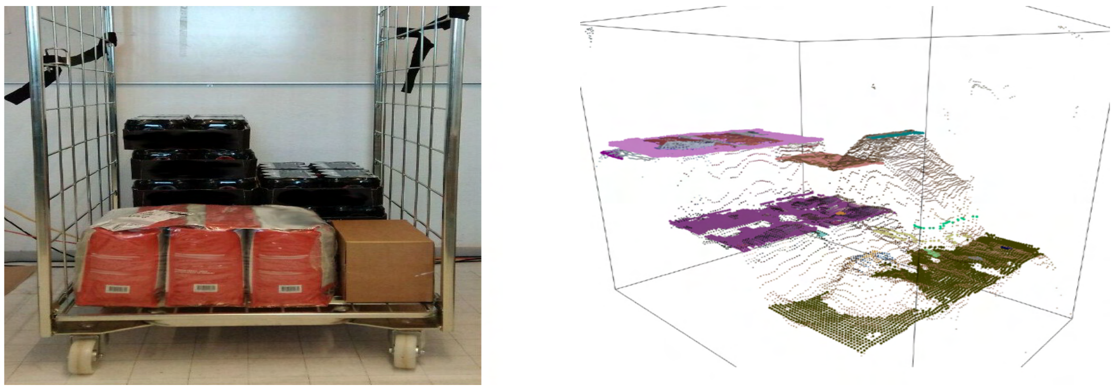
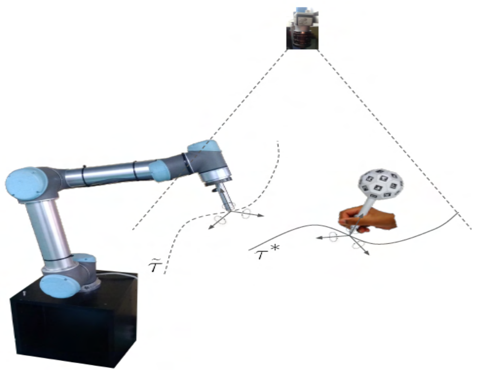

(a) (b)

Figure 1.5: (a) A partially filled roller-container with common goods (b) The developed pipeline uses a region-growing algorithm to compute a new placement for a new load, where good surfaces are clustered in different colours.

[9] is a very interesting study which describes a method to allow for a more natural human-robot interaction by developing a trajectory vision-based teaching system with a vision-based positioning, called Solpen. It generates pose paths of six degrees of freedom (6-DoF) for robotics applications such as welding, cutting, painting, or polishing. The system is consists of fixed 2D camera, a 5 DOF robotic arm and the printed ArUco markers which are hand-glued on 31 surfaces of the designed 3D-printed Solpen. The described scheme is shown in Fig.(1.6). ArUco markers are very common in the world of vision-guided robotics, and they will be more explained in chapter 3. Within a 1-meter working range, the average errors of for 3 different trajectories with *Solpen* are 2.79 mm, 1.09 mm, and 1.44 mm for X, Y, and Z axis, respectively. For rotation, the error is slightly higher than 0.1 deg.



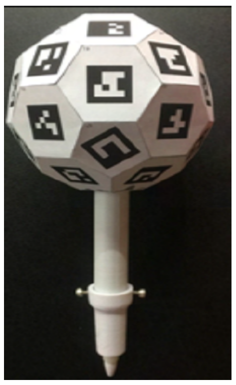
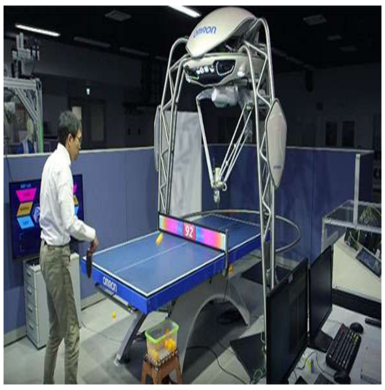
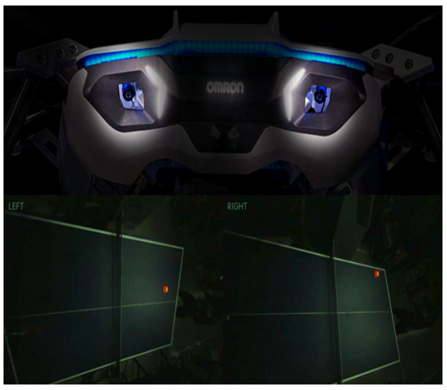


Figure 1.6: A robot trajectory teaching system with a vision-based positioning pen is developed to generate pose paths of six degrees of freedom (6-DoF) for vision-guided robotics applications such as welding, cutting, painting, or polishing...etc

Speaking of teaching, it is not uncommon to hear about humans teaching robots, but not the other way around. Developed by *Omron* Corporation, *FORPHEUS*, seen in Fig.(1.7a), holds the Guinness record for the first table tennis robot teacher[17]. The robot's sensing module is composed of a high-speed stereo camera, seen in Fig.(1.7b), that measures the ball's movement at 80 frames per second, two motion sensors for tracking the player's movement, and another high-speed camera for tracking the racket's movement[14]. Furthermore, the robot has a 6-axis arm, seen in Fig.(1.7c), enabling it to perform a wide combination of skillful strokes in the way that a human player does[14]. Using AI, FORPHEUS can then predict the trajectory of the ball and hit the ball back, and learns about the player's characteristics over time to help him make progress[17]. Unfortunately, it is not yet available for sale, and is mainly presented in technology expos[17].


(a)



(b)

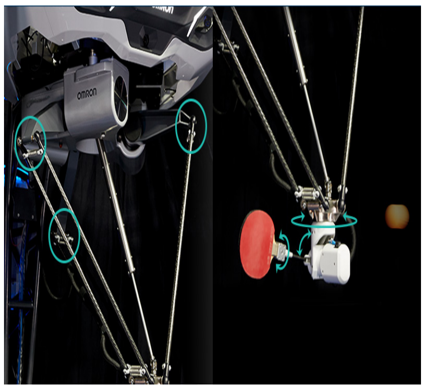

(c)

Figure 1.7: (a) FORPHEUS: First Table Tennis Robot Tutor developed by *Omron* (b) High-speed stereo cameras that tRack the ball's motion (c) 6-axis robotic arm that allows skillful strokes. Source: *Omron*

For what concerns robotic bricklayers, some of them use 3D-scanning technologies to navigate through their environment and perform the picking and placing of bricks. A good example is the *In-situ Fabricator*, shown in Fig.(1.8). It is currently still under development at The National Centre of Competence in Research (NCCR) Digital Fabrication[11]. The robot's arm is equipped with an ABB IRB 4600 robotic arm with 2.55 m reach and 40 kg payload. IF can achieve a maximum driving speed of 5 km/h on non-flat or soft grounds[11]. It is also equipped with a laser range finder that generates a 3D map of its surroundings every time the robot sweeps its arm. This map is compared to previous scans in order to calculate the robot's relative position. This scanning feedback loop is also used to compare the existing state of construction to the CAD model of the desired structure, and the control system of IF is able to determine any small deviations from design and make adaptions within the construction process [11].

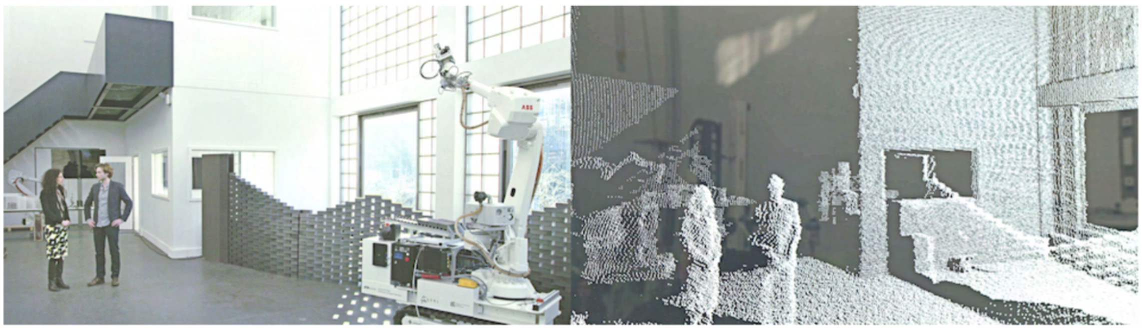


Figure 1.8: The In-situ Fabricator robot's arm is equipped with a laser range finder that generates a 3D map of its surroundings. The map enables the robot to localize itself as well as verify the construction with its CAD [11]

1.4 Goal Statement

This study aims to use an RGB-D camera to enable a robotic arm to perform the grabbing of a suspended block. The workflow can be divided into the following tasks:

1. Choose the RGB-D camera and be familiarized with its available functions
2. Find a proper way to identify the pose of a block with respect to the camera frame
3. Get and validate the transformation matrix from camera frame to robot base frame(Eye-to-hand calibration), and develop an interface for making this process easily repeatable.
4. Integrate the camera data into the ROS pipeline of the robotic arm

5. Test the pipeline for static and dynamic grabbing of a suspended block

6. Propose an approach for dynamic tracking of a moving block

Chapter 2

Used Equipment

This chapter introduces the different hardware/software used throughout the study, including the RGB-D camera and the robotic arm.

2.1 RGB-D Camera

2.1.1 RGB-D Principle

A normal image taken from a standard camera is a 2D grid of pixels where each pixel is a set of 3 values, which are usually thought of as Red, Green and Blue, or RGB. Each value ranges from 0 to 255, so black, for example, is (0,0,0) and a pure bright red would be (255,0,0). On the other hand, a depth camera outputs pixels with only one numerical value associated with them, that is the distance from the camera or the "depth". Some depth cameras have both an RGB and a depth system, which can give pixels with all four values, or RGBD, and hence comes the name RGB-D camera. In order to understand which RGB-D camera to choose, it is imperative to understand the methods for depth perception, as each has its own advantages and disadvantages depending on the application. According to [16], there exist three main types of methods:

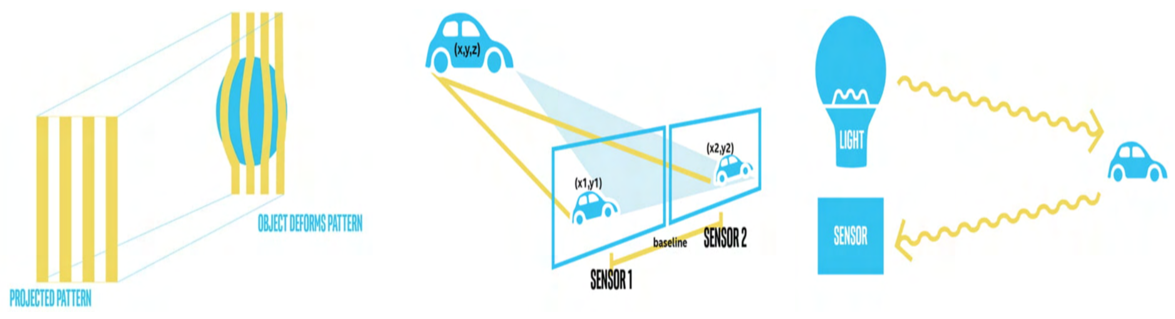
1. **Structured Light and Coded Light**
   Coded light and structured light depth cameras use similar technologies. They depend on projecting light from some kind of emitter, most often an infrared one, onto the scene. The projected(emitted) light is patterned, either visually or over time, or a combination of both. Patterns deform in a certain way on different geometries and hence, the camera sensor estimates depth by comparing the projected pattern to the

deformed one[16]. Using the disparity between an expected image and the actual image viewed by the camera, distance from the camera can be calculated for every pixel[4]. An example is shown in Fig.(2.1a). If the projected pattern is a series of stripes projected onto a ball, the stripes would deform and bend around the surface of the ball in a specific way. The pattern also changes if the ball moves closer to or away from the emitter. Because this technology depends chiefly on how accurate projected pattern of light is seen, coded and structured light cameras do best indoors at relatively short ranges (depending on the power of the light emitted from the camera) [16]. Another issue with systems like this is that they are vulnerable to other noise in the environment from other cameras or devices emitting infrared[4]. Ideal uses for coded light cameras are things like gesture recognition or background segmentation (also known as virtual green screen)[16].

2. **Stereo Depth**
   In the same way the human brain perceives depth, stereo cameras use two eyes(two cameras) for depth perception. The brain compares what every eye sees to realize how far an object is, and that is why objects closer to us will appear to move significantly from eye to eye, where an object in the far distance would appear to move very little[16]. As seen in Fig.(2.1b), stereo depth cameras have two sensors, spaced a small distance apart, and a processor computes the disparity between the images of the two sensors and as the sensor-to-sensor distance (baseline) is known, depth is calculated[4]. Unlike coded or structured light cameras, stereo cameras can use any light to measure depth, which enables them to work well in most lighting conditions, including outdoors[16]. Another benefit of stereo cameras is that they don't interfere with each other in the same way that a coded light or time of flight camera would, and hence there is no limit to how many you can use in a particular space[4]. Furthermore, the distance these cameras can measure is directly related to how far apart the two sensors are – the wider the baseline is, the further the camera can see[16]. On the other side, one disadvantage of stereo depth is that it relies on finding correspondences between two images to extract depth, which is easier if the scene in the field of view as a highly varied texture[2]. This means degraded performance for a low-textured image. For example, walls or floors.

3. **Time of Flight and LiDAR**

   Every type of depth cameras depends on a known information in order to calculate depth. In coded light and structured light, the projected pattern of light is known. In stereo, the baseline is known. In case of time of flight, the speed of light is the known variable used to calculate depth. As shown in Fig.(2.1c), in any time of flight device, there is some kind of light emitter that is swept over a scene, and then based on how long it takes light to get back to a sensor on the camera, it calculates depth[4]. LiDAR sensors are a type of time of flight camera which use laser light to calculate depth[16]. Depending on the power and wavelength of the light, time of flight sensors can measure accurate depth at significant distances – for example, being used to map terrains from a helicopter[16]. The main drawback of time of flight cameras is that they can, similar to structured light and coded light cameras, be vulnerable to other cameras, or other sources of light like sunlight, in the same space and can also perform less well in outdoor environments[4]. Furthermore, they are more expensive than the previously mentioned types.



