Universidade de Aveiro Departamento de Engenharia Mecânica
2018

Joana Beatriz Carvalho Mota

Bin-Picking de Precisão usando um Sensor 3D e um Sensor Laser 1D

Precision Bin-Picking using a 3D Sensor and a 1D Laser Sensor

Universidade de Aveiro Departamento de Engenharia Mecânica
2018

Joana Beatriz Carvalho Mota

Bin-Picking de Precisão usando um Sensor 3D e um Sensor Laser 1D

Precision Bin-Picking using a 3D Sensor and a 1D Laser Sensor

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica de Vitor Manuel Ferreira dos Santos, Professor Associado com Agregação do Departamento de Engenharia Mecânica da Universidade de Aveiro e de Miguel Riem Oliveira, Professor Auxiliar do Departemento de Engenharia Mecânica da Universidade de Aveiro.

agradecimentos / acknowledgements

É este o momento de tornar público o meu profundo agradecimento a todos os que, nos últimos cinco anos, me acompanharam no meu percurso universitário.

A minha primeira palavra vai para o Professor Vítor Santos por todo o profissional e dedicado acompanhamento, disponibilidade e sabedoria transmitida ao longo dos últimos três semestres, que influenciaram de forma significativa a minha opção por esta área de estudo. Agradeço também ao Professor Miguel Oliveira pelo apoio na ultrapassagem de diversos obstáculos.

Aos meus colegas do LAR por estarem sempre dispostos a ajudar e por terem proporcionado (a maior parte das vezes) bom ambiente de trabalho e cooperação. Em particular tenho de agradecer ao Bernardo Lourenço pela paciência e importante apoio durante todo o trabalho e ao Luís Sarmento pela constante presença com oportunas (e inoportunas) sugestões. Não esquecendo o eng. Rui Heitor que assegurou que no laboratório nada me faltasse.

Um grande obrigada à minha familia, em especial à Dona Mercedes e ao Senhor Vítor, por sempre me apoiarem e aturarem e por me assegurarem o jantar sempre que precisava, mesmo quando chegava de madrugada. Ao Gabriel por, apesar do desinteresse demonstrado, lá ter tentado de vez em quando fazer de conta que se interessava. Às primas por me oferecerem sempre o seu maior carinho nos meus momentos de maior stress.

Gostaria também de agradecer ao Miguel Gaspar por me ter ajudado na tarefa árdua de segurar o xadrez durante toda a calibração intrínseca da Kinect, como é possível de vizualizar na figura 4.9, e agradeço o facto de me ter ajudado e encorajado a atirar-me de cabeça nesta área.

Agradeço à Carmen, à Sara, ao Barros, ao Nunes, ao Carneiro e ao Hardcore pelo gratificante percurso de cinco anos e pelas longas e extensas discussões em inúmeros trabalhos de grupo.

And to you Liz thank you for everything and in particular for all the inspirational talks.

Palavras-chave *Bin-picking*, Fanuc LR Mate 200iD, Sensor Kinect, Sensor Laser, ROS, *ROS-Industrial*, *MoveIt*, Nuvem de Pontos, PCL, Precisão

Resumo

À tecnologia usada por um robô para agarrar objetos que estão dispostos de forma aleatória dentro de uma caixa ou sobre uma palete chama-se *bin-picking*. Este processo é de grande interesse para a industria uma vez que oferece maior autonomia, aumento de produção e redução de custos. O *bin-picking* tem evoluido de forma significativa ao longo dos anos graças aos avanços possibilitados pelo desenvolvimento tecnológico na área da visão, *software* e soluções de diferentes garras que estão em constante evolução. Contudo, a criação de um sistema versátil, capaz de agarrar qualquer tipo de objeto sem o deformar, independentemente do ambiente desordenado à sua volta, continua a ser o principal objetivo. Para esse fim, o recurso à perceção 3D é imprescindível. Ainda assim, a informação adquirida por sensores 3D não é muito precisa e, por isso, a combinação deste com a de outros dispositivos é uma abordagem ainda em estudo.

O objetivo principal deste trabalho é então desenvolver uma solução para a execução de um processo de *bin-picking* capaz de agarrar objetos pequenos e frágeis sem os partir ou deformar. Isto poderá ser feito através da combinação entre a informação proveniente de dois sensores: um sensor 3D (Kinect) usado para analisar o espaço de trabalho e identificar o objeto, e um sensor laser 1D usado para determinar a sua distância exata e assim se poder aproximar. Adicionalmente, o sistema desenvolvido pode ser acoplado a um manipulador de forma a criar uma unidade de perceção ativa.

Uma vez tendo um sistema global de sensores, os seus controladores e o manipulador robótico integrados numa infraestrutura ROS (*Robot Operating System*), os dados fornecidos pelos sensores podem ser analisados e combinados, e uma solução de *bin-picking* pode ser desenvolvida.

Por último, a fase de testes demonstrou, depois de alguns ajustes nas medidas do sensor laser, a viabilidade e fiabilidade do processo de *bin-picking* desenvolvido.

Keywords Bin-picking, Fanuc LR Mate 200iD, Kinect Sensor, Laser Sensor, ROS, ROS-Industrial, MoveIt, Point Cloud, PCL, Precision

Abstract

The technique that is being used by a robot to grab objects that are randomly placed inside a box or on a pallet is called bin-picking. This process is of great interest in an industrial environment as it provides enhanced automation, increased production and cost reduction. Bin-picking has evolved greatly over the years due to tremendous strides empowered by advanced vision technology, software, and gripping solutions which are in constant development. However, the creation of a versatile system, capable of collecting any type of object without deforming it, regardless of the disordered environment around it, remains a challenge. To this goal, the use of 3D perception is unavoidable. Still, the information acquired by some lower cost 3D sensors is not very precise; therefore, the combination of this information with the one of other devices is an approach already in study.

The main goal of this work is to develop a solution for the execution of a precise bin-picking process capable of grasping small and fragile objects without breaking or deforming them. This may be done by combining the information provided by two sensors: one 3D sensor (Kinect) used to analyse the workspace and identify the object, and a 1D laser sensor to determine the exact distance to the object when approaching it. Additionally, the developed system may be placed at the end of a manipulator in order to become an active perception unit.

Once the global system of sensors, their controllers and the robotic manipulator are integrated into a ROS (Robot Operating System) infrastructure, the data provided by the sensors can be analysed and combined to provide a bin-picking solution.

Finally, the testing phase demonstrated the viability and the reliability of the developed bin-picking process.

Contents

List of Figures

Chapter 1

Introduction

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

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

1.1 Framework and Motivation

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

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

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

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

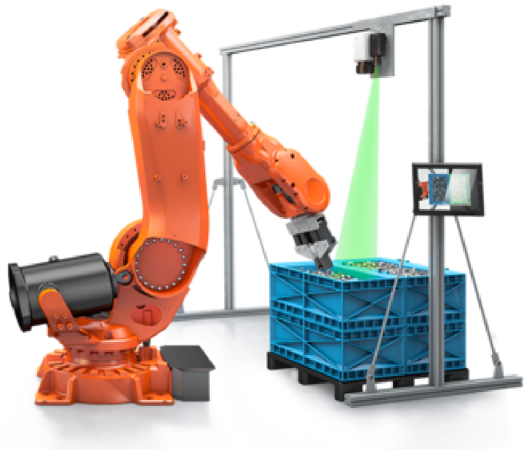


Figure 1.1: A bin-picking infrastructure example [5]

Together with vision systems and intelligent sensors, the robot is able to analyse a surface, detect the objects to be grabbed and move them to the desired location without the need to spend a lot of programming time. However, the process of detecting objects and identifying their correct positioning is a trivial task when carried out by a person, but of high difficulty when performed by a machine, making this one of the greatest challenges in the area of industrial robotics, especially because of the complexity of perception [6]. This challenge is due to the fact that, although robots are very advantageous when it comes to repeatability, the bin-picking process requires high accuracy in the face of an environment of disorder. The robot must be able to locate the piece in an unstructured environment where objects are constantly changing position and direction whenever something is removed from the box. This requires a delicate balance between the manipulator's dexterity, artificial vision, software, computing power to process all this information and the projection of a viable solution to the manipulator's grip in order to extract the parts without causing any damage and collision.

This technique has been researched by numerous organizations for over 3 decades, and one of the first attempts to create such a robust system was developed in 1986 at MIT [7]. Initially only 2D images were used, coupled with distance sensors, to acquire data, but the continuous technological evolution allowed for this to be done with 3D devices.

In order to improve the bin-picking technique, several problems need to be overcome [7]:

- Difficulties in determining the correct geometry, position and direction of the object due to being in a three-dimensional environment surrounded by randomly distributed pieces;
- Ensure adequate and constant lighting of the work space. If there is a large variation, there may be shadows that make it impossible to correctly detect the part to be collected;
- Cases of overlap and occlusion of parts;
- Correct determination of how to grab the part and the force to be exerted by pressing on it;

- Definition of the most adequate trajectory so as not to collide with other parts, containers and devices;
- For industrial implementation, cycle speeds must also be taken into account.

Presently there are 3 main forms of bin-picking:

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

The first two types presented are notoriously simpler processes to execute and can be quickly and easily implemented with technologies already on the market. In contrast, the Random Bin-picking process is considered the the ultimate goal for creating a vision-controlled robot, also known as VGR (Vision-guided robots) [7].

However, the creation of this type of versatile and precise system, capable of collecting any type of object without deforming it, regardless of the disordered environment around it, remains the main objective. Although several companies have already proposed different solutions to date, these are indicated to solve specific problems and are still not sufficiently versatile. In addition, most of the existing processes are used to grab non-fragile objects, due to the high degree of precision needed to avoid deforming sensitive objects.

1.2 Problem description

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

1.3 Objectives

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

The combined implementation of a 3D sensor with a 1D laser sensor will be the strategy to approach in order to execute this process with high precision. After their correct calibration, it may be possible to locate some types of surfaces (mainly planar) with great accuracy and thus allow for a better bin-picking process. This is achieved by combining the point-cloud with limited accuracy and reach provided by the 3D camera and accurately measuring the location of the point where the object is taken by a single-dimensional laser sensor (Sensick DT20 HI).

This system to be developed may be placed at the end of a manipulator and become an active perception unit for the detection of objects with high accuracy for further manipulation. Therefore, the first major objective will be the installation of the Kinect and Sick laser sensor in the FANUC LR Mate 200 iD manipulator and its connection to the acquisition and control system. Later, it will be necessary to integrate the global system of sensors, their controllers and the robotic manipulator into a ROS (Robot Operating System) infrastructure. Finally, an application demonstrating the accuracy of the bin-picking process will be developed.

1.4 Document Structure

This dissertation is composed of 8 chapters, including the present one, in order to meet the objectives mentioned above in a more organized way. Those chapters are arranged as follows:

1. **Introduction -** The current chapter in which is presented a brief framework, a description of the problem per se and an enumeration of the objectives.

2. **State of the Art -** Describes developed projects in the field of bin-picking and some solutions already commercialized in industrial applications;

3. **Experimental Infrastructure -** Introduces and describes all the hardware and software used to develop this bin-picking process;

4. **Calibrations -** Describes the calibration techniques and procedures performed to incorporate all the hardware and respective coordinate frames in an overall system;

5. **Segmentation of the Point Clouds -** Presents a detailed explanation of the Kinect data processing;

6. **System Integration -** Describes the steps required for the development and organization of the entire bin-picking process.

7. **Experiments and Results -** Describes all the experiments performed to demonstrate the versatility of the developed bin-picking system.

8. **Conclusions and Future Work -** Presents conclusions of the developed project and some suggestions for future work in this field of study.

Chapter 2

State of the Art

The creation of a bin-picking system, especially of the random type, has been one of the focus of several companies in the robotics industry and research organizations. There have been tremendous strides empowered by advanced vision technology, software, and gripping solutions which are in constant development. This chapter describes some recently developed projects in this field and some already commercialized solutions in industrial applications. The purpose of this chapter is not to present all types of bin-picking systems already developed, but rather explain ones that focus more in combining information from different sensors to perform a precise bin-picking process capable of grasping fragile objects without damaging them.

2.1 Industrial Implementations of Bin-picking Systems

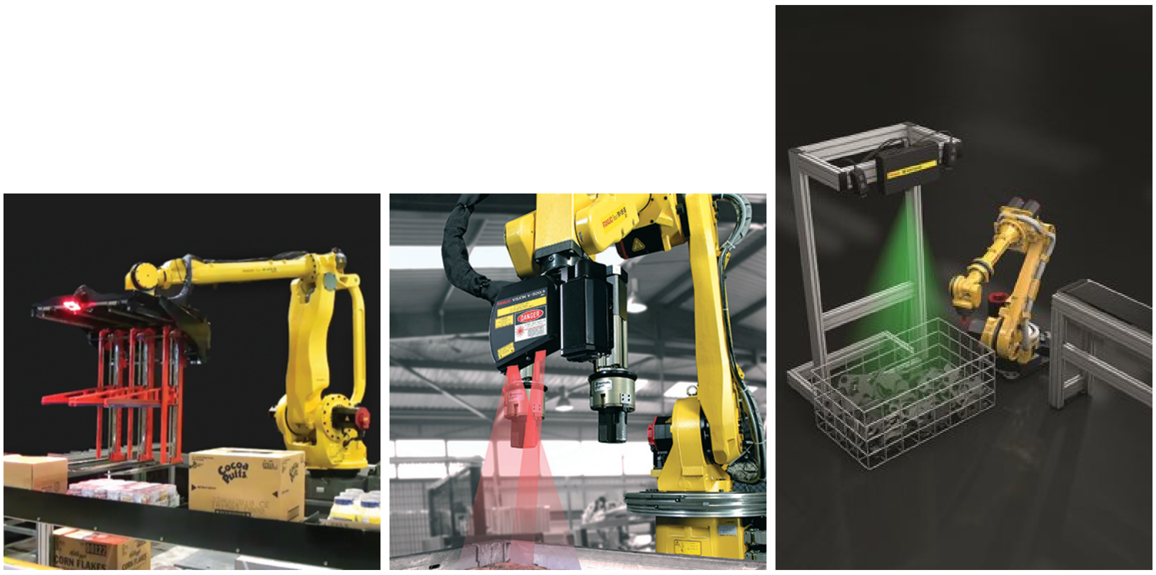
The number of bin-picking systems already implemented in the industry is very large. The ones presented next are here for the purpose of presenting some of the solutions with enough reliability to integrate an industrial process and to demonstrate the type of robust systems already in use in the market.

2.1.1 FANUC and its visual detection system iRVision

FANUC developed a plug and play visual detection system called *i*RVision. According to FANUC this type of system is capable of locating workpieces whatever their size, shape or position by using either 2D or 3D part recognition. It can also read 1D and 2D bar codes, sort according to colour and support flexible parts feeding, high-speed visual line tracking and bin or panel picking and placing. With this system is possible to increase the overall production flexibility and efficiency in the workplace. The *i*RVision can be fully integrated into an entire range of FANUC robots without complicated programming or expert knowledge [8]. This solution is suitable to various applications and industries such as Automotive, Food, Metal, Plastic, Aeromotive and Pharma industries.

FANUC *i*RVision supports several vision types such as 2D, 2.5D, 3D, 3D Laser and 3D Area Sensor. The 2D vision is capable of detecting objects positioned in one layer (X,Y,R) and 2.5D in two or more layers (X,Y,Z,R), both capable of picking up non-moving parts. The 2.5D vision system is represented in figure 2.1a . The 3D laser vision detects objects in position and posture by laser projection (X,Y,Z,W,P,R) and is presented in figure 2.1b .

Lastly, figure 2.1c presents the most robust system which incorporates a 3D area sensor for detection of objects by 3D map (structured light projection) (X,Y,Z,W,P,R). This last one can especially be used for high-end vision based bin-picking and material handling applications despite the parts conditions, like being dirty or rusty [9].



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

Figure 2.1: Different types of vision systems supported by FANUC *i*RVision [8].

In order to facilitate the selection and purchase of the users' desired vision system among the ones available, FANUC offers to test the vision process with the user environment. The FANUC's simulation software ROBOGUIDE enables an evaluation of the time, effort and feasibility of the entire process, allowing the user to select and modify parts and dimensions as required, before making a purchase decision [9] [10].

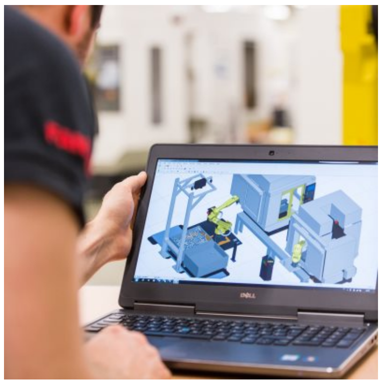


Figure 2.2: FANUC ROBOGUIDE robot simulator [10].

2.1.2 Pick-it

Pick-it is also a plug and play smart automation product at a fixed price. This robot vision system guides the client's robot to pick and place a wide range of products in different applications and scenarios. This system can be incorporated in different manipulators (FANUC, ABB, KUKA). According to Pick-it, this system is capable of identifying a wide variety of products. This can be done by using their FLEX detection engine which is capable of finding geometric shapes such as cylinders, boxes, planes and circles, as represented in figure 2.3, in which the operator chooses a shape and sets a range of sizes that need to be picked. Another solution is by using the TEACH detection engine, also provided, which can handle more complex 3D shapes after the operator takes a 3D picture of the product with the Pick-it camera so that the system can look for this shape [11]. Both vision engines offer the same tools to select parts from random bins by finding their position and orientation [12].

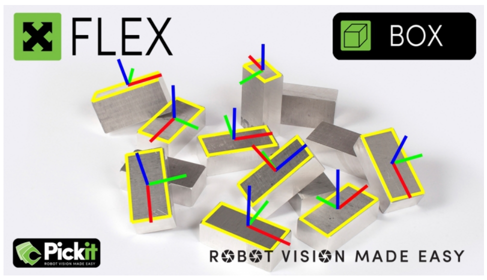
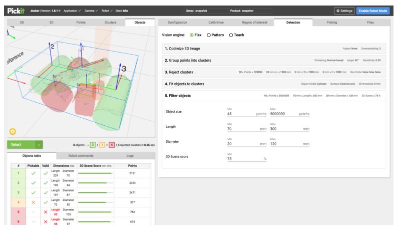


Figure 2.3: Pick-it's FLEX engine for finding geometric shapes [11].

The Pick-it product includes a 3D camera, the Pick-it software and an industrial processor. The 3D camera, presented in figure 2.4a, uses structured light, which projects a known pattern onto a scene, to compute the 3D image. This way it is capable of finding overlapping products of varying sizes and materials, even in changing and poor light conditions. It also contains an RGB camera which shows the current workspace for a better visualization of the process [13], but it is not used for perception. The technical data sheet of the camera can be found in [14]. The Pick-it software, figure 2.4b, has an easy to configure interface and there's no programming needed. The configuration of this type of application is done through a web interface in a Google Chrome web browser. One of the main advantages of the Pick-it solution is the capability of integrating their product in a ROS interface [15]. ROS is explained in the Software section of chapter 3. The Pick-it software comes pre-installed on an industrial processor with 12 cores at 3.5 GHz [16], presented in figure 2.4c.

(a) 3D Camera.


(b) Pick-it Software.

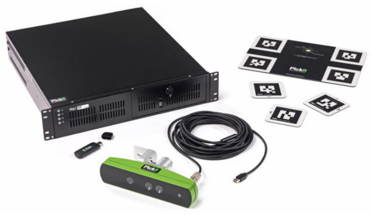

(c) Industrial Processor.

Figure 2.4: Pick-it product components [17].

These bin-picking systems and many others already implemented in the industry are very robust and precise, however, these types of services are often too expensive. Quite often once a user wants to implement this type of solution, offered by this sort of companies, he has to be able to pay the whole package for the system to work. He has pay for the hardware, software and, in cases like the Pick-it product, an industrial processor as well. The buyer is not only buying the system but is also paying for the brand. For instance, the 3D camera included in the Pick-it package product is an ordinary RGB-D camera but it is shielded by the brand, thus being more expensive than a normal one. Another example is the proposed FANUC's system. If a user wants to implement this kind of plug and play visual system to execute a task, he can select and purchase the package which is more adequate. However, if the user wants to perform a different task with the manipulator he most likely has to buy another different package for the purpose.

Conversely, research organizations have been investing in cheaper alternatives to replicate this kind of robust and precise bin-picking processes. Several entities made it possible by using off-the-shelf and open source hardware and software, easy to use and integrate into an overall system. Being bin-picking a process with so many variants to improve in order to create a sustainable solution, researchers have been tackling some of the main ones. Those include precise estimation of an object's position and pose, object recognition, grasp planning, among others, all in a bin/pallet full of randomly stacked objects. There are many papers and other academic work that focuses on refining each of these variants in particular. Since this project will focus on developing a precise bin-picking process capable of correctly grasping fragile objects without deforming them by combining the information of different sensors, the following applications present related solutions.

2.2 Related Academic Work

2.2.1 Projects that combine the information of different sensors

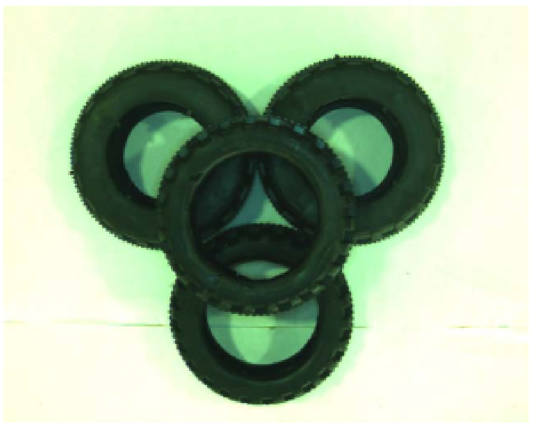
Bin picking of toroidal objects - developed in LAR - 2010

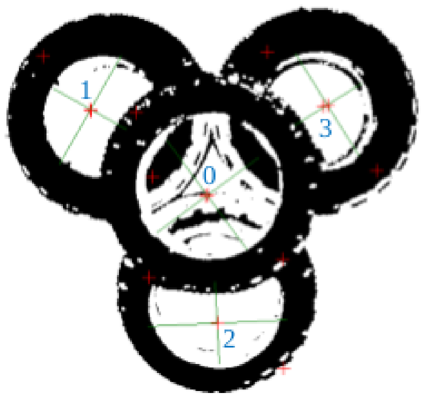
LAR is the Laboratory for Automation and Robotics located in the Department of Mechanical Engineering at the University of Aveiro where the former student Luís Rodrigues developed, in 2010, a bin-picking system for toroidal objects [18].

In this project Luís gave a manipulator the ability to grasp tyres, which have a toroidal form, by developing a vision system based on a monocular camera mobile configuration with local illumination. The JAI PULNIX TMC-1327G, present in figure 2.5, was the camera used for the environment perception and the image processing was performed with the Sherlock software. With both this hardware and software Luis is capable of identifying the different tyres and detecting their interior edges, since the robot's gripper will be grabbing the tyres by their interior surface.

Figure 2.5: PULNIX Camera.