(a) (b) (c)

Figure 2.1: Different depth perception methods(a)Structured or Coded Light(b) Stereo Depth (c) Time of Flight and LiDAR, Source: *Intel Realsense*

The following table summarizes the advantages and disadvantages for each type of depth camera:

	Advantages	**Disadvantages**
Structured Light and Coded Light	suitable for low range and indoor use	influenced by other cameras or devices emitting infrared
Stereo Depth	work well in most lighting conditions, can be long range	less performance for low-textured images and dark environments
Time of Flight and LiDAR	very accurate , could be very long range applications. Can work in darkness	influenced by other cameras or other sources of light , expensive

Table 2.1: A summary of the advantages and disadvantages of the different depth cameras

2.1.2 Camera Selection

The first criterion for camera selection is that it should be able to operate outdoors, as in a construction site. The second is that it should have a wild field of view (FOV) so that it can detect as much information as possible from one place. And finally, the working range is 1-5 meters. The first criteria immediately rules out the structured light and coded light, and time of flight cameras as they are both not suited for outdoor use. For what concerns stereo cameras, this study does not aim to do a comprehensive study of the all the stereo cameras in the market, rather choose the best off-the-shelf product suited for vision-guided robotics. To the best of the author's knowledge, the most common stereo cameras are the ZED series by Stereolabs and D400 series by Intel Realsense. A comparison of the most important specs between the latest models of the two series, ZED2i and D455, is shown in Table 2.2. ZED2i has more than triple the range of D455, a better RGB and depth resolution and is equipped with more sensors. It is more suited for 3D mapping of its environment and for AI applications like skeleton tracking thanks to the use of neural networks. Furthermore, it has an IP66 protection rating for water and dust resistance, which D455 does not have. However, there is not much difference in accuracy under 6 m and D455 has an IR projector. As explained in the previous section, the key feature that enables the stereo camera to retrieve the depth of each pixel is finding correspondences between the two pictures, which is problematic for poorly textured surfaces. To overcome this, the IR projector projects random light pattern on the scene, enriching it with more identifiable key-points that help to find correspondences in the poorly textured surfaces. This feature could be helpful for a low-textured object like the block shown in Fig.(1.2). Last but not least, according to the different projects seen in vision -guided robotics whilst writing the state-of-the-art, the lower cost of the D455 made it a more favorable choice for a lot of research projects allowing for more online material available for it than for the ZED series in general. In the end, although the ZED2i is more suitable and offers better performance, the D455 was chosen because of the great shipping delay of ZED2i.

Feature		**D455**	**ZED2i 2.1 mm**[1]
Use Environment		Indoor & Outdoor	Indoor & Outdoor Has an IP66 rating for water and dust resistance
Baseline		9.5 cm	12 cm
Depth	**Technology**	Active IR[2] stereo	Neural stereo depth sensing
	Resolution	Up to 1280 x 720	Up to 2208 x 1242
	FPS[3]	Up to 90 fps	Up to 100 fps
	FoV (H x V)[4]	87° x 58°	110°x 70°
	Range	.6 m to 6 m	.2 m to 20 m
	Accuracy	<2% at 4 m	<1% at 3 m <5% at 15 m
RGB	**Resolution**	Up to 1280 x 800	Up to 2208 x 1242
	FPS	Up to 30 fps	Up to 100 fps
	FoV (H × V)	90 × 65°	110° x 70°
Sensors		Accelerometer, gyroscope	Accelerometer, gyroscope magnetometer, barometer temperature
Special Features		IR projector	Skeleton Tracking Object detection Spatial mapping
Price[5]		399 $	499 $ [6]

[1] 2.1 mm is the focal length, there is also an option with 4 mm for increased resolution and depth accuracy at longer range

[2] Infra Red

[3] Frame Per Second

[4] FoV: Field of View(Horizontal x Vertical)

[5] The prices are according to the official website without the costs of shipping

[6] The price jumps to 549 $ for 4 mm focal length

Table 2.2: Comparative study between the stereo cameras D455 by Intel Realsense and ZED2i by Stereolabs

2.1.3 Camera Data

The camera output can be viewed directly from its SDK in three formats, as shown in Fig.(2.2). The higher fps offered by the camera are at the expense of lower quality.

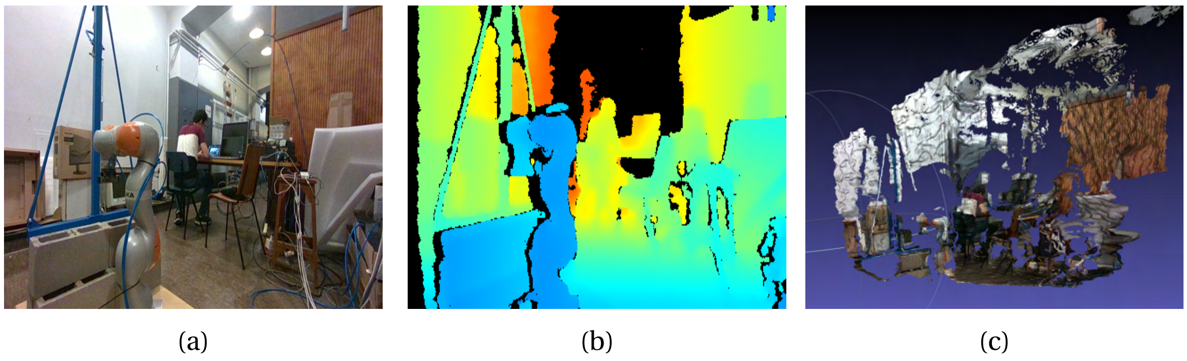


Figure 2.2: Different outputs for Intel Realsenes D455 (a) Color frame: 2D grid of RGB pixels(b) Depth map: a color frame where each pixel has its values based on depth (c) Point cloud: a set of 3D coordinates w.r.t the camera frame

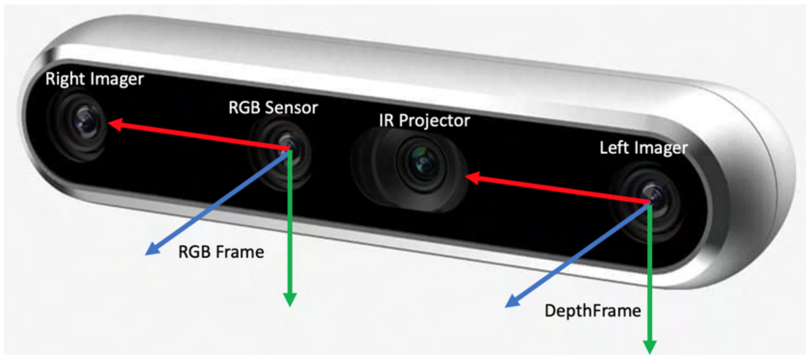


Figure 2.3: Intel Realsense D455 Frames, Red, Green, and Blue for X, Y, and Z, respectively

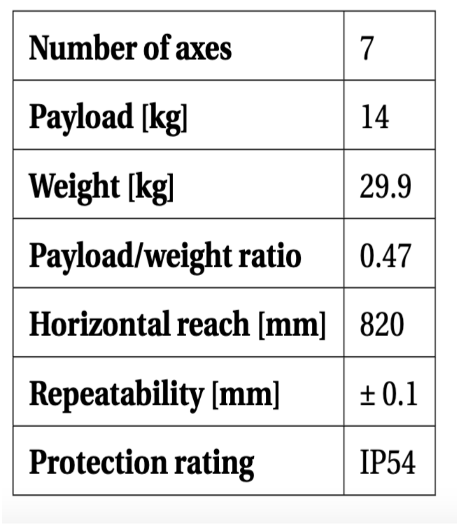
The different outputs can also be accessed through the famous computer vision library OpenCV. From any pixel given in Fig.(2.2a), depth can be retrieved w.r.t. RGB frame[1], hereby referred to as camera frame throughout the study. Depth is nothing but the z coordinate w.r.t. camera frame. Furthermore, for each pixel, the 3D coordinates of the associated point in 3D space can be retrieved as well. In section 8.1 of the Annex, practical hints about the camera setup and the use of its tools are present.

[1]Depth is always retrieved w.r.t. depth frame shown in Fig(2.3). For that, the depth reading should always be aligned with the RGB frame using the dedicated sample code from Intel Realsense

2.2 Robotic Arm

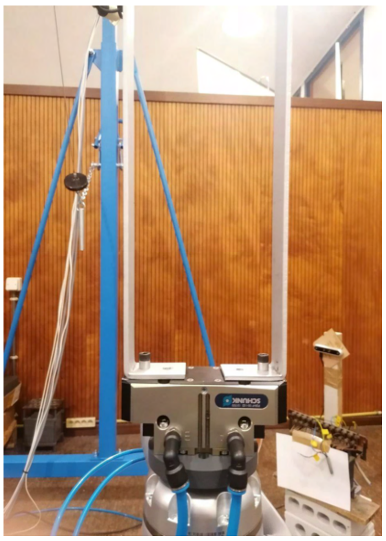
The robotic arm is a Kuka LBR IIWA14 R820, for which, the important specs are given in Table 2.4b. The arm has a pneumatic parallel gripper, as shown in Fig.(2.4c), with a max-min opening: 12-16 cm.

(a)



Number of axes	7
Payload [kg]	14
Weight [kg]	29.9
Payload/weight ratio	0.47
Horizontal reach [mm]	820
Repeatability [mm]	± 0.1
Protection rating	IP54

(b)



(c)

Figure 2.4: (a)Kuka LBR IIWA14 R820 Robotic arm (b) Technical data (c) Pneumatic gripper with parallel fingers

2.3 Software

All the development of this study was done using the following software versions:

- operating system: Ubuntu 20.04.4 LTS
- OpenCV[2]: version 4.5.5
- Python: version 3.8.10
- ROS[3] Noetic
- FRI[4] library 1.17

[2]Tip: OpenCV is inherently installed with ROS
[3]Robot Operating System
[4]Fast Research Interface: library used for Kuka robot control

Chapter 3

Object Detection and Pose Estimation

This chapter discusses the proposed method used for a block's detection and its pose estimation.

3.1 Proposed Algorithm for Object Detection

The real challenge about any vision-guided robotic system is to be able to extract the object of interest out of the image frame. However, detection is not enough because the final goal is to estimate the position and orientation of a block. Detection of an object from a 2D image can be done in several ways. One way to go is using a machine learning model trained with a large image data set of the object, and this is what famous libraries like TensorFlow are used for. Such models usually draw a bounding box around the detected object. However, in many vision-guided robotic applications, a development of such big data sets is troublesome, costly and time-consuming [13]. Another way would be using some image processing like binary thresholding, blurring, contour detection, polygon fitting...etc, to extract objects with certain features: like color, shape, repeating patterns, markers like QR codes, and this is what the famous library OpenCV is good for. Moreover, if the object to be detected is already known, SIFT (Scale-invariant Feature Transform) and SURF (Speeded Up Robust Features) are well-known algorithms for finding correspondences between a given image and an image of the known object, which enables them to finally detect the object. An implementation of these two for a service robot classifying food boxes is seen in [12]. Going further, depth information present in RGB-D images provide better classification for machine learning models as the model can make decisions also on the object's dimension [16]. For a point cloud, libraries like PCL(Point Cloud Library) can be used to do some segmentation and clustering processes to identify special features like some geometrical shapes or planar surfaces or even do different filters as in [12]. In the end, the training of a machine learning model is already a hard process. Furthermore, the functions related to 2D image processing are more flexible than those related to point clouds, not to mention that the use of easily identifiable fiducial markers or other hand-crafted features is well-justified as they provide good results in a variety of research projects, to the best of the author's knowledge. Hence, **this study proposes** the use of ArUco markers, a type of fiducial markers that is commonly used in vision applications. The next section will elaborate upon this type of markers.

3.2 About ArUco Markers

3.2.1 Theory and Implementation

ArUco markers are binary square fiducial markers that were initially developed for Augmented reality applications [21]. ArUco library stands for: **Augmented Reality University of Cordoba** and it was developed in [6]. A typical 6x6 ArUco marker is shown in Fig.(3.1a). It is called binary because it is made up of small squares(bits) that can either be black or white. Binary square markers in general are easily detectable in the image and their inner binary codification makes them specially robust, allowing the possibility of applying error detection and correction techniques[21]. As ArUco is written in C++, is extremely fast, and easily implemented, it has gained a great popularity in computer vision applications and a lot of online material is available on it, and this is why it is chosen for this study. Several other fiducials exist and have their own libraries like AprilTag, ArToolKit+, ARTAG, CHILITAGS [6]. The uses for ArUco, and fiducials in general, are camera calibration and pose estimation, for which, the methodology is well-explained on the website of OpenCV [21], however it is not comprehensive to all the details of the marker's use. A general summary of the implementation steps is given below:

Marker Creation A marker can be created as an image, as shown in Fig.(3.1a) with a sample script from OpenCV, however the image has to be printed in real dimensions on a PDF, which depends on PPI(Pixels Per Inch) rate. Therefore, it is recommended to use this online tool that generates the PDF for the marker with real dimensions.

Choice of Dictionary The dictionary is an indication of how many bits the marker is made of. There could be 4x4, 5x5, 6x6 and 7x7 and the choice depends on the image resolution and how far away the marker is. The more bits, the more words in the dictionary, and the smaller the chance of confusion. However, more bits mean that more resolution is required for correct detection [18]. That is why for a low resolution, it is better to choose a smaller dictionary. Each marker has its own ID and thus each dictionary has a limited size of markers. There are sizes ranging from 50 to 1000 for each dictionary, depending on the chosen type.

Detection Parameters Detection has a set of parameters that need to be tuned to have better detection of the marker, and for that it is advised to thoroughly understand the detection process explained in [21]. Detection for a normal image full of markers, as in Fig.(3.1b), starts with an adaptive thresholding which is good for detecting change of pixel intensities, as seen in Fig.(3.1c).Note that playing with the window size of the thresholding can have a significant effect on the boundaries of the markers, therefore these must be checked according to the lighting conditions. Furthermore, counter filtering followed by square polygon approximation is used to extract possible candidates of markers based on minimum and maximum allowed contour perimeters in **pixels**. Here also, allowing small perimeters to detect far markers can be at the expense of increased false positives. The next step is to do perspective transformation, as in Fig.(3.1d), in order to extract the value of each bit, which will be followed by dividing the transformed image into a grid of dimension equal to the chosen dictionary plus one for the border, as shown in Fig.(3.1g). Each small square in the grid will be assigned a value of one or zero and finally identify the ID of the marker. The parameters related to this process are better left to their default values. After the markers are detected, their corners are refined using several methods, The fastest method is the default one, however it is not the best. Better refinement comes at the expense of slower code hence a trade-off must be made. The selected parameters after tuning will be listed in section 3.4

Pose Estimation Pose is referred to as the position and orientation of a frame in space. If the camera's intrinsic parameters, namely the camera matrix and the distortion coefficients, and the marker's real length are known, then the frame constructed from detecting the marker's 4 corners in the **image frame**, as seen in Fig.(3.1f), can be de-projected into the corresponding 3D frame, which is the marker frame w.r.t. the **camera frame**. The estimated marker frames are drawn as in Fig.(3.1g), where the marker frame is centered at the marker center with R,G,B being for X, Y, Z, respectively.