Figure 2.6 presents an example of the processing applied on the image acquired by the camera, present in figure 2.6a, in which the relevant objects of the work area are identified.


(a) Image captured by the monocular camera.



(b) Detection of 4 candidate objects.

Figure 2.6: Image processing for the identification of the candidate objects [18].

However, as the camera doesn't give information about the depth, Luís also used a distance sensor to complement the information from the camera. The analog distance sensor *Sharp* GP2D12 was the one selected, presented in figure 2.7. This sensor uses infrared beams to do a scan of the environment and generate a distance pattern of the work area. Through a further analysis of this pattern it is possible to acquire information on the validation of the detected object and its distance to the sensor.

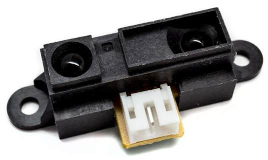


Figure 2.7: Distance Sensor *Sharp* GP2D12.

By combining both the information from the vision system and the distance sensor it was possible to send the manipulator to the exact position to correctly grab toroidal objects regardless of its orientation and dimensions.

One of the problems of using monocular cameras in object detection is the correct illumination of the working area. Luís tried to solve this problem by using a LED ring light source, but this does not always solve the problem completely. However, by using 3D cameras to obtain depth information of the workspace for object recognition, which is the most common procedure nowadays, there is no need for proper illumination.

A Multi-Resolution Approach for an Automated Fusion of Different Low-Cost 3D Sensors - Institute of Geodesy and Geoinformation, University of Bonn, Germany - 2014 [19]

This article uses low-cost 3D imaging systems to substitute the expensive measuring systems, that already exist used to acquire precise 3D representations of objects. Since common low-cost sensors have the limitation of a trade-off between range and accuracy, providing either a low resolution of single objects or a limited imaging field, this article attempts to resolve this problem by using multiple sensors. In order to achieve low-resolution scans of entire scenes with a high level of detail for some selected objects, the Microsoft Kinect and the David laserscanning system was used. The Kinect is also used in this project so it is fully described and explained in the Hardware section 3.1.2 of the Experimental Infrastructure chapter. The David lasescanning is a software package for low-cost 3D laser scanning [20]. It allows scanning and digitalizing 3D objects by using a USB-attached digital camera, a hand-held laser emitter with line optics and two plain boards in the background with a calibration pattern specified for the David software, which is normally referred as the "calibration corner". Figure 2.8 presents a David laserscanning system.

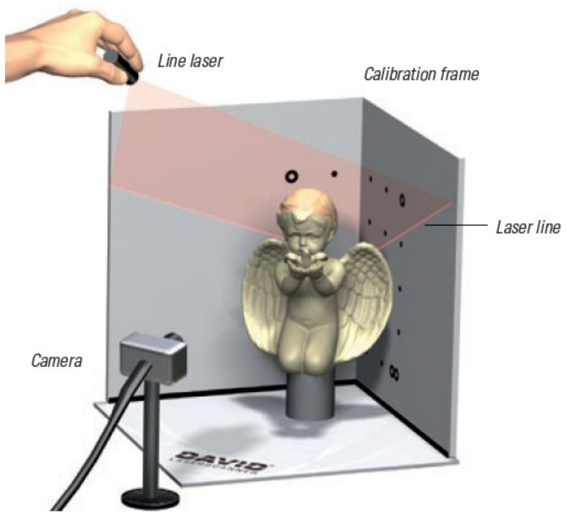
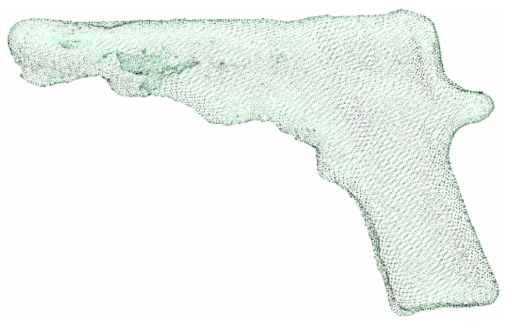


Figure 2.8: The David laserscanning system.

The signals of the two different low-cost sensors were separately recorded and subsequently combined. The Kinect sensor was used for the acquisition of the original scenarios, resulting in a low-resolution, wide-range point-cloud. To complement this low-resolution the David laserscanning system is then used for precise and highly detailed scans of single objects. This system is capable of capturing small details on objects with high resolution but in a severely limited measuring range of several decimeters.

By using the data from the Kinect and the laser scanning system a detailed 3D representation of single objects is possible without losing the original spatial arrangement of the scene. The object's position is given by the low-resolution Kinect scan, while a more detailed representation of the small details in the point-cloud is given by the David system.

The point cloud acquired by the Kinect sensor and the David laserscanning system is presented in figure 2.9a and 2.9b respectively. Figure 2.10 represents the result of the combination of both data.


(a) Point-cloud acquired by the Kinect.

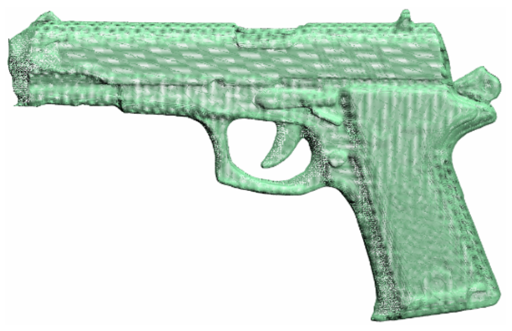

(b) Scan acquired by the David laserscanning system.

Figure 2.9: Corresponding point-clouds from the Kinect and the David system.

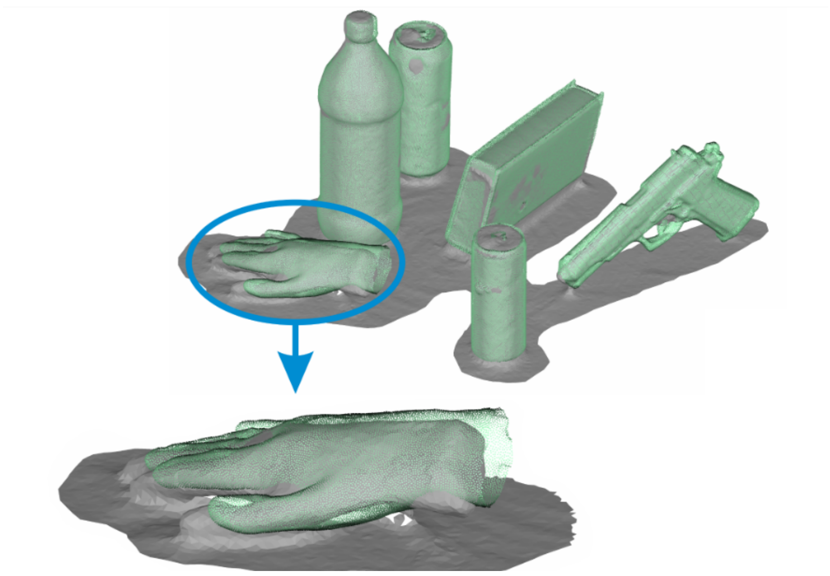


Figure 2.10: Combination of the Kinect (grey point-cloud) and the David system information (green point-cloud).

The David system, despite being very precise in representing objects with great detail, has the drawback of having a limited measuring range, thus every object would have to be scanned separately for an accurate representation. Therefore, this is probably not the most practical solution for a bin-picking process despite being very accurate.

Another possible solution for creating a precise bin-picking system capable of grasping objects without deforming them, besides combining the data from different sensors, is adapting the manipulator's gripper to a more appropriate one.

2.2.2 Projects that developed different types of grippers to catch fragile objects

Grasping complicated, small and soft objects has always been a challenge. However, robotic grippers have been improving over the years due to technological advances in this area [21]. Nowadays, there are vacuum grippers, adhesive, finger, hydraulic, magnetic, soft and more. The use of a gripper capable of grabbing deformable and fragile objects would be a great asset for a bin-picking process. This way any error from the vision system of the process can be prevented with a gripper capable of picking up objects without necessarily having to deform them.

Grippers capable of grabbing fragile objects have been emerging since the improvement of end effector sensors. Those grippers were mainly developed for food and medical industry, especially for the food industry because it is not always possible to have a CAD model of the object to catch or even make a 3D scan of the object since they have all different formats.

In [22], an end effector was designed for harvesting lettuce plants. This design included a 3 degrees of freedom manipulator, an end effector, a lettuce-feeding conveyor, an air blower, a machine vision device, six photoelectric sensors and a fuzzy logic controller. This last one is a control system that analyzes analogue input values in terms of logical variables that take on continuous values between 0 and 1. The fuzzy logic control was applied to determine appropriate grip force on lettuce plant according to leaf area, index and height. The designed end effector was able to harvest lettuce at a rate of 5 seconds per plant with a success rate of 94.12%.

An enclosed hygienic food gripper was designed with force feedback sensors for flexible production, in [23]. The gripper contains 2 fingers: one is stationary while the other moves by magnetic attraction. The actuator is placed inside with an inner magnet, while an outer magnet moves the finger on the outside of the container, as shown in figure 2.11. The product's location, orientation and dimensions are extracted by a vision system to aid the gripping process. The gripper was, therefore, able to handle an in-feed mixture of tomatoes, apples, carrots, broccoli and grapes without intermediate adjustments.

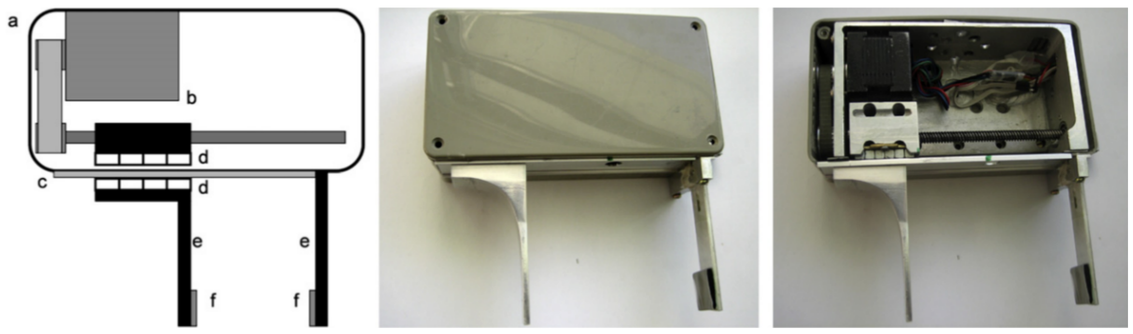
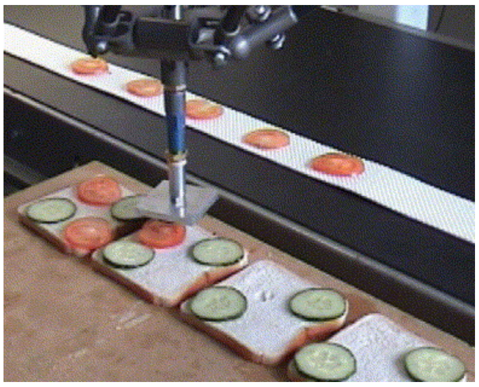


Figure 2.11: Gripper with force feedback for handling different types of fruits and vegetables [23].

A Bernoulli principle type end effector was designed for handling delicate sliced fruit and vegetable products commonly found in the food industry [24]. It allows the objects to be lifted using the airflow over the surface to minimize contact. This then reduces the probability of cross-contamination and damage to the sliced product. Besides the contamination, another benefit of using this type of gripper is to reduce the amount of surface moisture on the object. Figure 2.12b and 2.12a present the gripper's prototype and the same mounted on a robot, respectively.


(a) Gripper mounted a manipulator.

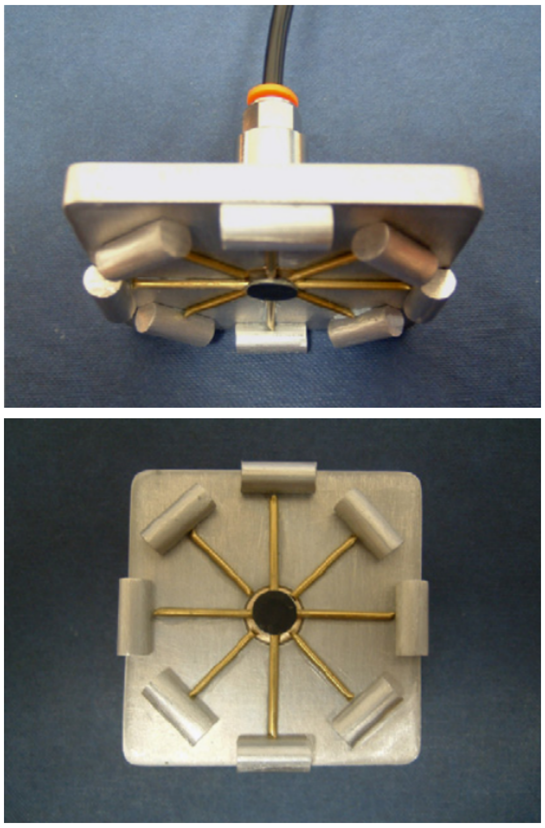

(b) Gripper's prototype.

Figure 2.12: Gripper developed for the handling of delicate sliced fruit and vegetable [24].

After numerous analyses and research on several types of robotic grippers, the paper in [21] concludes that the most common grippers for fragile and irregular objects are the attractive grippers. This attraction can be vacuum suction, magneto-adhesion or electro-adhesion. However, grippers based on force feedback and soft grippers are also very common for these type of objects. Figure 2.13 presents an example of a soft gripper, which is very much in use nowadays.

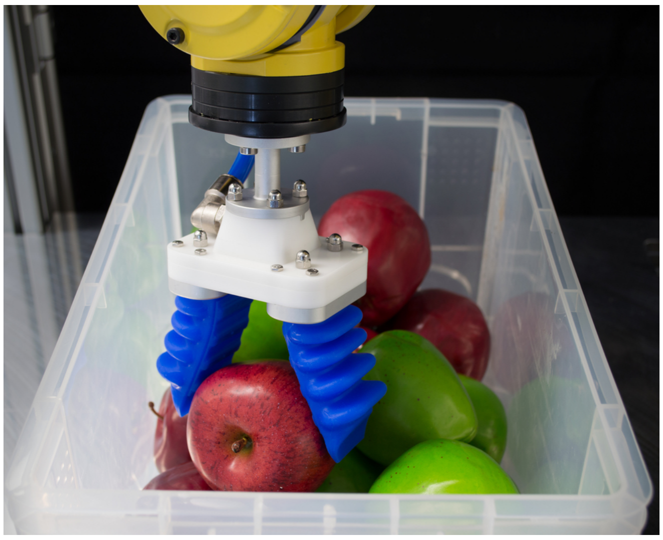


Figure 2.13: Example of a soft gripper [25].

Therefore, by combining a soft gripper or an attractive gripper, for example of the vacuum suction type, with a feasible and off-the-shelf vision system, that uses different sensor data, it will then be possible to replicate a precise bin-picking process for fragile objects.

Chapter 3

Experimental Infrastructure

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

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

3.1 Hardware

3.1.1 Robot FANUC LR Mate 200iD

As bin-picking is a technique used by a robot to grab objects, the use of a robotic handler is imperative. Additionally, it has to have the flexibility to position itself in various configurations in order to correctly collect different types of objects.

The robot used in this project is the FANUC LR Mate 200iD represented in the figure below. This model is a compact 6 axis robot with the approximate size and reach of a human arm, more precisely 717 mm. It has a payload of 7 kg, is capable of reaching velocities of $4\,\mathrm{m\,s^{-1}}$ and has a repeatability of ± 0.018 mm. Weighing only 25 kg, LRMate 200iD offers huge locational versatility including in extremely narrow spaces [26].

Figure 3.1: Fanuc LR mate 200iD [26].

For this project, the robot was equipped with a suction gripper, one of the most common process of bin-picking. This type of gripper is also preferred due to only requiring one grasping point to pick up an object.

The robCOMM language

For easier communication between the computer and the industrial manipulator the robCOMM language was used for simple tests and movements. By using this language and a TCP/IP communication, it is possible to exchange messages with the robCOMM server running in the robot. This language specifies a set of instructions which indicates to the server what type of task to perform, thus controlling the robot's behavior.

3.1.2 Kinect Sensor

The Kinect is and add-on device produced by Microsoft for Xbox 360, Xbox One and Windows. This product was an attempt to broaden the audience for console beyond its typical gamer base and it was an immediate success.

Figure 3.2: Microsoft Kinect [27].

In 2008, Microsoft licensed a chip developed by the Israeli startup PrimeSense which created the basis behind the Kinect. The chip manages audio and visual information independently and both are accessible via the USB connection [28].

The first-generation Kinect for Xbox 360, which is the one used in this project, was introduced in November 4th of 2010. Microsoft shipped 8 million units in the first quarter, making it one of the fastest selling tech product debuts in history [29].

This device is mainly a RGB-D sensor which is capable of combining RGB color information with per-pixel depth information. The critical parts that make up the Kinect sensor are represented in the figure 3.3 and are the following:

An RGB camera that stores three channel data (red, green and blue);

An infrared (IR) emitter and an IR depth sensor. The emitter projects infrared light beams and the depth sensor reads the IR beams reflected back to the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor creating a 3D image of the scene.

A multi-array microphone that contains four microphones for capturing sound. Therefore, it is possible to set the location of the sound source and the direction of the audio wave while recording it.

A 3-axis accelerometer to determine the current orientation of the Kinect.

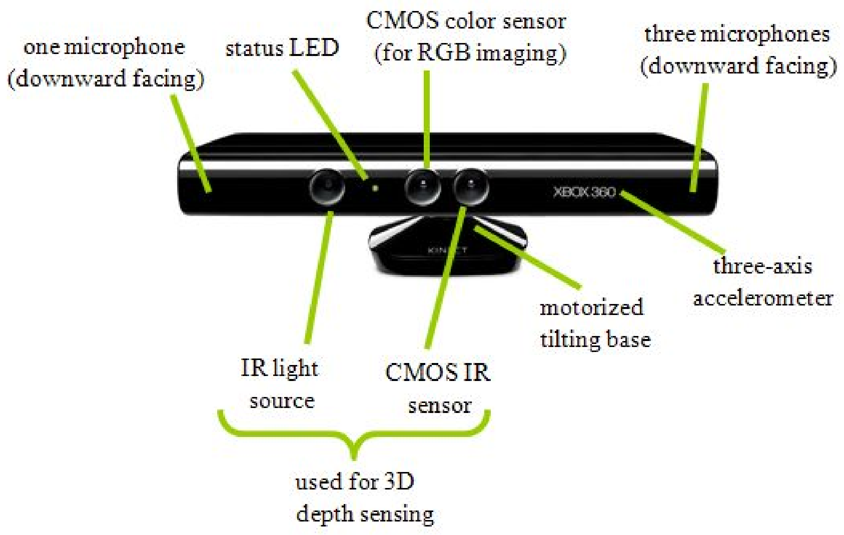

one microphone (downward facing)
status LED
CMOS color sensor (for RGB imaging)
three microphones (downward facing)
three-axis accelerometer
motorized tilting base
IR light source
CMOS IR sensor
used for 3D depth sensing

Figure 3.3: Kinect Sensor Components [30].

The characteristics of the Kinect are listed below [31].

- **Characteristics**:
  - Depth Sensor Range: min 800mm and max 4000mm;
  - Viewing angle: 43° vertical by 57° horizontal field of view;
  - Vertical tilt range: ±27°;
  - Frame rate: 30 FPS(frames per second);
  - Resolution depth stream: 640 x 480 (default), 320x240 or 80x60;
  - Resolution color stream:1280x960 at 12 FPS, 640 x 480 at 15 FPS or 640 x 480 at 30 FPS(default).

- **Operation Mode**:

  The PrimeSence technology used in the Kinect uses an infrared laser to project a pseudo-random beam pattern. The projected pattern, which is an image of the dots that are reflected off of the objects, as in the figure 3.4, is then captured by an infrared camera. The receiver contains a monochrome high resolution CMOS image sensor. Afterwards, the software within the Kinect is responsible for figuring out how much the dot pattern has been distorted, since these types of patterns had already been captured previously at known depths. The depth of each pixel is estimated based on which reference patterns the projected pattern matches best. All this happens at 30 FPS. The depth data provided by the infrared sensor is then correlated to a calibrated RGB camera. Finally, it is possible to generate an RGB image with a depth associated with each pixel.

A common representation of this type of data is a point cloud. Point clouds are a set of data points in a three dimensional space, in which each point can have additional features, such as an associated color.

Since the Kinect uses IR, the Kinect will not work in direct sunlight, for example in the outdoors.

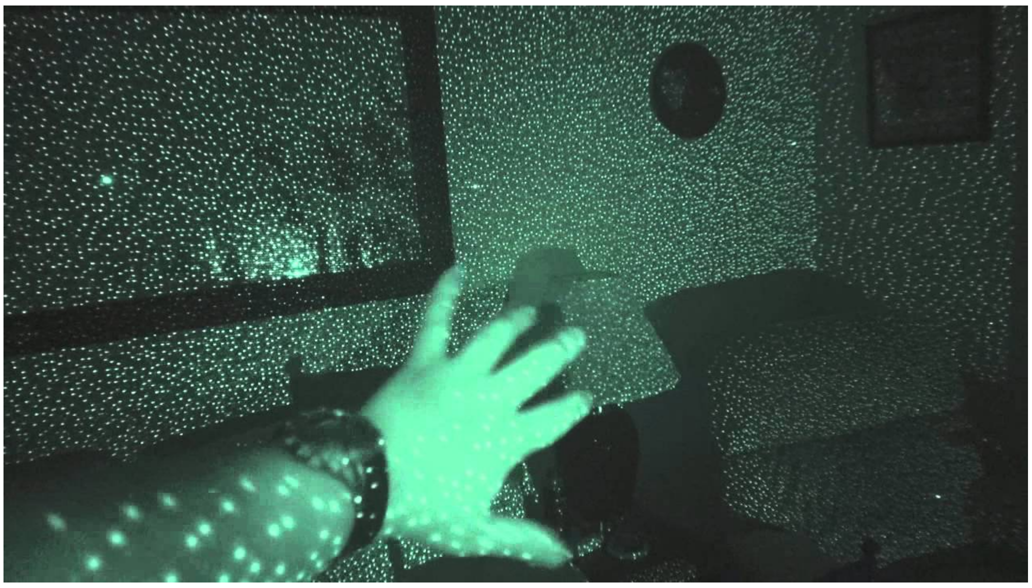
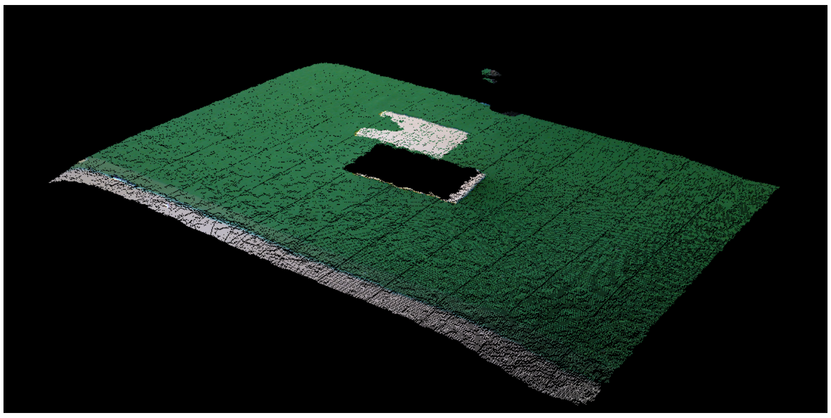


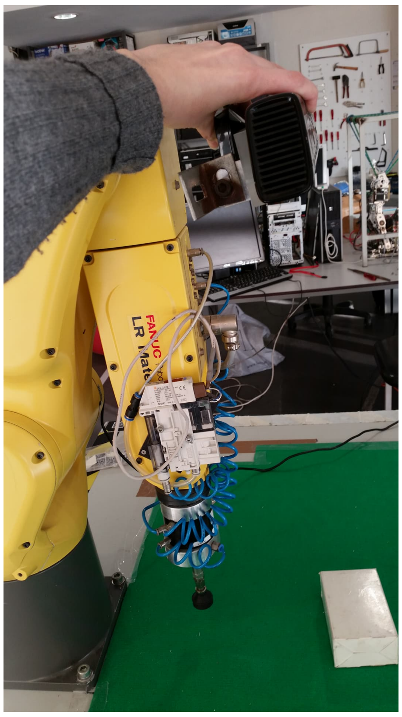
Figure 3.4: The laser grid used by the Kinect to calculate depth.