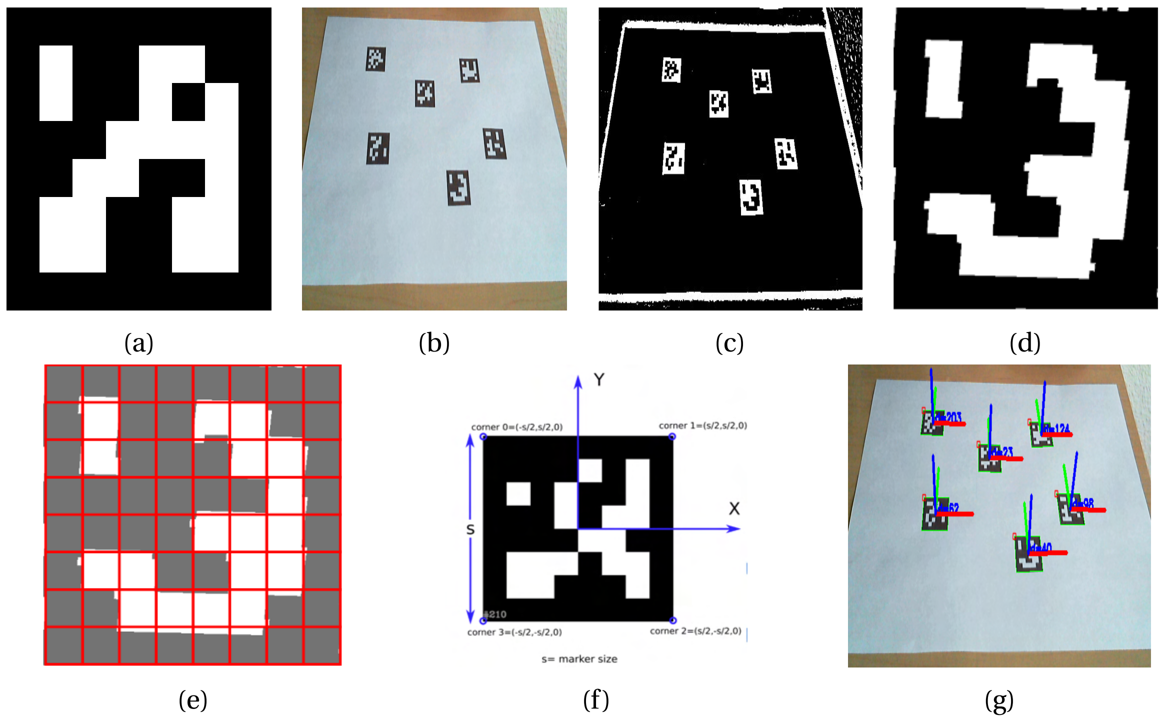
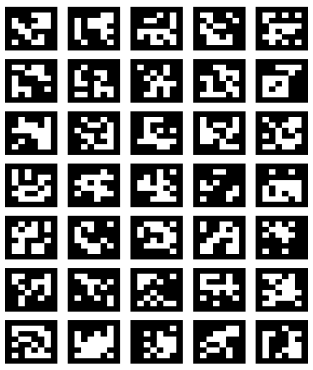


Figure 3.1: ArUco Detection and Pose Estimation (a) 6x6 ArUco Marker (b) Raw Image with ArUco Markers of different IDs (c) Image after Adaptive thresholding fingers (d) Marker candidate after perspective transformation (e) Marker cells to know the ID of marker (f) ArUco Marker frame in image frame (g) Marker frames in 3D w.r.t. camera frame drawn after pose estimation

3.2.2 Variants of ArUco

There are different variants spawning from ArUco, and for reasons that will be explained later, two will be used during this study: ArUco and ChArUco boards, Fig.(3.2a) and Fig.(3.2b), respectively.

For what concerns the creation, there is no online tool for ArUco boards, so they must be created as an image then converted to PDF. Care must be taken to not apply any scaling while printing to conserve dimensions. Practical tips on this aspect are present in the Annex in section 8.2. For ChArUco boards, this online tool is used.


(a)

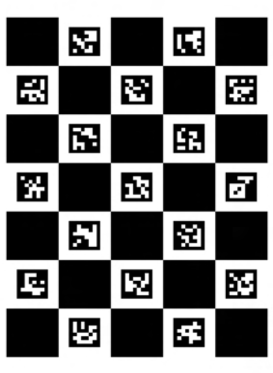

(b)

Figure 3.2: (a) ArUco Board (b) ChArUco Board= Chess + ArUco

3.3 Rotation Matrices and Homogenous Transformations

Before proceeding further, a brief recall is given on the rotation matrices and homogeneous transformations, which are very important tools in describing poses of rigid bodies w.r.t. each other.

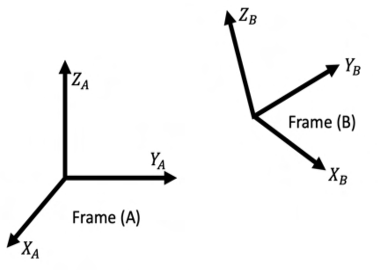
Rotation Matrix For two rigid bodies (A) and (B) in space having their respective frames, as shown in Fig.(3.3a), it is possible to find the **orientation** of frame (B) w.r.t. frame(A) using the rotation matrix as follows:

$$^{A}R_{B} = \begin{bmatrix} ^{A}X_{B} & ^{A}Y_{B} & ^{A}Z_{B} \end{bmatrix} \tag{3.1}$$

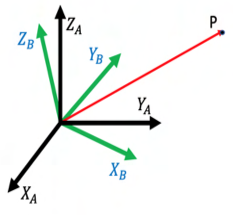
where $^{A}X_{B}, ^{A}Y_{B}, ^{A}Z_{B}$ are the unit vectors of frame (B) expressed w.r.t. frame (A).
The following properties are noted for a rotation matrix:

- Chain rule property, meaning: $^{A}R_{C} \times ^{C}R_{B} = ^{A}R_{B}$
- It is a 3x3 orthogonal matrix, meaning: $R^{T} = R^{-1}$, hence $^{A}R_{B} \times ^{B}R_{A} = I$ and determinant $= 1$
- For two frames with the same origin, as in Fig.(3.3b), a point P can be expressed in the other frame as follows:

$$^{A}p = ^{A}R_{B} \times ^{B}p \tag{3.2}$$



(a)



(b)

Figure 3.3: (a) Two frames in space (b) Two frames having the same origin

The rotation matrices corresponding to rotations around the x,y,z axes will be used throughout the study and are given below:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}, R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

There exist other mathematical notations for expressing orientation like axis-angle representation, and unit quaternions, each having its advantages and disadvantages; however, they will not be used throughout this study.

Homogenous Transformation For two rigid bodies (A) and (B) in space having the respective frames, as shown in Fig.(3.4), it is possible to find the **position** and **orientation** of frame (B) w.r.t. frame(A) using the homogenous transformation matrix as follows:

$$^{A}T_B = \begin{bmatrix} & ^{A}R_B & & t_{AB} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

where $^{A}R_B$ is the rotation matrix of (B) w.r.t. (A) and t_{AB} is the translation vector of (B) w.r.t. (A), which is the coordinates of O_B in (A).

The following properties are noted for a transformation matrix:

- Chain rule property, meaning: $^{A}T_C \times ^{C}T_B = ^{A}T_B$
- Always invertible, meaning: $(^{A}T_B)^{-1} = ^{B}T_A$
- A point p expressed in frame in (B), as shown in Fig.(3.4), can be expressed in frame

(A) as follows:

$$^{A}p = t_{AB} + ^{A}R_{B} \times ^{B}p \quad or \quad \begin{bmatrix} ^{A}p \\ 1 \end{bmatrix} = ^{A}T_{B} \times \begin{bmatrix} ^{B}p \\ 1 \end{bmatrix} \tag{3.5}$$

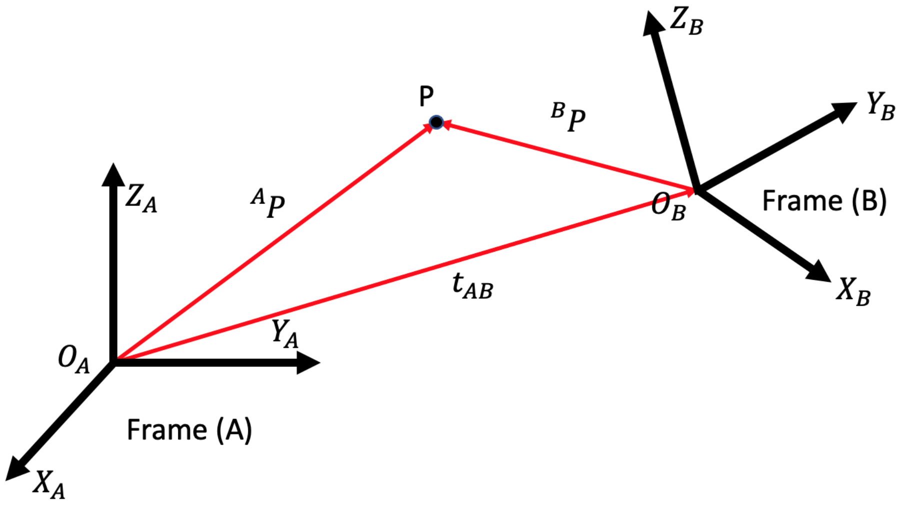


Figure 3.4: Two frames with different origins

3.4 Pose Estimation of Block

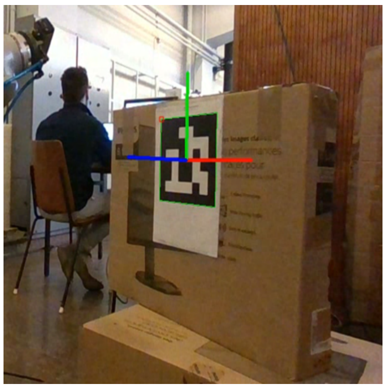
As explained in section 3.2.1, pose can be estimated by knowing the camera intrinsic parameters and the real size of the marker. So far, this can be performed by a regular monochrome camera[1]. The advantage of having the Intel Realsense D455 is:

- It is factory-calibrated, and unless the camera has received a blow or has been used for a long time, there is no need to perform camera calibration to extract the intrinsic parameters
- It is a depth camera. Pose is returned from the OpenCV pose estimator function as a translation vector *tvec* and a rotation vector *rvec* in axis-angle representation[2]. *tvec* is nothing but the 3D coordinates of the marker center w.r.t. camera frame, hence they can be replaced by the 3D coordinates retrieved by the camera at the pixel of the marker center. *rvec* can also be estimated by constructing vectors from the 3D points retrieved by the camera associated with the marker corners; however, this becomes tricky when dealing with an ArUco board, as it will be seen below. Therefore,

[1]A camera producing images in one color scale. For example, grayscale.
[2]It can be converted to a rotation matrix using Rodriguez formula, given by *cv2.Rodriguez()*

the translation vector is from the camera and the rotation one remains from the pose estimator.

In theory, if a marker is fixed on one surface of a block, as in Fig(3.5a), the marker frame is also considered as the block frame, and this frame can be transformed to any other frame rigid to the block. This is useful because for a such a block, the grabbing is performed on the side face, hence the block frame must be centered at the grabbing point. This will be made clearer in the grabbing experiments in chapter 5.


(a)

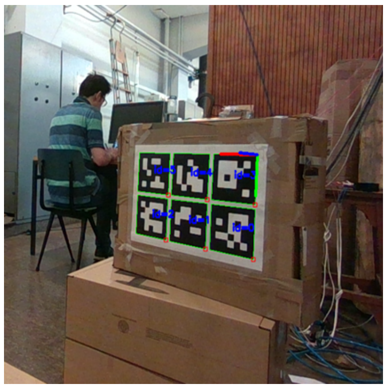
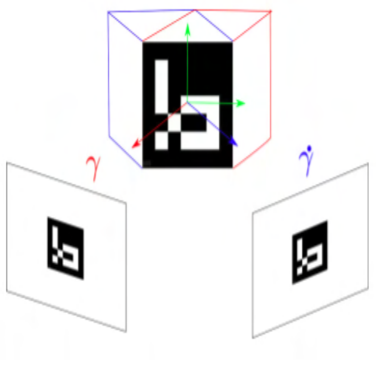
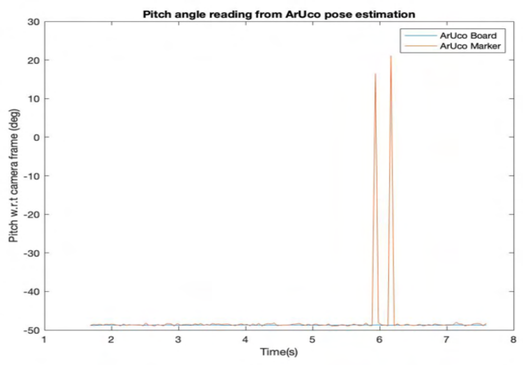

(b)

Figure 3.5: (a) Pose estimation of marker frame from single marker: X for Red, Y for Green and Z for Blue R. (b) Pose estimation with a 3x2 ArUco Board

In practice, some rotation matrix terms heavily fluctuate between two consecutive frames due to the ambiguity problem. The pose estimation using only 4 co-planar points, as seen in Fig.(3.1f), is subject to ambiguity [18]. A marker, as shown in Fig(3.6a), could project at the same pixels on two different camera locations. In general, the ambiguity can be solved, if the camera is near to the marker. However, as the marker becomes small, the errors in the corner estimation grows and ambiguity comes as a problem[18].