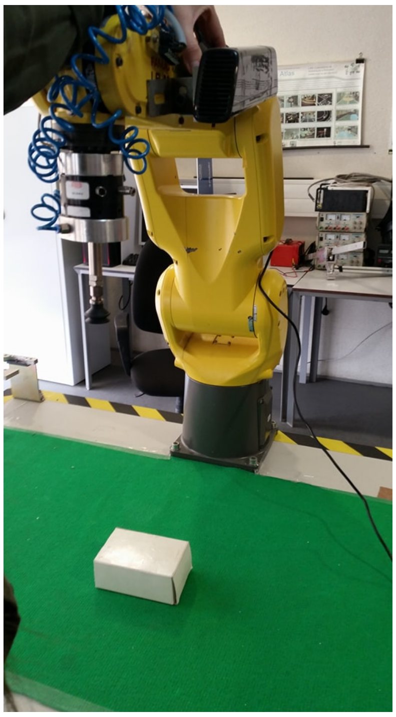
Support for the installation in the Robot

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

In order to decide the correct position to install the Kinect, it was necessary to analyze what the sensor saw in each possible position. To do so, some frames of the point cloud acquired by the Kinect in the different positions were recorded. Figures 3.5, 3.6 and 3.7, present each position tested and the corresponding point cloud. The number of degrees of freedom (DOF) that the Kinect should and could have were taken in consideration when choosing the most appropriate positions. Those will vary according to the joint in which the Kinect is attached to. Although having more degrees of freedom would be better, that would mean the Kinect should be attached to the robot's end effector. Since the Kinect head is 279.4 x 63.5 x 38.1 mm in width, depth and hight respectively, it would not be advisable to have such a big device at the tip of the manipulator because it would probably hit other objects while grabbing the desired ones.

A.1- 

A- 

B- 

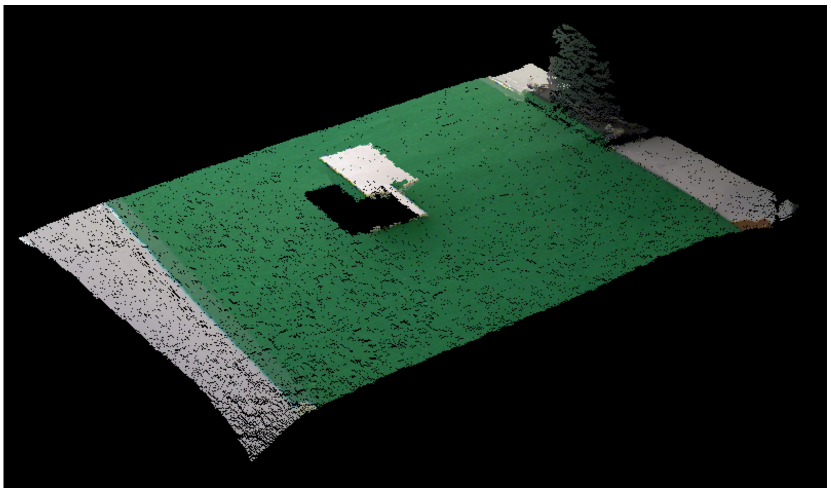
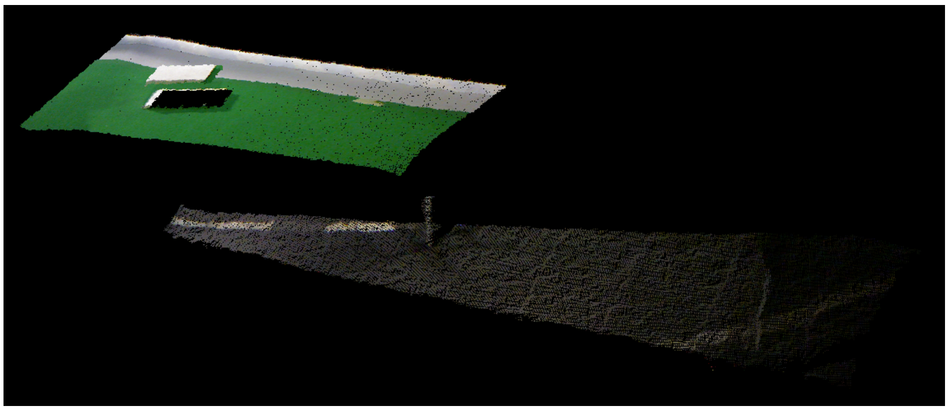
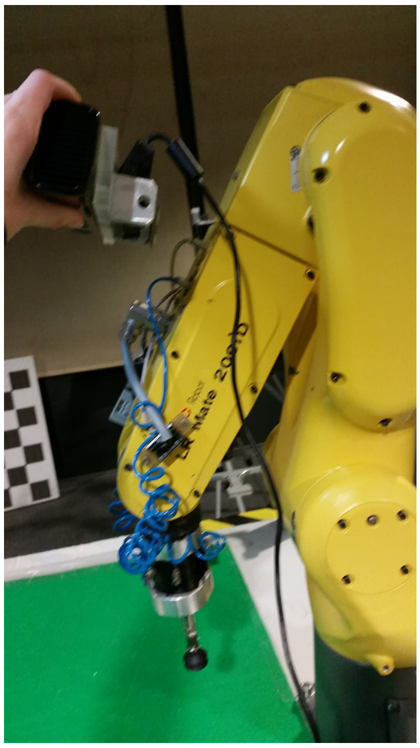
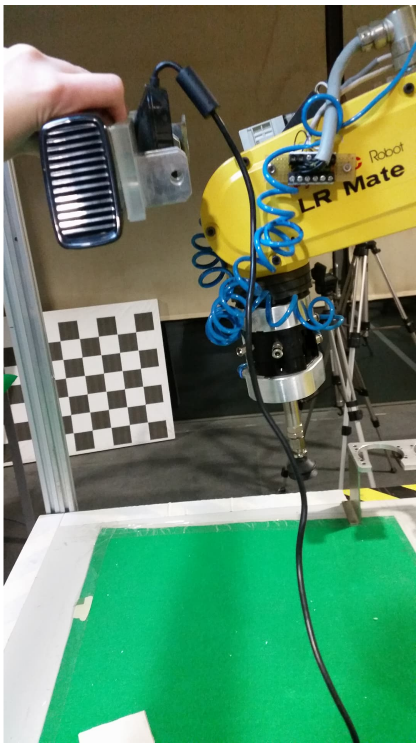
B.1- 

Figure 3.5: A- Kinect attached to the 3th joint with 3 degrees of freedom and the respective point cloud(A.1), B- Kinect with 4 degrees of freedom on the side of the robot's 4th joint and the respective point cloud(B.1).


C.1-



C-



D-

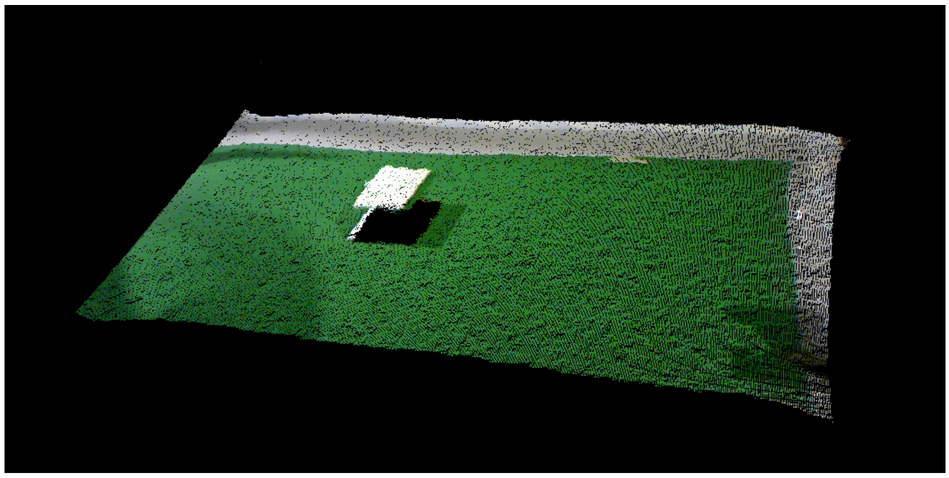
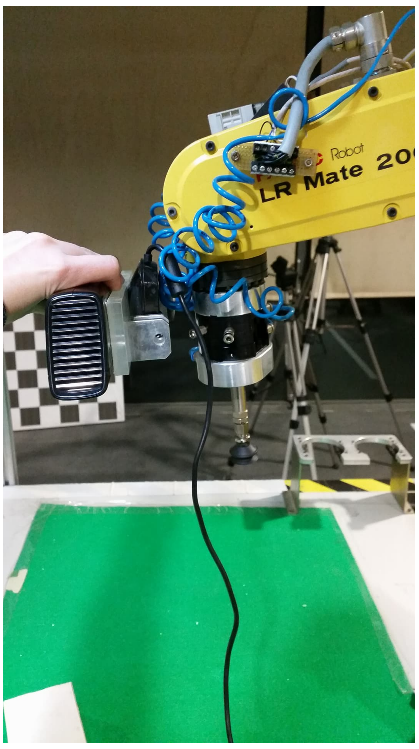

D.1-

Figure 3.6: C- Kinect with 4 degrees of freedom closer to the 3th joint of the robot, D- Kinect with 4 degrees of freedom closer to the 5th joint of the robot.

E-


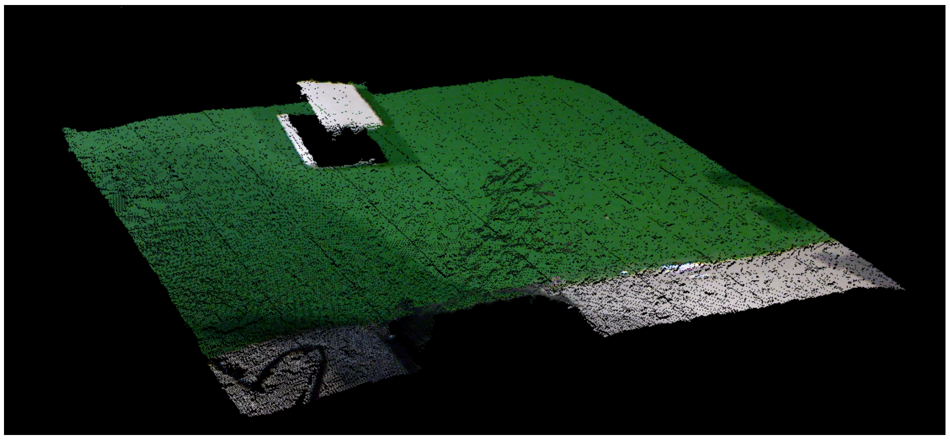
E.1-


Figure 3.7: E- Kinect with 6 degrees of freedom attached to the end effector.

After analyzing all the possible positions, the most adequate was selected. The one in which the Kinect has 4 degrees of freedom and is attached to the side of the robot's arm was chosen as the most suitable. This option is represented in figure 3.5 B and was the one chosen for the combination of the following reasons. By having the Kinect in this location the robot does not need to move to a position closer to the workspace limits to enable the camera to take a frame that contains the whole workspace. The frame acquired by the Kinect in option B represents a closer workspace in which the robot can reach every point. Besides, in this position the dimensions of the Kinect will not interfere with the movements of the manipulator.

In order to attach the Kinect to the robot's arm, a structure which could hold the device and be screwed to the steel sheet that protects the manipulator, was modeled. This support is represented in the figure 3.8 below.

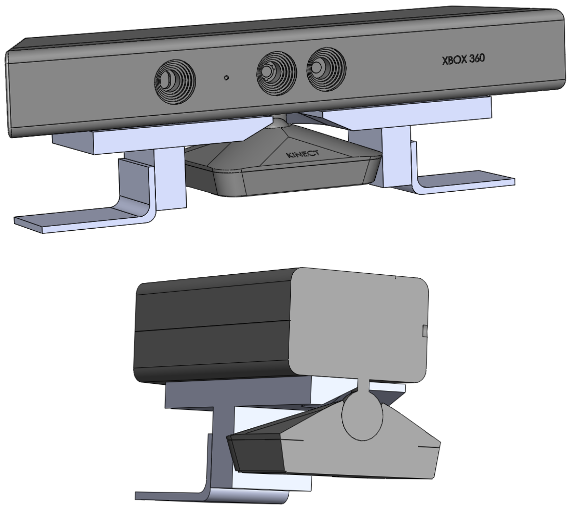


Figure 3.8: First support to attach the Kinect to the manipulator.

To safely fix the support to the robot's arm, at least two holes had to be drilled in the steel sheet. When the arm of the robot was unscrewed it was possible to observe that there was not enough space to safely drill two holes to secure the support. The interior of the robot can be seen in figure 3.9.

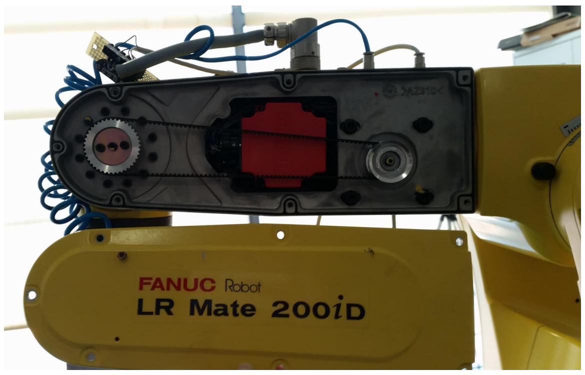


Figure 3.9: Robot's interior.

Therefore, another support was designed to avoid drilling the robot. The final model is represented in the figure 3.10 and is composed by three main parts. The largest part which is glued to the lower side of the Kinect's head; the two cobbles that allow the device to have the necessary height to give space to the base of the Kinect, and the two parts that will be screwed to the arm. These last parts have the correct dimensions in order to take advantage of the holes present in the steel sheet. Consequently, only 4 screws have to be replaced by longer ones to correctly fix this support. All these parts are connected with one screw in each side.

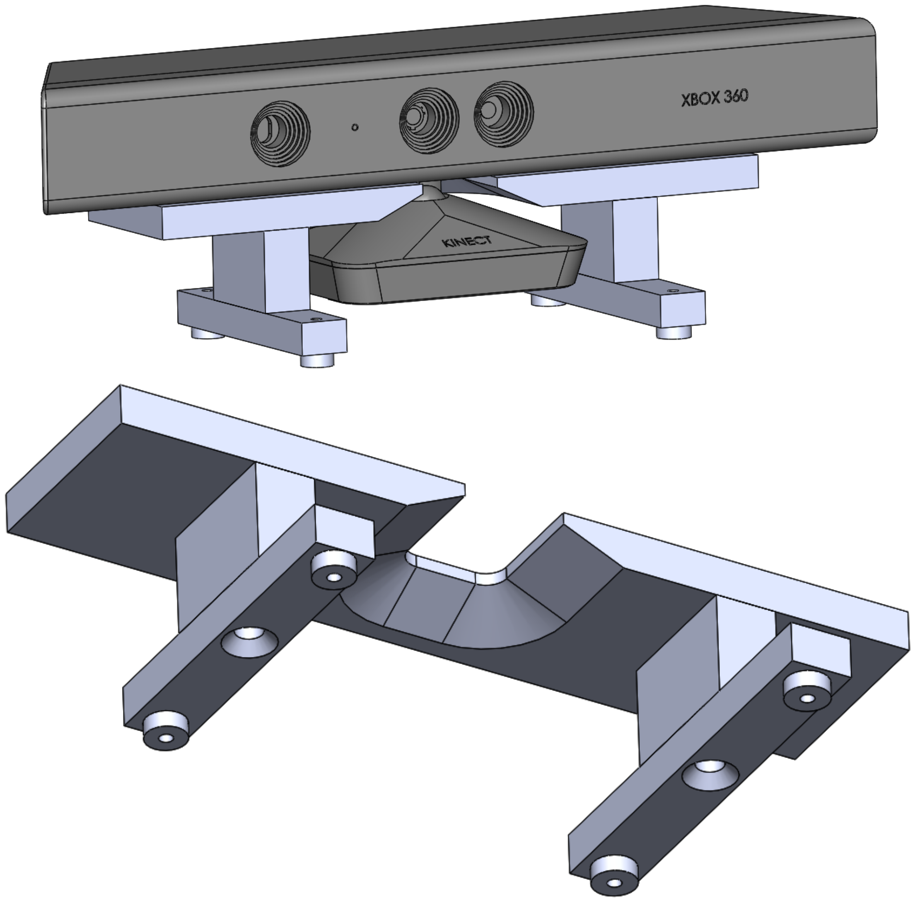


Figure 3.10: Final Support for the Kinect.

Afterwards, this model was manufactured in acrylic and the Kinect was correctly installed. The pictures bellow show the support already fixed to the arm of the robot and the Kinect glued to the structure.

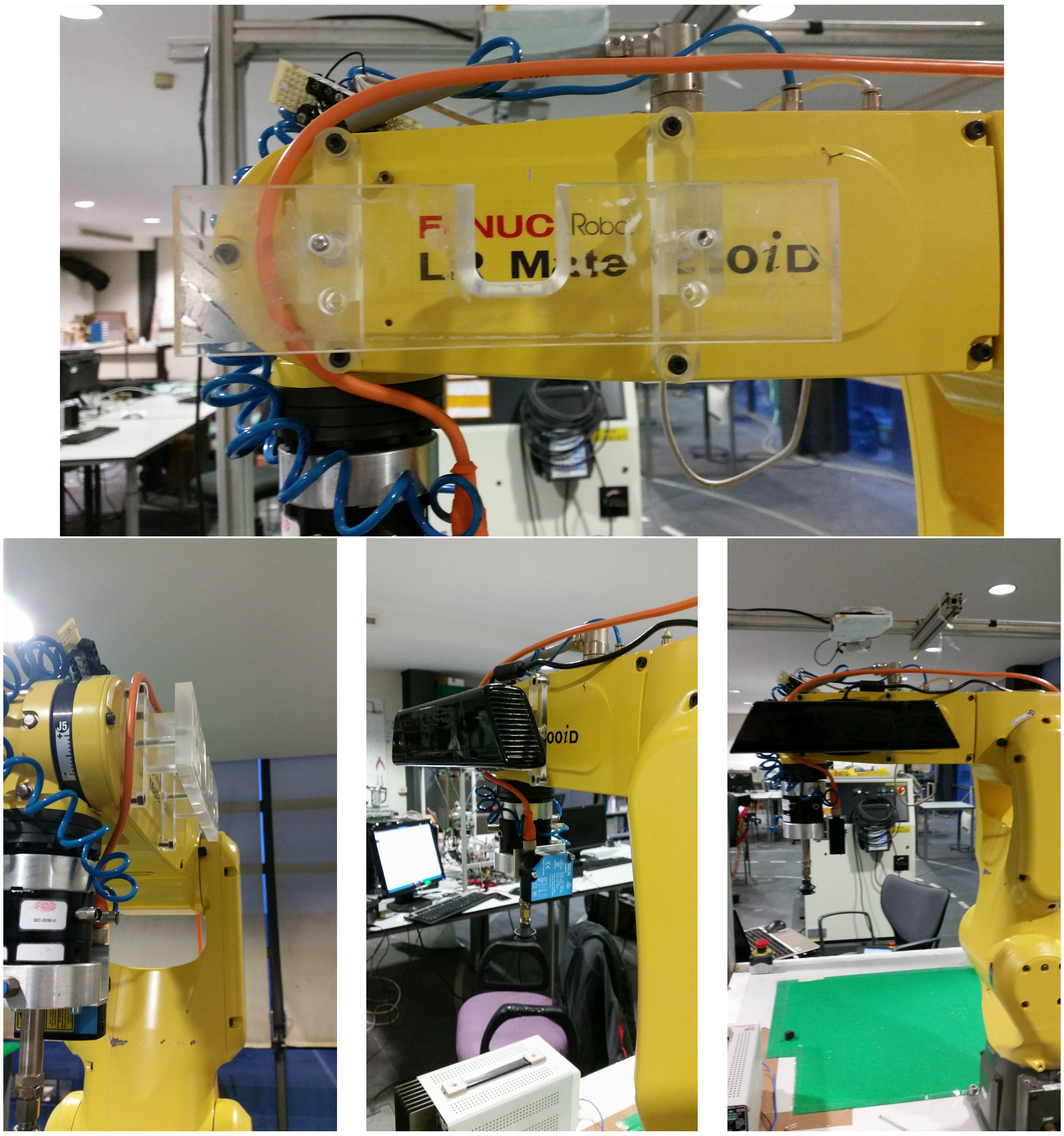


Figure 3.11: Kinect and Support fixed to the Robot.

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

3.1.3 Laser Sensor DT20 Hi

Figure 3.12 shows a picture of the 1D laser sensor used in this project. The DT20 Hi sensor is an opto-electronic sensor, developed by SICK [32],and is used for optical determination of object distances without contact. The sensor includes a red laser transmitter with a wavelength of 655 nm, which is capable of measuring a precise distance regardless of the surface's roughness of the target.

Figure 3.12: SICK DT20 HI [33].

The main attributes os this type of sensor are the following.

- **Characteristics**:
  - Measurement Range : 100 mm - 1.000 mm;
  - Resolution : 1.000 µm;
  - Linearity : ± 6 mm;
  - Response time : 2,5ms / 10 ms / 40 ms, according to configuration applied in the sensor's menu (fast/medium/slow);
  - Light spot size (distance) : 6 mm x 12 mm (1000 mm - maximum range);
  - Supply voltage : 10V - 30V
  - Analog output : 4mA - 20mA;
  - Resolution Analog Output: 12bit.

  Other characteristics, such as the exact dimensions of the device, can be found in the sensor's data sheet [34].