(a)



(b)

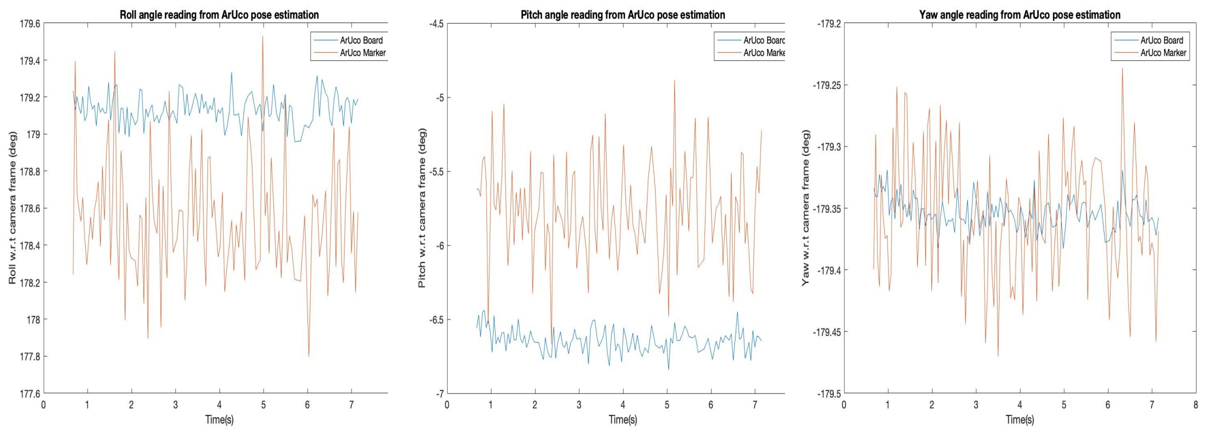
Figure 3.6: (a) The ambiguity problem: the same projection could come from two different cubes: the ones in red and blue (b) Spikes in pitch angle reading from pose estimation from one ArUco marker compared to the same pose estimation from an ArUco board

Another solution to **reduce** ambiguity is to estimate pose using several markers, and this is the role of ArUco boards presented in the previous section. ArUco boards provide more consistent and accurate readings, and also deal with occlusions[3]. A comparison is done to show the difference between the Euler angles readings from an ArUco marker and from an ArUco board. Note that the detection parameters were tuned and are shown in Table(3.1). Over a fixed interval of time, the pose of the ArUco board seen in Fig.(3.5b), 1 m away from the camera, is estimated. This pose is compared to that obtained from one marker of the board. Fig.(3.6b) shows spikes in the pitch angle reading from the marker, whereas there are no spikes from the ArUco board. Furthermore, Fig.(3.7) shows the readings of both poses when the block is parallel to the camera (little to no ambiguity). The board clearly provides readings with fewer fluctuations, having the following standard deviations in mm:

	Roll	**Pitch**	**Yaw**
Board	0.0768	0.0799	0.0128
Marker	0.3088	0.3290	0.0504

It can be seen that the board is almost 4 times less fluctuating than the marker. Moreover, the average of roll and pitch from the marker is shifted from that of the board. It is, therefore, concluded that single markers are not a reliable source for orientation estimation.

[3]Occlusions are when some markers are not detected in the image. For example, covered by an object



(a) (b) (c)

Figure 3.7: Euler angles readings from pose of ArUco board vs pose of one marker of the board: board orientation is almost 4 times less fluctuating than a marker's. Measurements at 30 Hz, 1 m away from the camera

adaptiveThreshWinSizeMin	5
adaptiveThreshWinSizeMax	20
adaptiveThreshWinSizeStep	5
cornerRefinementMethod	CORNER_REFINE_SUBPIX
cornerRefinementWinSize	3
cornerRefinementMinAccuracy	0.01
cornerRefinementMaxIterations	50

Table 3.1: ArUco Detection Parameters. The other parameters are left to their default values.

An attempt was made to validate the estimated pose with the motion capture system OptiTrack. However, it was suspended due to several technical difficulties. More details on this can be found in the Annex in section 8.3

Chapter 4

Eye-to-Hand Calibration

After having determined the block's pose w.r.t. the camera frame, it must now be obtained w.r.t. the robot's base so that the robot can grab the block. This chapter is about the eye-to-hand calibration, the process by which, the homogeneous transformation from camera frame to the robot's base frame is obtained.

4.1 Theoretical Framework

The camera to robot calibration is a well-known problem is vision-guided robotics, and in general, there exist 2 configurations [9]:

1. Eye-to-hand: the camera is attached to a fixed pole, facing the working range of the robotic arm's end-effector.

2. Eye-on-hand: the camera is mounted on the robotic arm, aligned with the end-effector.

The eye-to-hand configuration is chosen because the robotic design of the Brikiebot Project does not allow an 'eye-on-hand' solution; therefore, an eye-to-hand solution is chosen in this work.

Fig.(4.1) shows the problem of the eye-to-hand calibration. A calibration board, normally a chess board or an ArUco or a ChArUco board, is rigidly attached to the gripper and thus, ${}^{grip}T_{board}$ is fixed. At two different positions of the gripper, namely (i) and (j), ${}^{grip}T_{board}$ is the same, hence:

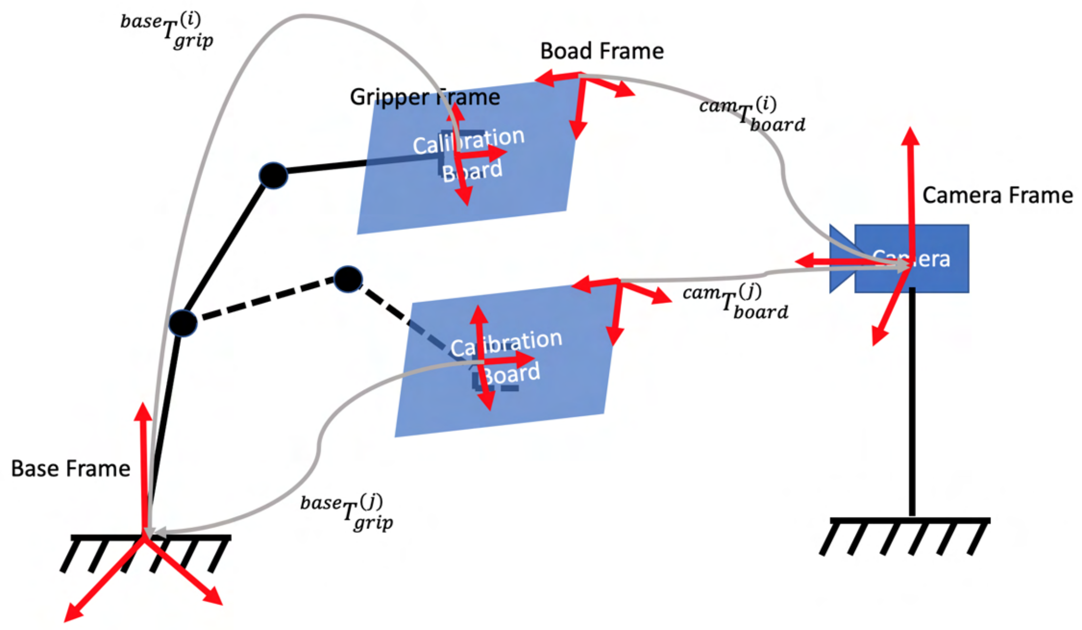
$$ {}^{grip}T^{(1)}_{base} \, {}^{base}T_{cam} \, {}^{cam}T^{(1)}_{board} = {}^{grip}T^{(2)}_{base} \, {}^{base}T_{cam} \, {}^{cam}T^{(2)}_{board} \tag{4.1} $$

Right multiplying by $({}^{cam}T^{(1)}_{board})^{(-1)}$, then left multiplying by $({}^{grip}T^{(2)}_{base})^{(-1)}$, we get:

$$({}^{grip}T^{(2)}_{base})^{(-1)}\,{}^{grip}T^{(1)}_{base}\;{}^{base}T_{cam} = {}^{base}T_{cam}\;{}^{cam}T^{(2)}_{board}({}^{cam}T^{(1)}_{board})^{(-1)} \tag{4.2}$$

(4.2) can be seen as the system $A_iX = XB_i$, where X is ${}^{base}T_{cam}$

A_i and B_i can be obtained by getting the following for each motion of the gripper: ${}^{cam}T_{board}$ and ${}^{grip}T_{base}$



${}^{base}T^{(i)}_{grip}$
Boad Frame
Gripper Frame
Calibration Board
${}^{cam}T^{(i)}_{board}$
Camera Frame
Camera
${}^{cam}T^{(j)}_{board}$
Calibration Board
Base Frame
${}^{base}T^{(j)}_{grip}$

Figure 4.1: Eye-to-hand calibration problem: obtain ${}^{base}T_{cam}$ where (i) and (j) correspond to different positions of the robotic arm's gripper

4.2 Calibration Setup

Calibration Tool OpenCV provides a function that solves the previously mentioned system. At least 3 **different and non-parallel** poses are required to perform the calibration. However, more poses are recommended to have converging results. OpenCV offers five different methods to perform the calibration, for all of which, the convergence study will be done. However, it is out of the scope to explain about each method. Additional information can be found in [22].

Camera Fixation The camera is first fixed to a wooden plate using two M3 screws in the mounting threads on the back of the camera. Then, the plate is screwed to a wooden beam fixed to the ground. The fixed camera is shown in Fig.(4.3a).

Calibration Board Two boards are used: ArUco and ChArUco boards. Calibration was performed 4 times: the first three with an ArUco board and the last with a ChArUco one. They are both recommended to be used by OpenCV and by the different online forums. A wooden plate is rigidly attached to the gripper using two screws, then the calibration board is taped to it. The setup can be seen in Fig(4.3b). Care must be taken to not allow any air pockets under the board.

Data Recording For each position of the gripper, the following data are recorded in a spreadsheet:

- ${}^{cam}T_{board}$, using the pose estimation with OpenCV explained in 3.4
- ${}^{grip}T_{base}$, using the robot's direct kinematics model in MATLAB[1]

Procedure

1. Display the camera view
2. Manually manipulate the robot's joint angles to a position close to the camera whilst the board is fully detected, as in Fig.(4.3b)
3. Record the joint angles at the current position and feed them to the MATLAB model. Then, record the board's pose with camera[2].
4. Repeat the previous steps for a new sample. Note the following:
   (a) manipulated positions should be taken as diversely as possible in all directions in the image frame, as explained in Fig.(4.3c).
   (b) Go to very close positions from the camera (the closer you are, the more accurate the results), then go further(not more than 1 m) from the camera. When taking close positions, go to the extent where the frame is filled with the board while being fully detected, as in Fig.(4.3d)

[1]This model was validated by a researcher in BrikieBot project
[2]The camera can only do one thing at a time: record or display

(c) A practical tip is to change only one joint angle between two consecutive positions then start changing another angle when the current is not giving the desired variation

5. After taking at least 20 samples, check the convergence of the results. 20 is an empirical number based on what was seen in different online forums.

The flowchart of the whole procedure can be seen in Fig.(4.2)

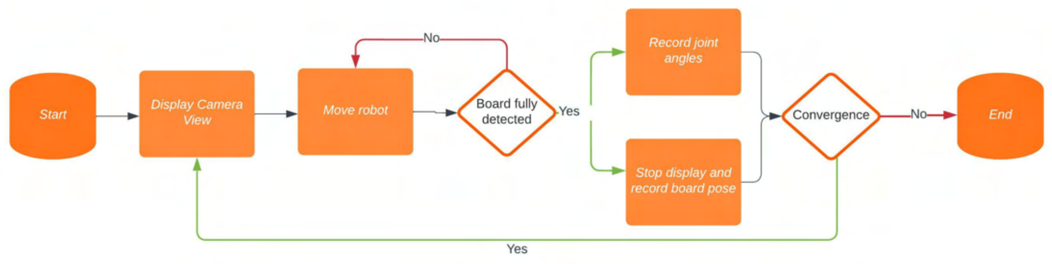


Figure 4.2: Eye-to-hand Calibration Flowchart

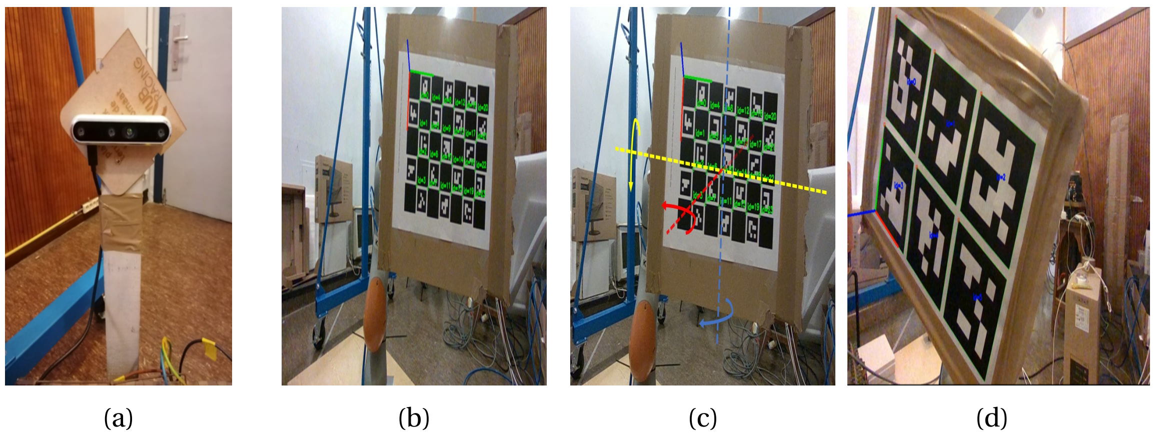
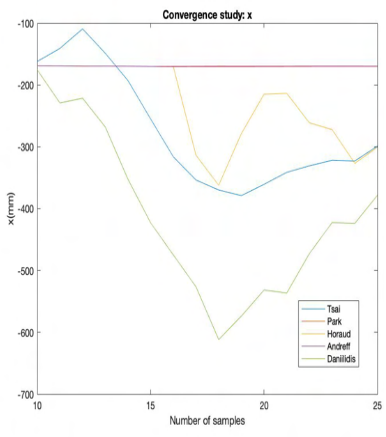


Figure 4.3: (a)Camera fixation (b)Initial calibration position: calibration board fully detected by camera (c) Random poses should include different rotations around 3 perpendicular axes (d) Close pose of a board filling image frame

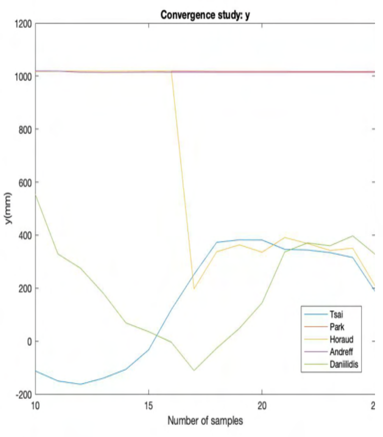
4.3 Convergence

As stated previously, OpenCV offers 5 methods for performing the calibration: *Tsai, Park, Horaud, Andreff,* and *Daniilidis.* It is important to check the convergence for each of them to know if the results can change by taking more samples. 4 calibrations were performed: 3 with an ArUco board and 1 with a ChArUco board. ChArUco results converge more smoothly.