- **Operation Mode**:

  An opto-electronic sensor from SICK converts optical information into electric signals that can be evaluated. This type of sensors belong to a bigger family of displacement sensors.

  A Displacement Sensor is a device that measures the distance between the sensor and an object by detecting the amount of displacement through a variety of elements and converting it into a distance. There are several types of sensors, such as optical displacement sensors, which is the case of the DT20 Hi, linear proximity sensors, and ultrasonic displacement sensors [35].

  The optical displacement sensors uses a triangulation measurement system. Some sensors employ a PSD, and others employ a CMOS (CCD) as the light receiving element [35]. The DT20 Hi uses a CMOS receiving element [36].

  Laser triangulation is a measurement procedure that measures an object directly without contact. A light beam is projected to an object to be measured. The reflections of the beam are depicted on a light-sensitive element by the reception optics. Depending on the distance of the object, the position of the depicted light spot changes. As a result, the distance to the measurement object can be determined very precisely for small distances by adding the light reflection to the geometric calculations [37]. This process is represented in figure 3.13.

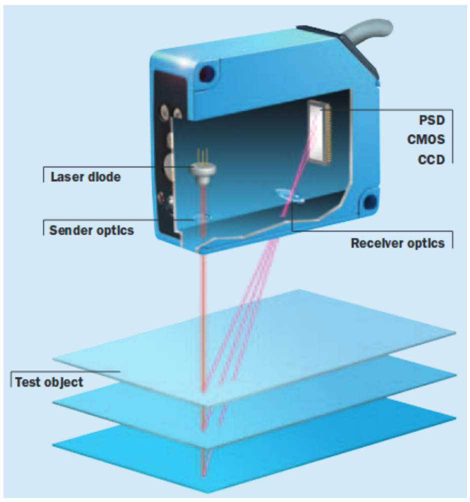


Figure 3.13: Triangulation Measurement [38].

  Compared with a sensor that employs the PSD as the light receiving element, a sensor that employs a CMOS (CCD) provides a more accurate measurement of displacement without being affected by surface color and texture of objects [35].

- **Connections**:

  This SICK's sensor contains two output signals, an analog output (Q_A) and a switching output (Q), and also a multi-function input (MF). The connection diagram is represented in figure 3.14. The analog output, which behaves as a current source, was the one used in this project. Although the current range by default covers the whole range of the sensor, it is possible to configure it in the build-in menu. This way it is possible to set a maximum and a minimum distance in which the current will vary between 4 mA an 20 mA. By narrowing the distance range the readings will be more precise because the 16 mA will spread within a smaller scope.

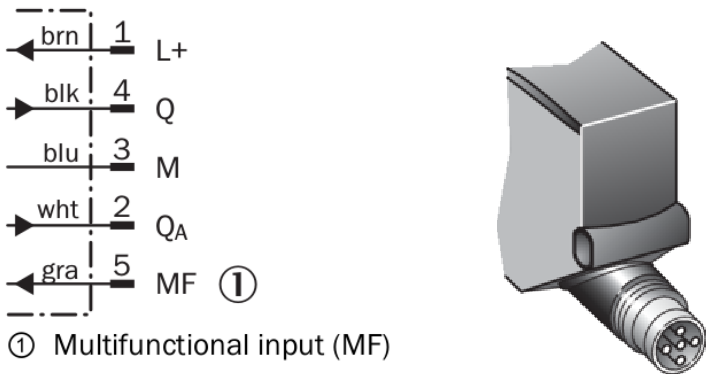


Figure 3.14: Connection diagram and type [33].

The microcontroller used to interpret and send the readings of the laser sensor was an Arduino UNO, which is described in the next section. Since the output signal of the sensor behaves as a current source and the analog input of the Arduino UNO does a voltage reading, it was necessary to convert the output current of the sensor to a proportional voltage, by using a resistor in series. Knowing that the Arduino UNO has an operating voltage of 5V and in order to use its entire range, the most adequate resistor to convert the current was calculated. To do so, the following equation and the maximum current outputted by the sensor (20 mA) were used.

$$R = \frac{5V}{0.02A} = 250\Omega \tag{3.1}$$

A 250Ω resistor was then the one used to convert the signal. With all this in mind, a schematics with the main connections, presented in figure 3.15, was created for a better understanding.

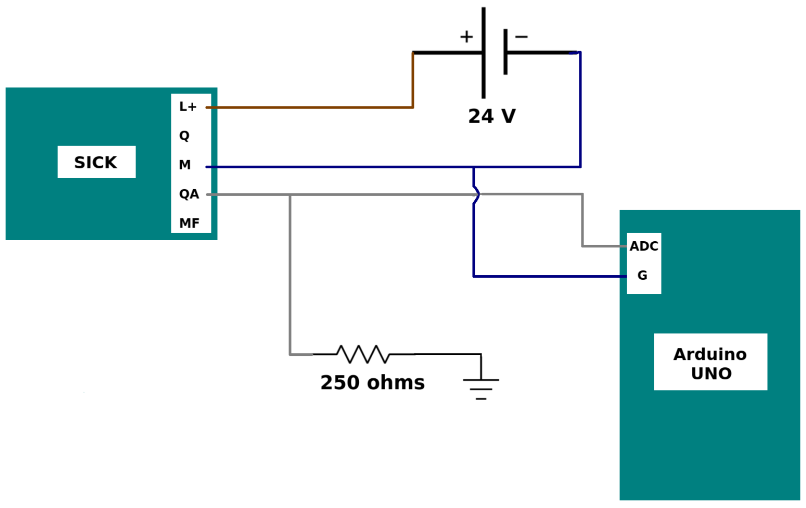


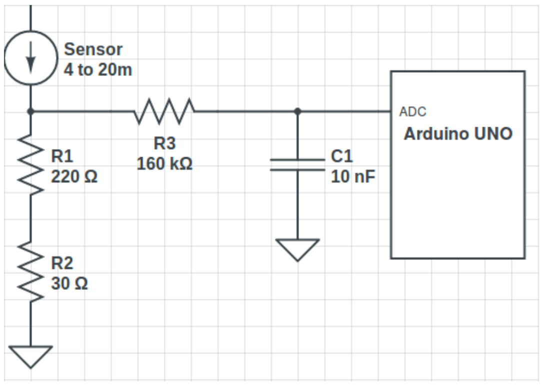
Figure 3.15: Connections between SICK sensor and Arduino UNO.

- **Readings's Resolution**:

  In sensors that deliver an analog output, their resolution is determined primarily by the noise on the signal lines and the input resolution of the device to which the sensor is connected. The noise is the main limiting factor in most measurement systems. Even if the resolution of the sensor is theoretically infinite, output changes which are smaller than the noise level will be "lost" in the noise. To solve this problem a low-pass filter was used to remove the noise [39]. This filter only allows low frequency signals from 0Hz to its cut-off frequency (fc) point to pass while blocking those any higher. The cuf-off frequency is calculated with the following equation:

$$f_c = \frac{1}{2\pi RC} \tag{3.2}$$

  The first circuit used to convert the output signal and filter the noise had a 100Ω resistor and a $1\mu F$ capacitor, which represents a filter with a cut-off frequency os $1592Hz$. All the components required to assemble the filter were welded together and the readings of the sensor were analyzed. It was possible to observe that the readings continue to oscillate excessively. Therefore, a new circuit had to be welded with a lower cut-off frequency. After redoing the calculations a filter with a fc of $99.5Hz$ was used to eliminate undesired higher frequencies. To do so, a $160k\Omega$ resistor and a 10 nF capacitor were used to weld the filter. The circuit implemented and the respective schematics are represented in figure 3.16.



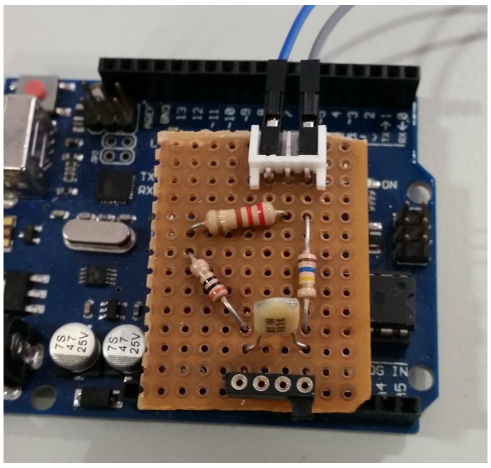


Figure 3.16: Low-pass filter before data acquisition.

Upon applying these changes, the readings improved but continued to oscillate slightly. This can be explained by the second parameter that determines the resolution of the sensor which is the microcontroller's input resolution. Analog inputs on microcontrollers must âĂIJdigitizeâĂİ an analog signal in order to utilize the information. This can be accomplished by using an Analog-to-Digital Converter (ADC). An ADC accepts the analog signal and assigns a discrete, digital value to a defined signal value. The Arduino UNO has a 10-bit ADC, which will return values between 0 and 1024, and since the sensor was limited to a minimum of 100mm and a maximum of 500mm each value will represent a variation of 0.4mm per bit. However, the sensor has a resolution output range between 0 and 4096 which represents 12bits, thus 2 bits are potentially lost when interpreted by the Arduino. This difference in resolution represents a variation of half a milliliter, so the average of the readings, which normally oscillate between two values, has to be calculated for higher accuracy.

- **Data acquisition and conversion**:

  After having everything successfully connected and operational, the data acquisition was carried out. Since the output signal of the sensor is not a distance in a scale of meters, it has to be converted by a linear regression. To determine this equation, it is necessary to gather some values from the sensor readings and the corresponding distance, and plot the best regression. Taking into account that a 250Ω resistor was used and that the readings range was set to a minimum of 100mm and a maximum of 500mm, using the incorporated menu display, the readings will vary between 0V and 5V. Forty-one readings were gathered using the analogRead() function. This values and the corresponding distance read on the sensor's display are listed in table B.1 in Appendix B.

After the collected data, the linear regression which is the calibration curve that links both readings, was calculated. The regression's plot is presented in figure 3.17. On the basis of this graph it is possible to demonstrate the linearity of this correlation, due to the accuracy of this type of displacement sensor. Consequently, the following equation was used to calculate the exact distance to the objects, through the sensor's readings.

$$Distance = -0.4969 \times Reading + 598.92 \tag{3.3}$$

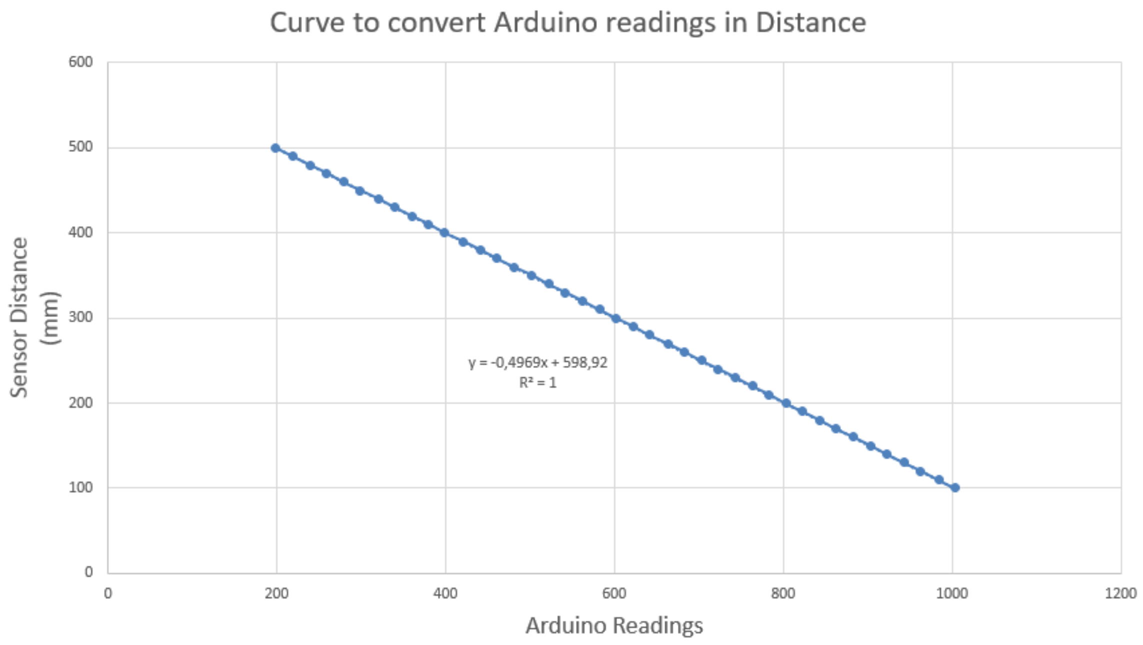


Figure 3.17: Graph that represents the calibration curve which links the sensor distance and the ADC reading.