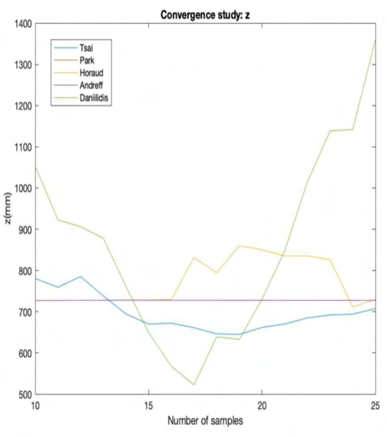
For the sake of brevity, only the ChArUco results will be presented, whereas the ArUco results for 35 samples can be found in section 8.4 of the Annex. Fig(4.4) shows the convergence study of ChArUco board calibration where 25 samples were taken. It can be clearly seen that only the methods of *Park* and *Andreff* show a converging behavior. Each method solves the system presented in the previous section differently; however, it is not the purpose of this study to analyze these methods. Further information on these methods can be found in [23].



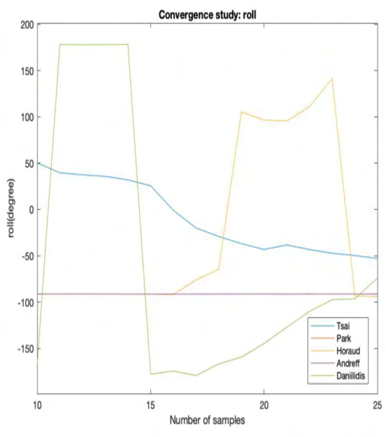
(a)



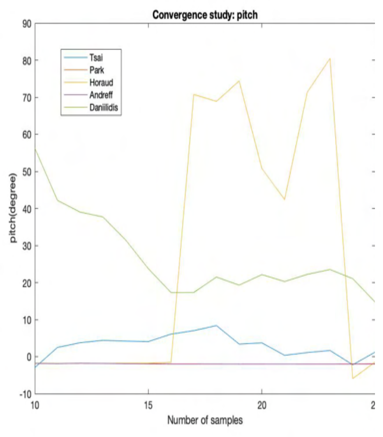
(b)



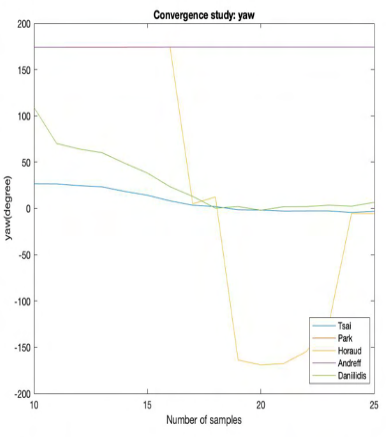
(c)



(d)



(e)



(f)

Figure 4.4: Convergence study of eye-to-hand calibration with ChArUco board for the 5 different methods offered by OpenCV. Only the methods of *Park* and *Andreff* show a converging behavior

4.4 Validation of Results

Having obtained the ${}^{base}T_{cam}$, it is important to have a sense of its accuracy. In what follows are two proposed experiments for a rough validation of the results.

4.4.1 Position

The first experiment is to validate the position. For this, the gripper center is manually controlled to the center of an ArUco marker as shown in Fig.(4.5), then the gripper position from the direct kinematics is compared to what the camera reads after lifting the gripper from the marker. One marker is used because only position is needed for this experiment, so the orientation is irrelevant, and that is why using a bigger marker than the individual markers of the board is chosen. Three measurements were taken at different positions. The results are shown in Table(4.1). As the gripper was manually controlled to the desired position, an acceptable error range would be 5 mm. The method of *Park* shows less RMS[3] error than that of *Andreff* with a max error of 3.04 mm, compared to 4.27 mm for the latter, which is acceptable for both.

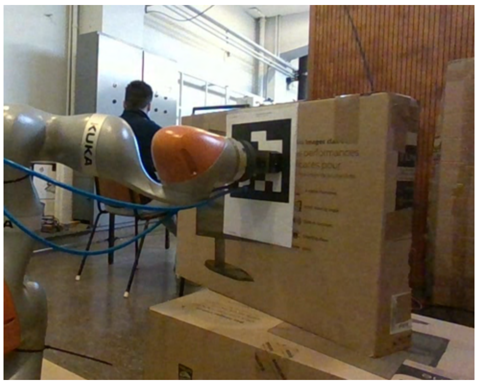


Figure 4.5: Approximate validation of position by aligning the gripper center to the center of the ArUco marker, then marker center position is measured. Results are in Table(4.1).

Measurement 1 (mm)				**Measurement 2 (mm)**				**Measurement 3 (mm)**			
3D Coordinates	**x**	**y**	**z**	**3D Coordinates**	**x**	**y**	**z**	**3D Coordinates**	**x**	**y**	**z**
Park	-556.07	-381.32	705.33	**Park**	-572.74	79.52	735.63	**Park**	-436.74	-168.21	703.02
Andreff	-557.59	-383.92	707.95	**Andreff**	-573.86	76.98	737.57	**Andreff**	-438.10	-170.90	705.15
Real	-553.99	-376.72	703.29	**Real**	-570.20	78.66	732.20	**Real**	-434.20	-170.60	704.10
			RMS Error								
Park	2.40	3.04	2.39								
Andreff	3.72	4.27	4.15								

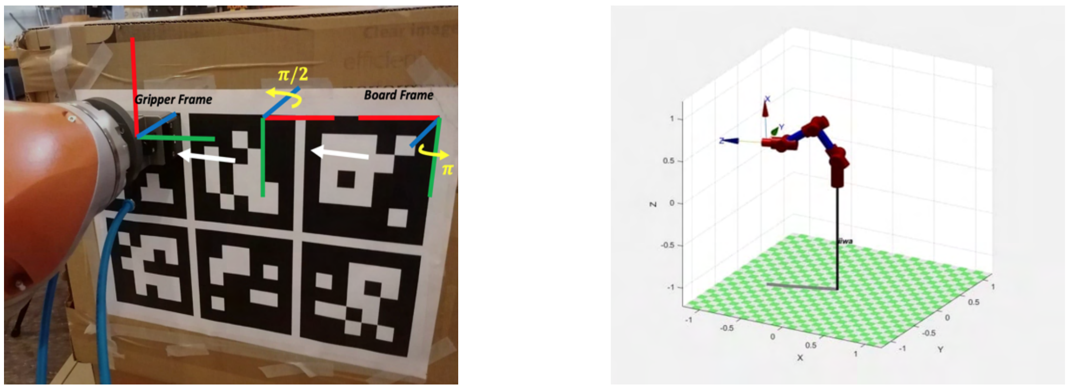
Table 4.1: Position validation of eye-to-hand calibration. Results are within the acceptable range due to manual alignment of gripper

[3]Root Mean Square

4.4.2 Orientation

The second experiment is to validate the orientation. For this, the gripper is manually moved so that its edges are aligned to the edges of the ArUco board, as shown in Fig.(4.6a). It does not matter where the gripper center is placed as long as its edges are aligned with the board's edges. For this experiment, only orientation is relevant, hence a board is used. Furthermore, as the orientation is very sensitive to any small perturbations, the measurement is done while the gripper is on the board. Moreover, Fig.(4.6b) shows the gripper's orientation in the base frame, which is not the same as the board's orientation. For that, a first rotation is done around y by 180 deg to align z , then a second rotation around the new z by 90 deg to align x and y. Mathematically, this is expressed as:

$$^{base}R_{aligned2grip} = ^{base}R_{board} \times R_y(180) \times R_z(90) \tag{4.3}$$



(a) (b)

Figure 4.6: Approximate orientation validation of eye-to-hand calibration (a) gripper edges are manually aligned with marker edges, then board orientation is measured (b) Gripper frame drawn from direct kinematics at the robot position in (a). The board frame must be rotated to be aligned with the gripper frame. Results are in Table (4.2)

four measurements were taken, and the results are shown in Table(4.2). As the gripper was manually aligned, an acceptable error range would be 5 deg. The max RMS error of *Park* is 9.74 deg on roll, compared to 9.88 deg for *Andreff*, also on the roll. Furthermore, both methods have a very good approximation for pitch, scoring less than 1 deg RMS error. At first sight, the relatively high errors on roll and yaw only can be attributed to the non-diversity of taken samples; however, having repeated the calibration for four times, and taking as diverse samples as possible, the results merely change.

Measurement 1 (deg)				Measurement 2 (deg)			
Euler Angles	**Roll**	**Pitch**	**Yaw**	**Euler Angles**	**Roll**	**Pitch**	**Yaw**
Park	143.98	-85.34	-134.32	**Park**	150.60	-85.22	-114.99
Andreff	144.03	-85.46	-134.42	**Andreff**	150.17	-85.33	-114.60
Real	134.40	-84.64	-124.59	**Real**	143.00	-84.77	-107.60
Measurement 3 (deg)				**Measurement 4 (deg)**			
Park	151.74	-88.67	-121.38	**Park**	175.41	-88.06	-175.78
Andreff	150.68	-88.78	-120.37	**Andreff**	177.91	-88.14	-178.32
Real	138.50	-87.58	-108.53	**Real**	168.01	-87.10	-168.74
RMS error							
Park	9.74	0.84	9.54				
Andreff	9.88	0.94	9.71				

Table 4.2: Orientation validation of eye-to-hand calibration. Results are within the acceptable range for pitch but not for roll and yaw

4.4.3 Conclusion

Although the errors for the position are accepted, the errors of orientation are not within what is expected. This can either be related to something wrong done in the experiment setup, or it is indeed the calibration error. Normally, an offset can be added to reduce this error. However, this will be left to when the grabbing is done. For the next chapters, the method of *Park* will be used, as it has scored less RMS error in the position and orientation.

4.5 GitHub Repository

One of the important achievements of this study is the development of a pipeline in Python language that makes the eye-to-hand calibration with OpenCV easier. Even though the calibration function is already provided by OpenCV, the tools developed in this study serve for:

- organization of recorded samples in spreadsheets
- display of calibration board
- pose recording

- check convergence
- utility functions for Euler angles and rotation matrices

All this can be time-consuming if the camera needs to be re-calibrated. Therefore, all the written scripts will be made available in the form of a GitHub repository [4]: link. Furthermore, it has not been much time since OpenCV added this functionality, so it is useful for the robotics community to add the data obtained with the 5 calibration methods.

[4]The repository will be published shortly after the submission of this study to allow time for reorganizing all scripts in a clean way

Chapter 5

Grabbing Experiments

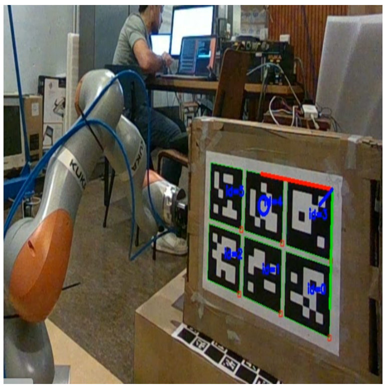
Grabbing the suspended block was the main reason of using a depth camera. In what follows, two experiments to perform the grabbing are proposed. One for a static block laying on the ground, and another for an oscillating block suspended to a crane.

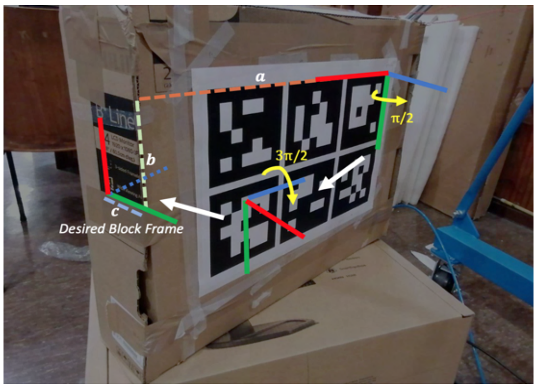
5.1 Static Block

As explained in section 3.4, the obtained pose from the marker frame is also considered as the block frame and this frame can be transformed to any other frame rigid to the block. The grabbing is performed on the side face and the gripper should be placed as in Fig.(5.1a), hence the block frame must be centered at the grabbing point, and it must be aligned with the gripper as in section 4.4.2. For this, the homogeneous transformation is a translation of the obtained frame along x,y and z then rotation by 90 degrees around y, followed by a rotation around z by 270 degrees. This is shown in Fig.(5.1b).

Mathematically, this is expressed as:

$$R_y(90) \times R_z(270) = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad , hence \quad {}^{cam}T_{block} = {}^{cam}T_{board} \times \begin{bmatrix} 0 & 0 & -1 & a \\ 1 & 0 & 0 & b \\ 0 & -1 & 0 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$


(a)



(b)

Figure 5.1: (a) Desired grabbing of block (b) Applying a homogeneous transformation to obtain the block frame at a desired grabbing position

As there is no collision avoidance in the trajectory planning of the robot, the robot is first asked to go to the ${}^{cam}T_{block}$, seen in eqn.(5.1), but shifted up in the z direction, as seen in Fig.(5.2). This is then followed by a vertical descent of the gripper by the amount that was shifted. At 1 m away from the camera, three grabbing experiments at different board orientations were realized and are shown in Fig.(5.3). The robotic arm in Fig.(5.3c) reached a joint limit and could not, therefore, descend any further. Note that the block width is 12 cm and the gripper opening is 15 cm. It was mentioned in section 4.4.3 that some tuning offset can be added to correct the obtained ${}^{base}T_{cam}$; however, the grabbing is visually acceptable to what is required. Hence, no tuning will be done.

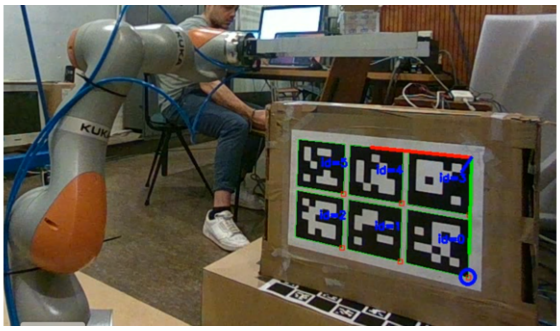
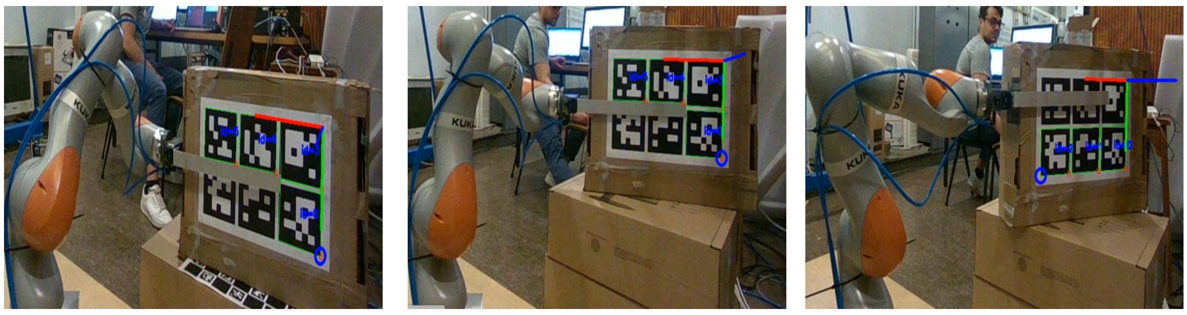


Figure 5.2: Gripper is first asked to go to a position vertically shifted up to avoid collision with block. Then, it descends vertically to the required position



(a) (b) (c)

Figure 5.3: Three grabbing experiments at different board orientations. The robotic arm in (c) reached a joint limit and could not descend further. The block width is 12 cm and the gripper opening is 15 cm

5.2 Oscillating Suspended Block

The grabbing of a suspended block, if static, is no different from the grabbing performed in the previous section. If the block is oscillating for some reason, the robot cannot proceed to grab. Therefore, this section proposes a criterion for determining whether a block is static or not, based on which, a grabbing decision is made. The simplest approach to say that a block is motionless is when its pose is no longer changing, and this is characterized by computing the standard deviation over a fixed-size, sliding window of previous measurements, also known as moving standard deviation. Fig.(5.4) demonstrates this idea with a window of size: 3. Note that this algorithm depends on previous values. Thus, it can only start working if there is at least a window size of previous and current values.

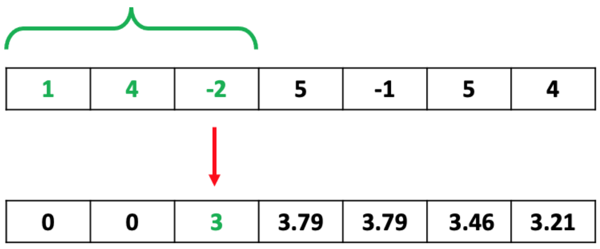
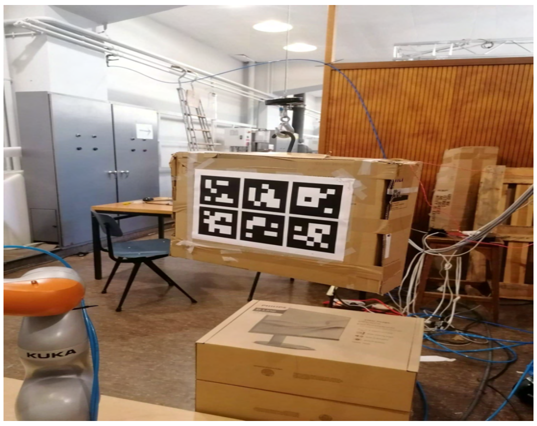


Figure 5.4: Moving standard deviation of window size 3

The proposed algorithm is to judge a block is static when the moving standard deviations of the pose coordinates are below certain limits. To determine these limits, a test to measure pose for a static, suspended block is done. The test is over 10(s) interval at 1 m away from the camera. Looking at the standard deviations in Fig.(5.6), the limits will be empirically defined as twice the values for what was obtained for each pose coordinate. For every measured pose, a block is considered static if **all** the pose coordinates are below their respective limits, otherwise it is considered moving.

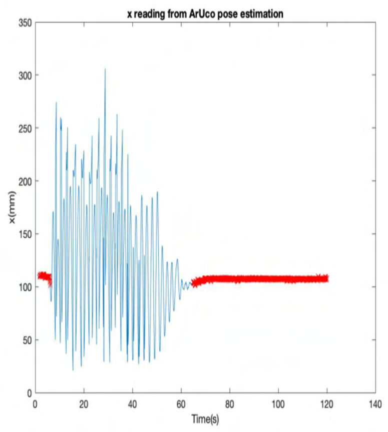


Pose	Standard deviation
x (mm)	0.65
y (mm)	1.12
z (mm)	4.27
Roll (deg)	0.08
Pitch (deg)	0.11
Yaw (deg)	0.07

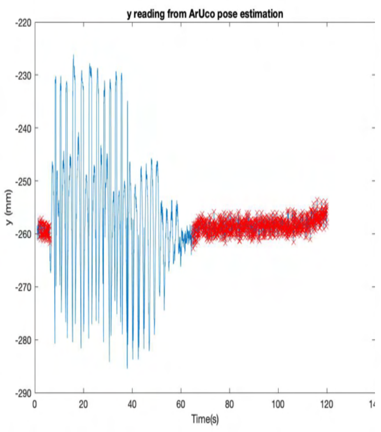
Figure 5.6: Standard deviations of pose measurements of a static suspended block over a 10(s) interval, 1 m away from the camera

After applying a perturbation on the suspended block, seen in Fig.(5.6), and giving it enough time to settle to the end, Fig.(5.7) shows in red the poses that were considered for a static block. A window size of 20 was empirically chosen; otherwise, there would be false positives. The algorithm determined two regions where the block is static, corresponding to the beginning before the perturbation and the end after settling down. The results are acceptable to proceed with grabbing the block; however, the following points are noted:

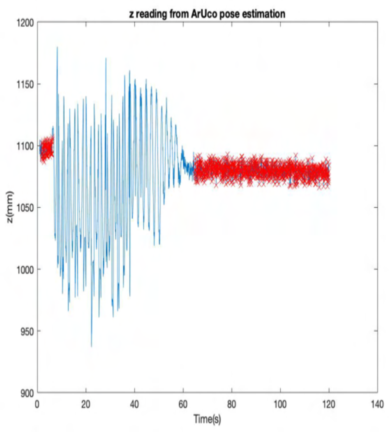
- A window size of 20 indicates that the camera needs to take 20 frames to judge a block is static. With a 30 fps, this means 0.66 (s) of delay. This can be optimized with a lower fps, at the expense of resolution
- A drawback of the use of Euler angles is the boundaries of their interval: $[-\pi, \pi]$. The oscillations between the two values can be clearly seen in the roll and yaw measurements. To the author's knowledge, this can be solved using quaternions, but for this study it does not cause a problem.



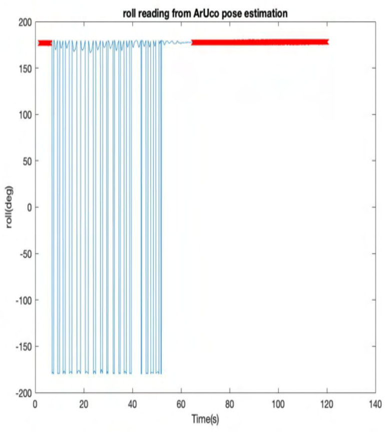
(a)



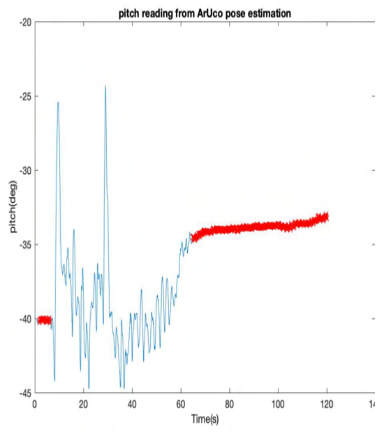
(b)



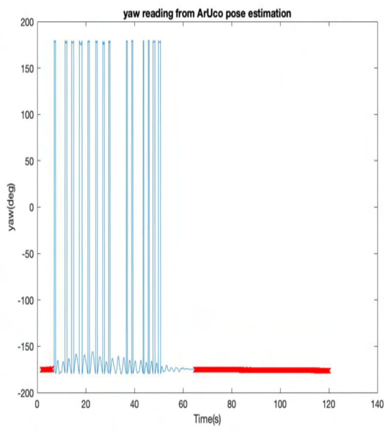
(c)



(d)



(e)



(f)

Figure 5.7: Using a moving standard deviation with a window size of 20 to determine if a suspended block is static. Poses in red indicate that the block is static in two regions, corresponding to the beginning before perturbing the block and the end after settling down

Fig.(5.8c) shows the final result of a grabbing action initiated only after a perturbed block was considered static. A video demonstration can be found here. As explained in the previous section, due to the absence of collision avoidance feature in path planning, the gripper is moved to a close position below the block, as in Fig.(5.8a), then it is aligned with the block frame just below the block, as seen in Fig.(5.8b). After that, it moves to the grabbing position in a series of movements to guarantee that it follows a straight line to the final grabbing position, seen in Fig.(5.8c).

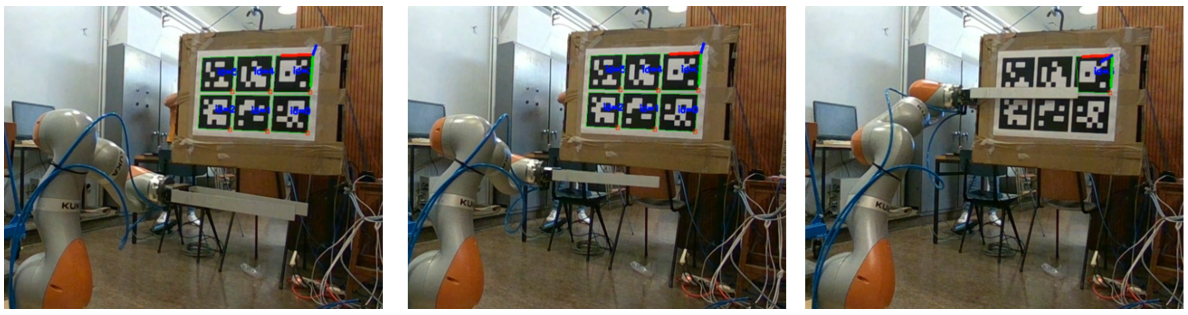

(a) (b) (c)

Figure 5.8: Grabbing experiment of an oscillating suspended block after settling down (a) initial position close under the block (b) intermediate position: alignment with block frame whilst under the block(c) final grabbing position

5.3 Camera and Robot in ROS

For what concerns the robot-camera interface in ROS, it is a simple architecture. There are two nodes: a node that performs the trajectory planning of the robot, referred to as "Kuka", and a node that processes the image frame and returns the pose of the block, referred to as "IntelD455". Camera node publishes to two topics and the robot node subscribes to two topics as explained below:

- "IntelD455" publishes pose to the topic "BlockPose"
- "IntelD455" publishes block state (static or moving) to the topic "BlockState"
- "Kuka" subscribes to the topic "BlockPose"
- "Kuka" subscribes to the topic "BlockState"

The corresponding rqt graph of this architecture is shown in Fig.(5.9). The algorithm is that when "Kuka" receives a "BlockState" of static, it takes the block's pose from "BlockPose" and proceeds to grab it.

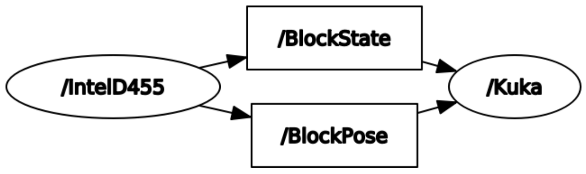


Figure 5.9: Camera-Robot interface in ROS: rqt graph of the related nodes and topics

The entire hardware/software architecture is shown in Fig.(5.10). From the robot's side, robot applications are programmed in Java and executed on the robot controller. In this study, KUKA Sunrise.OS 1.17, KUKA Sunrise.Workbench 1.17 and KUKA Sunrise.FRI 1.17 are used to program the robot. To execute programs in robot applications, the FRI client applications are created, in C++/Python, and are executed on an external system. In our case, on a Laptop with Intel(R) Core(TM) i7-6500U CPU 2.50GHz 2.60 GHz. The C++ applications implemented in this study[1] run in parallel and are the:

1. FRI Interface. This application interfaces the control architecture with the robot controller. It is needed to read robot joint positions and send robot torque commands.

2. The "Kuka" node, which uses the ROS package: Kinematics and Dynamics Library (KDL), distributed by the Orocos Project, to compute the inverse kinematics and plan the trajectory to any given pose. This node runs at 200 Hz

Lastly, the "IntelD455" node, which is written in Python, configures the camera, retrieves raw images, estimates the block's pose and evaluates whether the block is static or not. This node runs at 20 Hz[2].

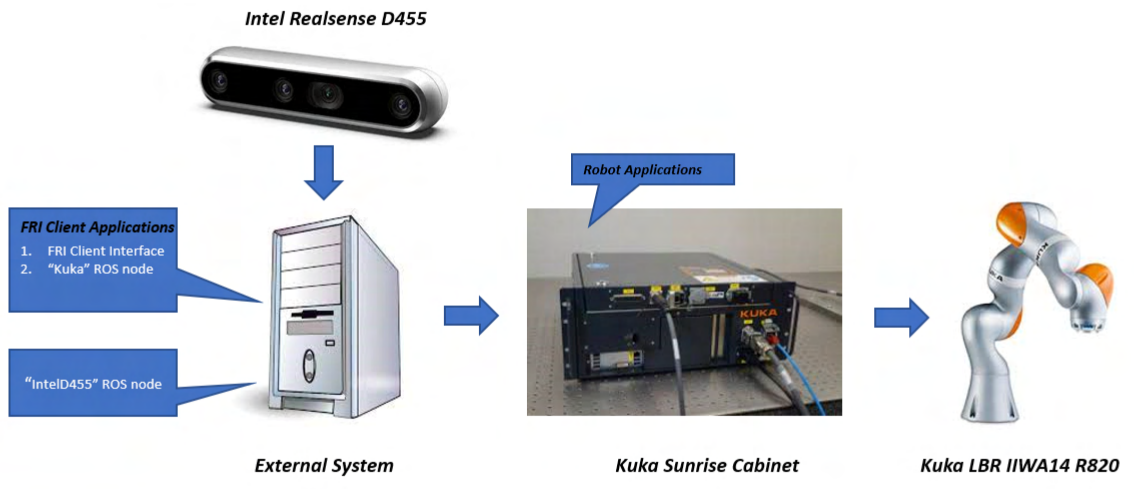


Figure 5.10: Software/hardware architecture of the grabbing experiments

[1]The FRI client application are already written by the BrikieBot researcher
[2]The "Kuka" node frequency must be a multiple of the "IntelD455" node frequency

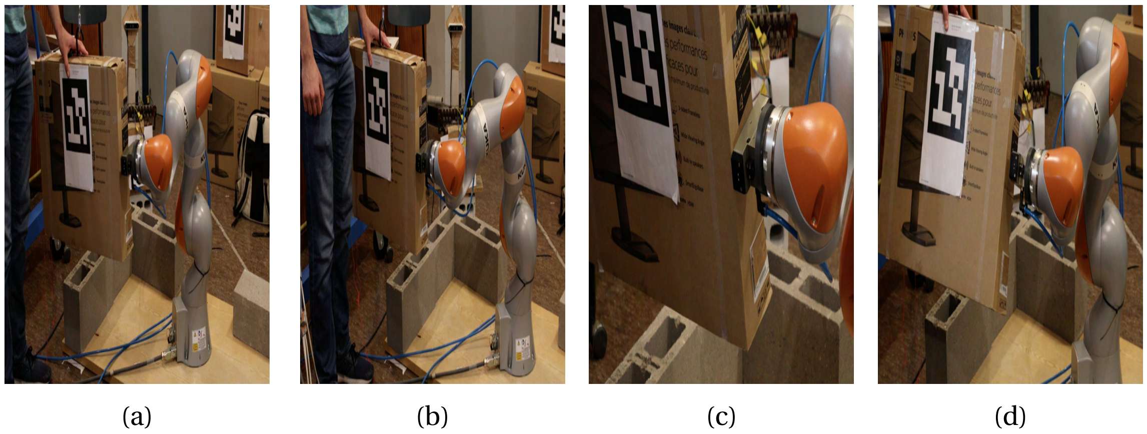
Chapter 6

Pose Tracking

The proposed grabbing algorithm in the previous chapter only works with blocks that are relatively lightweight w.r.t. the robot's payload. For real heavy blocks seen in Fig.(1.2), the robot must go along with the block and dampen its motion slowly so that its inertia does not damage the gripper. To test this approach, a very similar architecture is used as in the previous chapter with only the following differences:

- The "IntelD455" node is publishing to "BlockPose" topic at 40 Hz
- "BlockState" topic is no longer needed
- The "Kuka" node always asks the robot to go to the pose subscribed from "BlockPose"

Different orientations of the block when the robot is asked to follow its grabbing pose can be seen in Fig.(6.1), and a video demonstration can be found here. Note that the gripper fingers were unmounted for easier use.



(a) (b) (c) (d)

Figure 6.1: Different orientations of the block when the robot is asked to follow its grabbing pose

The following challenges are noted:

- The pose data is very noisy, causing heavy fluctuations of the robot around a fixed position of the block. This can be divided into two parts:

  1. Spikes due to ambiguity or other reasons. Spikes in poses physically induce wild movements of the robot. These spikes were already filtered out by empirically setting a derivative limit on each of the pose elements. However, implementing such a filter also means that the robot best responds to slow movements, which is acceptable because no grabbing will be done for rapidly oscillating blocks
  2. Small variations of readings for an almost static block make the robot's behavior unsteady. This is improved by tuning a moving average filter. A moving average filter is the same as the moving standard deviation seen in Fig.(5.4), but with an average instead.

- The frequency of pose estimation is not only related to the camera fps, but also to the speed of the pose estimation script. An **important** conclusion regarding Python is that it slows down the frame rate. For a frame rate of 30, and resolution of 1280x720, the total frequency of the camera node could not go more than 20 Hz because the execution time of the Python script is more than 1/30= 33 ms, which explains the decrease in frequency. For a frame rate of 60, and resolution of 848x480, the total frequency could not go more than 40 Hz, which is reasonable since lower resolution means lower processing time. To be able to achieve closer fps to the camera fps, it is strongly advised to write in C++ as it is a low-level language.

- Pose estimation frequency is obviously a hardware/software limit. Even with an optimized frequency, an important problem is that the camera could not be fast enough to map the dynamic behavior of the block. For that, a dynamic estimator, an extended Kalman filter for instance, can be used to fuse the pose of the block's dynamic model with the pose from camera.

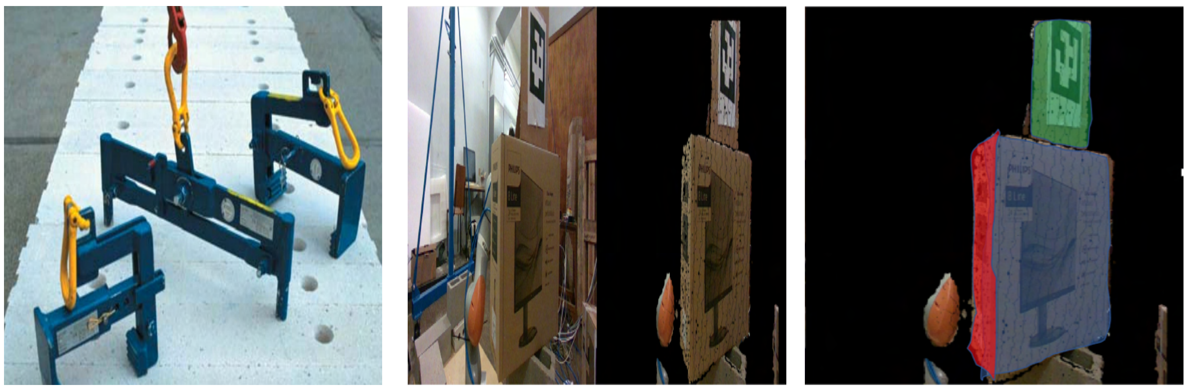
Chapter 7

Future Work and Conclusion

7.1 Future Work

The BrikieBot project proposes a quite innovative approach for bricklaying. From within the scope of computer vision, the following aspects could be interesting for future work:

Point Cloud Processing The study did not propose any detection approach based on point cloud processing. An ArUco marker can be placed above a block. In reality, the marker would be placed over a modified version of the clamp that connects the block to the crane cable, shown in Fig.(7.1a). Based on the measured marker's depth, the point cloud can be filtered in a way to include only the region around the box, as in Fig.(7.1b). Then, this point cloud can be analyzed with RANSAC, in PCL (Point Cloud Library) for example, to find all the flat surfaces in it, as is done in [19] and [2]. An expected output is shown in Fig.(7.1c). Note that it was already tried with typical image processing techniques like contour detection, shape fitting...etc. to extract surfaces, but the main problem was that there is no clear way to separate the adjacent surfaces, whereas in a point cloud, they can be separated because every pixel has 3D coordinates. Then, the detected flat surfaces can then be fitted into polygons of known corners, from which pose can be estimated as the vector product of the vectors of two adjacent edges. This way, there is no need to estimate orientation from an ArUco board. The disadvantage could be processing time and a possibly heavy fluctuations depending on the polygon-fitting algorithm.



(a) (b) (c)

Figure 7.1: Applying a depth filter based on an ArUco marker placed on a modified clamp (a) Regular clamp used with real blocks (b) Depth filtering with an ArUco marker (c) Expected output with point cloud segmentation algorithms like RANSAC

Inertial Measurement Unit A 9DOF IMU, namely an accelerometer, a gyroscope, and a magnetometer, can give the absolute orientation w.r.t. NED[1] frame. If such an IMU is placed on the connector between the block and the crane cable, and with a 9DOF IMU in the camera, the relative orientation between the IMU and camera frame can be obtained. A Python script was developed to read Euler angles from a BNO055[2] 9DOF IMU with an Arduino Uno board. Unfortunately, it was later discovered that Intel Realsense D400 series only features 6 DOF IMUs. This is why, referring to Table (2.2), the ZED2i camera would have been a better choice because it features a 9 DOF IMU. Furthermore, the ZED2i is more featured for spatial mapping, which is useful if, in the future, the camera is to compare a CAD model of a desired wall to an actual wall and make decisions. A possible disadvantage of an IMU is to align its frame to the block frame.

LiDAR Using a laser scanner or a LiDAR, as with the In-Situ Fabricator mentioned in section 1.3.2, provides denser depth maps than cameras, which is better for processing and extracting geometrical features. Unfortunately, these scanners are much more expensive than cameras.

Testing with Commercial Microprocessors To the author's knowledge, image processing algorithms behave differently when tested on real hardware than when tested on computers. An off-the-shelf microprocessor, with a graphics card, that is commonly used in vision-

[1]North East Down
[2]Helpful online material for this sensor is present here

guided robotics is the Jetson Nano board.

Eye-to-hand Calibration Validation A novel validation approach for the eye-to-hand calibration is proposed. It was not applied in this study because it was lately envisioned. The eye-hand calibration function in OpenCV works for both the eye-to-hand and eye-on-hand configurations according to what is input to the function.

- For eye-to-hand, where camera is fixed to ground and board is fixed to robot, it takes ${}^{cam}T_{board}$ and ${}^{grip}T_{base}$ and outputs ${}^{base}T_{cam}$
- For eye-on-hand, where board is fixed to ground and camera is fixed to robot, it takes ${}^{cam}T_{board}$ and ${}^{base}T_{grip}$ and outputs ${}^{grip}T_{cam}$

In fact, an eye-to-hand configuration can be seen as an eye-on-hand configuration if the board is considered as a camera fixed to the robot and the camera is considered as a board fixed to ground. ${}^{grip}T_{board}$ can be obtained by giving to the calibration function ${}^{board}T_{cam}$, obtained by transposing ${}^{cam}T_{board}$, and ${}^{base}T_{grip}$. This is useful because, while keeping the calibration board fixed to the gripper, the gripper can be manually rotated in roll, pitch or yaw(manually), and ${}^{base}T_{grip}$ is recorded. Then, this can be compared to the ${}^{base}T_{grip}$ obtained from ${}^{base}T_{cam} \times^{cam} T_{board} \times^{board} T_{grip}$, where:

- ${}^{base}T_{cam}$, from eye-to-hand configuration function
- ${}^{cam}T_{board}$, from camera
- ${}^{board}T_{grip}$, from the transpose of what eye-on-hand configuration function gives

7.2 Conclusion

The main reason of using the RGB-D camera was to grab a suspended block, which was clearly achieved in the video demonstration in chapter 5. Although the achieved grabbing follows a slow trajectory, is performed in a lab environment, and its accuracy lies within a rough margin of 2-3 cm, the aim of this study was to make a proof of concept of the idea of using a depth camera to visually assist the robot grab a block. Several challenges about the entire BrikieBot architecture were made clearer during this work. From the robot's side, the shape of the gripper makes the trajectory planning very tricky. Hence, the redesign of the gripper must be taken into consideration. From the camera's side, it is definitely very challenging to find an accurate and consistent pose for different orientations of the block. Starting with fluctuating pose readings from the ArUco markers, then eye-to-hand calibration errors and difficulty to measure them, and finally pose estimation code execution speed, along with camera fps limit.

Contributions The scientific and technical contributions of this thesis are summarized as follows:

- A proof of concept of the proposed grabbing approach in the Brikiebot project
- An open source GitHub repository for the eye-hand-calibration process with OpenCV, which can be used in any robotics project with the same hardware as this one.
- Integration of camera data into the robot's interface. This and the previous contribution will create a user manual for easy setup in all the future projects that use the robot and the camera.
- The results of this thesis will contribute to a conference paper that will be submitted to the IEEE International Conference on Robotics and Automation (ICRA) 2023. To increase this contribution, the work will continue during the summer break to improve the results obtained in chapter 6.

Learning Outcomes The learning outcomes of this thesis are listed as follows:

- In terms of programming, hands-on experience with OpenCV functions, ROS and Linux OS commands was acquired
- An excellent understanding of ArUco markers detection and pose estimation, along with its limitations
- A good understanding of the principles of stereo cameras. Particularly, Intel Realsense D455, an off-the-shelf camera used commonly in robotics projects.
- A good understanding of rigid bodies relative orientation and position.
- A good understanding of a collaborative robot's control concepts.
- A working knowledge of the motion capture system OptiTrack.

Bibliography

[1] Carlos Balaguer and Mohamed Abderrahim. *Robotics and Automation in Construction.* IntechOpen, Rijeka, 2008.

[2] Carolina Bianchi. Extracting contact surfaces from point-cloud data for autonomous placing of rigid objects. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2020.

[3] Cognex Corporation. Cognex website. Link.

[4] A. Deris, I. Trigonis, Andreas Aravanis, and Ellie Stathopoulou. Depth cameras on uavs: A first approach. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W3:231–236, 02 2017.

[5] Guoguang Du, Kai Wang, and Shiguo Lian. Vision-based robotic grasping from object localization, pose estimation, grasp detection to motion planning: A review. *ArXiv*, abs/1905.06658, 2019.

[6] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[7] Gergely Horváth and Gábor Erdos. Object localization utilizing 3d point cloud clustering approach. *Procedia CIRP*, 93:508–513, 01 2020.

[8] D. Hwang. Contour crafting-the emerging construction technology, 2005.

[9] Trung-Son Le, Quoc-Viet Tran, Xuan-Loc Nguyen, and Chyi-Yeu Lin. Solpen: An accurate 6-dof positioning tool for vision-guided robotics. *Electronics*, 11(4), 2022.

[10] Emanuele Garone Michele Ambrosino, Philippe Delens. Control of a multirobot bricklaying system. *Advanced Control for Applications*, 3, 2021.

[11] NCCR. Nccr digital fabrication (dfab) website. Link.

[12] António J. R. Neves, Rui Garcia, Paulo Dias, and Alina Trifan. Object detection based on plane segmentation and features matching for a service robot. *International Journal of Computer and Information Engineering*, 10(4):775 – 782, 2016.

[13] Krzysztof Okarma. Applications of computer vision in automation and robotics. *Applied Sciences*, 10(19), 2020.

[14] Omron. Technologies of forpheus. Link.

[15] Pickit-3D. Pickit-3d website. Link.

[16] Intel Realsense. Beginner's guide to depth. Link.

[17] Guinness World Records. The record breaking robot that teaches humans how to play table tennis. Link.

[18] Rafael Muñoz Salinas. Aruco: An efficient library for detection of planar markers and camera pose estimation. Link.

[19] Vladimir Tadic, Akos Odry, Ervin Burkus, Istvan Kecskes, Zoltan Kiraly, Mihaly Klincsik, Zoltan Sari, Zoltan Vizvari, Attila Toth, and Peter Odry. Painting path planning for a painting robot with a realsense depth sensor. *Applied Sciences*, 11(4), 2021.

[20] Mark Taylor, Sam Wamuziri, and Ian Smith. Automated construction in japan. *Proceedings of the Institution of Civil Engineers - Civil Engineering*, 156(1):34–41, 2003.

[21] OpenCV Website. Detection of aruco markers. Link.

[22] OpenCV Website. Opencv calib3d. Link.

[23] OpenCV Website. opencv::calib3d::handeyecalibrationmethod. Link.

[24] Zivid. Zivid website. Link.

Chapter 8

Annex

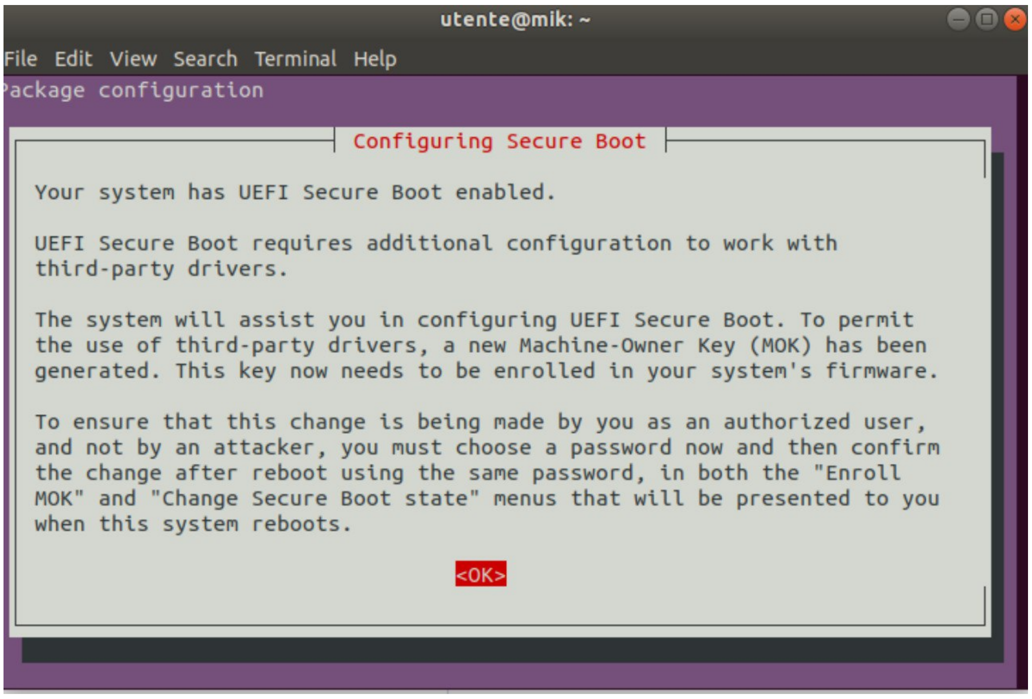
8.1 Additional Setup for Intel Realsense D455

This section mentions additional procedures related to the camera setup and getting familiar with the SDK.

SDK Installation and Setup

- Install SDK: link. Can start directly from the " Installing the Packages " section

  Camera not recognized on USB in Ubuntu 18.04 from first time and is recognized on other OS? If prompted with what is seen in Fig.(8.1) while installing dkms package, it means that you should disable secure boot. Click ok, then choose a key. After installation of package is finished, reboot, then you will be promoted with a blue screen where you choose enroll MOK and you enter the chosen key. If you did not enroll MOK, then uninstall the package and repeat the procedure. If still not recognized, check this link.



```
Package configuration

Configuring Secure Boot

Your system has UEFI Secure Boot enabled.

UEFI Secure Boot requires additional configuration to work with
third-party drivers.

The system will assist you in configuring UEFI Secure Boot. To permit
the use of third-party drivers, a new Machine-Owner Key (MOK) has been
generated. This key now needs to be enrolled in your system's firmware.

To ensure that this change is being made by you as an authorized user,
and not by an attacker, you must choose a password now and then confirm
the change after reboot using the same password, in both the "Enroll
MOK" and "Change Secure Boot state" menus that will be presented to you
when this system reboots.

<OK>
```

Figure 8.1: MOK Prompt

- Install Python wrapper: link. Note that if you are using python3[1], use:

  ```
  $ pip3 install pyrealsense2
  ```

  instead of

  ```
  $ pip install pyrealsense2
  ```

- Check python version

  ```
  $ python3 --version
  ```

- Check OpenCV version:

  ```
  $ python3
  $ import cv2
  $ cv2.__version__
  ```

- OpenCV not updating even with update and upgrade commands, use:

  ```
  $ pip3 install opencv-contrib-python3
  ```

[1]Difference in commands python vs python3: python3 instead of python and pip3 instead of pip

Important SDK Tools

- Launch Intel Realsense SDK: a quick way to check if camera is connected

  ```
  $  realsense-viewer
  ```

- Check available fps and resolutions for different stream profiles:

  ```
  $  rs-enumerate-devices
  ```

- Check available extrinsic and intrinsic camera information:

  ```
  $  rs-enumerate-devices -c
  ```

Running a c++ code with OpenCV on Ubuntu After installing the developer package from the installation page, try:

```
$  g++ filenmame.cpp -lrealsense2  ‘pkg-config --cflags --
   libs opencv4‘
```

or

```
$  g++ filename.cpp -lrealsense2
```

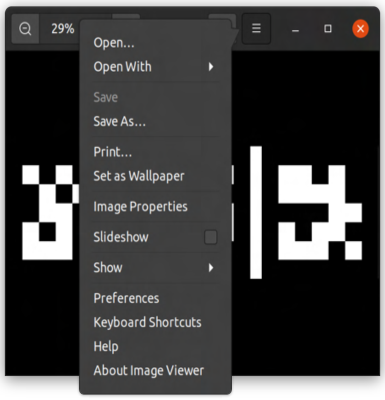
or

```
$  g++ -std=c++11 filename.cpp -lrealsense2
```

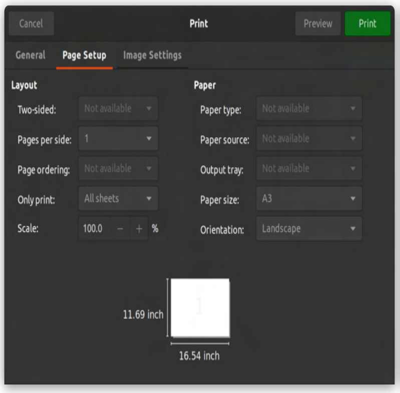
Extension USB Cable Any USB 3.0 super speed extension cable with a 5 Gigabits transfer rate should work

8.2 ArUco Board Creation in PDF on Ubuntu 20.04

After choosing the board's dimensions from the board creation script, choose to print the image from the menu in Fig.(8.2a).Then, choose the paper size to A3, as in Fig(8.2b). Finally, change units to millimeters and adjust them to what is expected, as in Fig.(8.2c). The example below shows a 3x1 board of marker size 100 mm with 10 mm separation, hence the width is 3x100 + 2x10 = 320 mm and the height is 100 mm. For reference, the OpenCV library in which the c++ sample code is given in here, but this study develops a similar code in python.



(a)



Layout

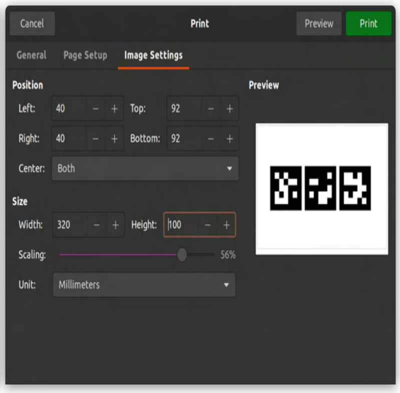
Two-sided: Not available
Pages per side: 1
Page ordering: Not available
Only print: All sheets
Scale: 100.0 %

Paper

Paper type: Not available
Paper source: Not available
Output tray: Not available
Paper size: A3
Orientation: Landscape

11.69 inch
16.54 inch

(b)



Position

Left: 40 Top: 92
Right: 40 Bottom: 92
Center: Both

Size

Width: 320 Height: 100
Scaling: 56%
Unit: Millimeters

(c)

Figure 8.2: ArUco board printing on PDF

8.3 Pose Estimation Validation with OptiTrack

OptiTrack is a motion capture system used for precise 6DOF position tracking, both in indoor and outdoor environments. The system is made up of a set of cameras that emit IR light, which, in its turn, is reflected by passive markers. As these markers can be detected by the camera sensors, they are used to construct a frame of any rigid body within the camera's working region. In an attempt to validate the estimated pose of a block with the OptiTrack system present at the University lab, the following points are noted:

- A block frame is constructed by gluing a set of reflective markers just next to the corners of the ArUco board, seen in Fig.(3.5b). So far, the camera reads board pose w.r.t. its frame and the OptiTrack system reads the constructed frame pose w.r.t. its chosen ground frame.

- A transformation must be done between the OptiTrack base frame and the camera frame. For this, a ChArUco board is used, as in Fig.(8.3). Before gluing the reflective markers to the board's corner, board pose is estimated w.r.t. camera frame. Then, the markers are placed at the corners and are used to define the ground frame of the OptiTrack system. That being done, all the measurements taken w.r.t. camera frame can be transformed to the ground frame, thanks to the pose estimated by ChArUco board. These measurements can be directly compared to what the OptiTack outputs. This should work in theory; however, the detected markers by the OptiTrack cameras were heavily blinking, the working space was not always available, lighting is a problem...etc. Eventually, the process was stopped due to its technical difficulties.

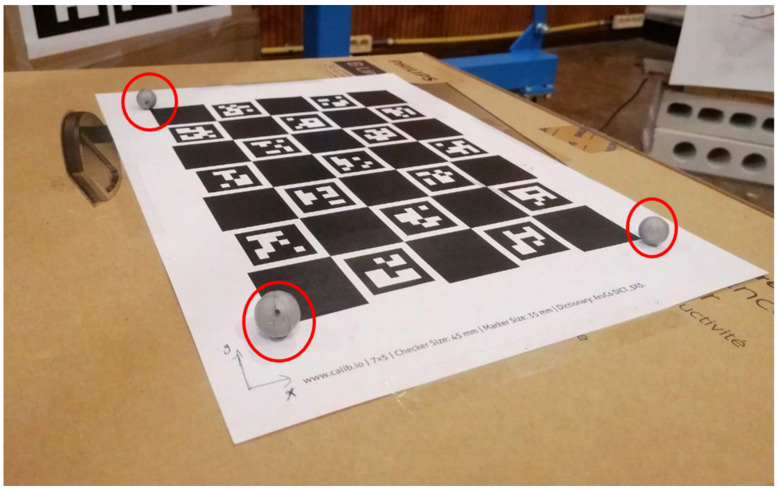
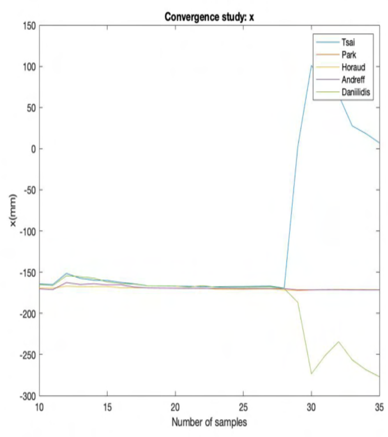
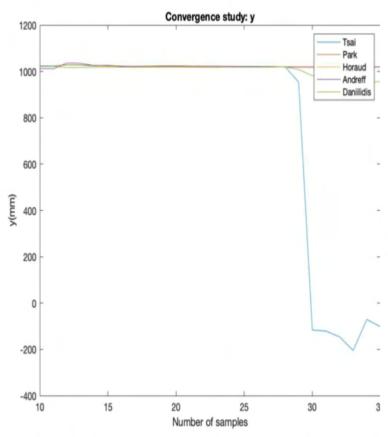


Figure 8.3: Proposed method to align OptiTrack ground frame w.r.t. ChArUco board frame by putting reflective markers, within red circles, at the corners of the board. Camera frame can then be transformed to board frame and hence the ground frame of OptiTrack

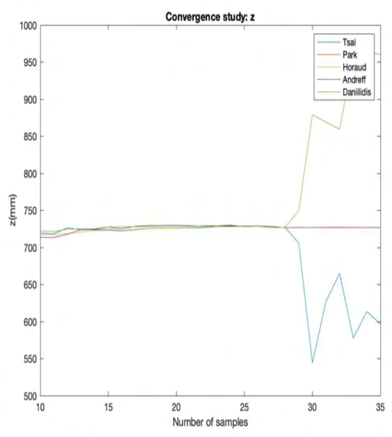
8.4 ArUco Board Eye-to-hand Calibration Convergence study



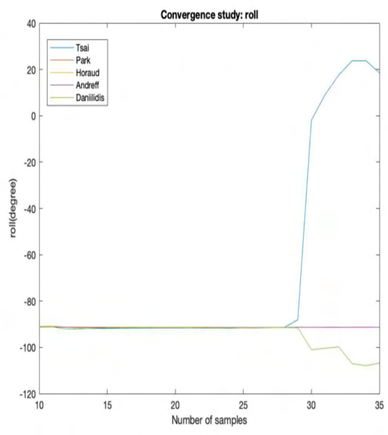
(a)



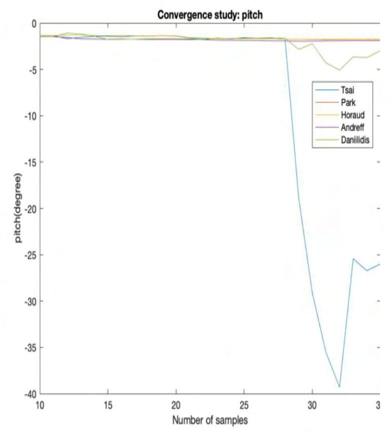
(b)



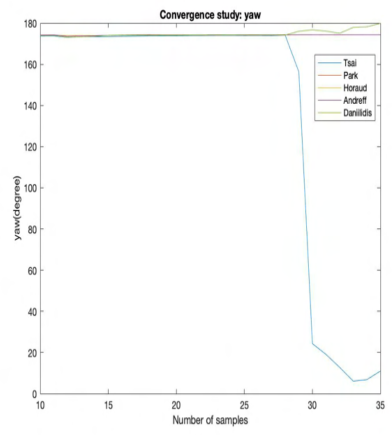
(c)



(d)



(e)



(f)

Figure 8.4: Convergence study of eye-to-hand calibration with ChArUco board for the 5 different methods offered by OpenCV. Only the methods of *Park* ,*Horaud*, and *Andreff* show a converging behavior