To confirm the viability of the 3.1 equation some measurements were undertaken. For example when measuring 250mm with the sensor the Arduino readings changed between the values of 250.21mm and 250.78mm. These measurements cannot be successfully verified because the sensor integrated display shows distances limited to the millimetre level. This is vindicated by the loss of resolution when entering the Arduino, as previously explained. However, this limitation will not interfere with the ultimate goal of this project.

- **Code**:

  The main code written to read the analog output of the sensor and calculate the distance it represents is presented . This piece of code will also send the read distance through the serial port. The *DistanceSICK* variable is of type float in order to collect as many decimal digits as possible, for a more accurate analysis. Finally, the sketch was uploaded to the Arduino UNO, using the Arduino Software(IDE).

```
const int analogInPin = A0;  // Analog input pin

float sensorValue = 0;        // value read from the sensor
float DistanceSICK;         // output distance

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin (9600);
}

void loop() {

  //----------SICK----------
  // read the analog in value:
  sensorValue = analogRead(analogInPin);

  // filter with fc=100Hz (100mm - 500mm) R=250ohm
  DistanceSICK = -0.4969 * sensorValue + 598.92;

  Serial.print(DistanceSICK);
  Serial.print("\n");

}
```

Support for the installation in the Robot

Just like the Kinect, the laser sensor was also installed in the robot manipulator, therefore, a suitable support was modelled for this last device as well. Since the sensor is 72.4 x 54.1 x 24.3 mm in height, width and depth, respectively, it is possible to attach it to the end effector of the manipulator, thus allowing the sensor to have as many degrees of freedom as possible (6 DOF).

This support was designed with the exact dimensions to assure that the laser beam would have the exact angle as the gripper, this way, the end effector and the laser will both be pointing to the same object's surface with the same orientation.

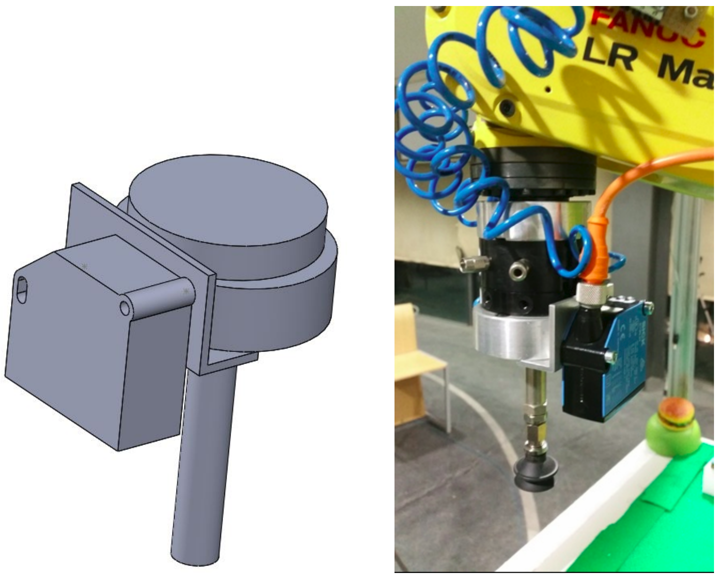


Figure 3.18: Support for the laser sensor. CAD model on the left.

3.1.4 Arduino UNO

The Arduino UNO, figure 3.19, is a widely used open-source microcontroller board based on the ATmega328P, which is a single-chip microcontroller. It contains 14 digital input/output pins, 6 of which can be used as PWM outputs, 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It can be powered simply by connecting it to a computer with a USB cable or by using a AC-to-DC adapter or battery [40]. This microcontroller is programmable with the Arduino IDE (explained in the software section) using a USB cable and was chosen to mark the release of Arduino Software (IDE) 1.0. This board is generally considered one of most user-friendly, popular and low-cost boards in the market [41].

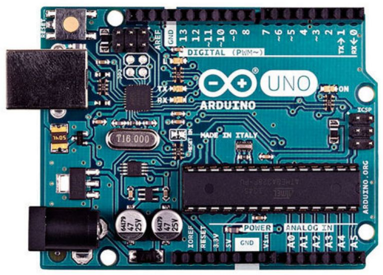


Figure 3.19: Arduino UNO board.

3.2 Software

3.2.1 ROS - Robot Operating System

Creating truly robust software for robotic systems is hard. To do so, there is a need to combine all the different equipments and the communications with each other, often with programs written in different languages. In order to simplify this type of software architecture the ROS framework was created.

The Robot Operating System (ROS) is a flexible framework for robot software development. It combines all the tools, libraries, and conventions with the aim of simplifying the task of creating complex and robust robot behavior across a wide variety of robotic platforms [42].

ROS contains different distributions which are a versioned set of ROS packages. The one used in this project is the ROS Kinetic Kame.

Figure 3.20: ROS Kinetic Kame logo.

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

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

Messages are structures of data filled with pieces of information by nodes. They are a combination of primitive types such as strings, Booleans, integers and floating point and messages which are a recursive structure.

Data is exchanged by two means, topics and services. A topic is a data transport system based on a subscribe/publish system which enables an asynchronous communication. One or more nodes are able to publish data to a topic and read data on that topic. A Service in contrast to a topic enables a synchronous communication between two nodes.

The overall process is represented in figure 3.21.

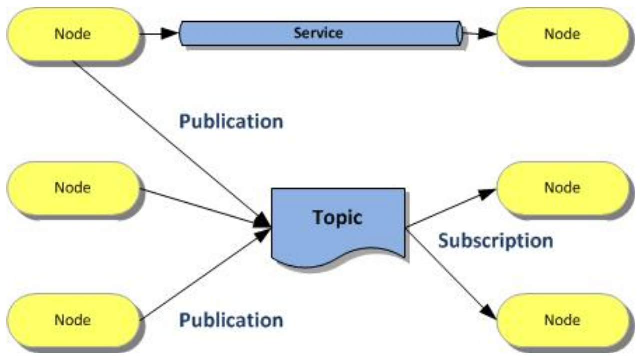


Figure 3.21: ROS basic concepts [43].

ROS also provides other facilities such as recording and playing back messages. Bags are formats for storing and playing back those messages.

In order to manage this loosely-coupled environment, there is a Master in ROS which is responsible for name registration and lookup for the rest of the system. The Master is a node declaration and registration service, which enables nodes to find each other and exchange data [43].

RViz

Rviz stands for ROS visualization and, as the name says, it is a 3D visualization tool for displaying sensor data and stating information from ROS. Using rviz, it is possible to visualize a manipulator's current configuration through it's URDF-described model. The Unified Robot Description Format (URDF) is an XML format for representing robot models, sensors, scenes, etc [44]. It is also possible to display a live representation of three-dimensional point clouds and sensor values coming over ROS Topics, such as sonar data, camera data and infrared distance measurement [45]. This tool is capable of using the information from the tf library to show all of the sensor data in a desired common coordinate frame, together with a 3D rendering of the robot. Visualizing all of this data in the same application allows the user to quickly see what the robot sees, and identify problems such as sensor misalignments or robot model inaccuracies [46].

Rqt

Rqt is a software framework for developing graphical interfaces for a robot by implementing various GUI tools in the form of plugins. With rqt it is possible to run all of the existing GUI tools in multiple widgets in a single window at one moment or just run it in a traditional standalone method. Users can also introduce new interface components by creating their own rqt plugins with either C++ or Python [46] [47]. However, rqt already has many useful plugins such as, *rqt_graph* for visualization of a live ROS system, showing nodes and the connections between them and *rqt_plot* to monitor encoders, voltages, or anything that can be represented as a number that varies over time.

`tf`

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

3.2.2 ROS Industrial

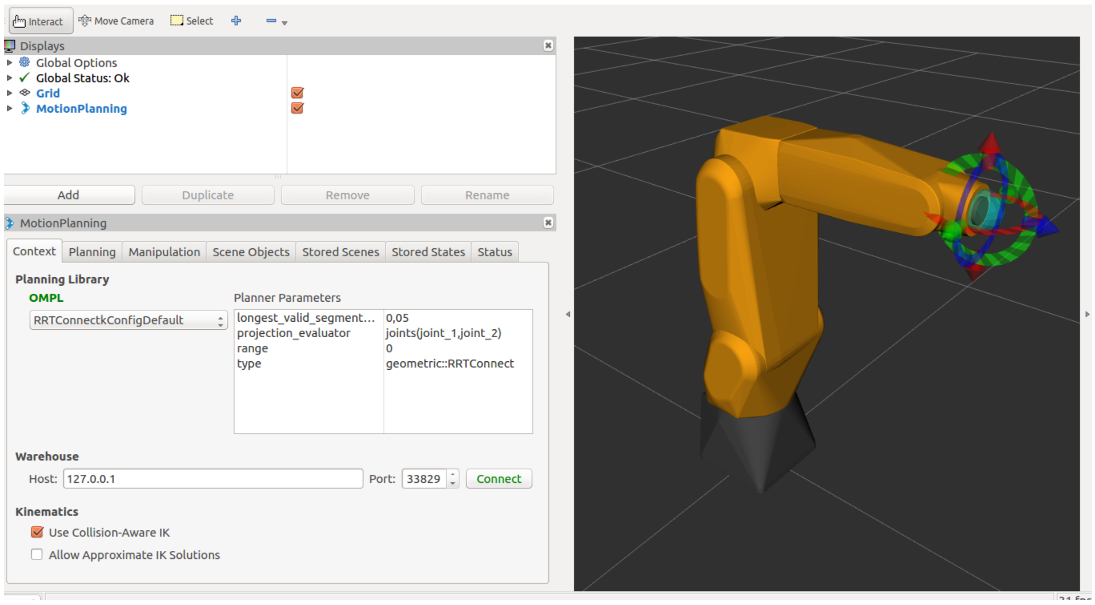
ROS-Industrial is an open-source project that extends the sophisticated capabilities of ROS software by bringing advanced robotics software to the industrial automation domain. This ROS extension contains libraries, tools, drivers and virtual models (URDF) for industrial hardware. [48] The instruction for the installation of ROS-Industrial can be found in [49]

The ROS-Industrial distribution contains metapackages for several industrial vendors, like ABB, Adept, Fanuc, Motoman, and Universal Robots. In order to manipulate FANUC LR Mate 200iD the ROS-Industrial relies, then, on the ROS *fanuc* metapackage. Additionally, the ROS MoveIt metapackage and the installation of the drivers on the controller are also required for the correct manipulation of the robot used in this project.

3.2.3 *MoveIt*

MoveIt! is an open source software which includes the tools and packages required for mobile manipulation. This package incorporates dynamic motion planning, manipulation, 3D perception, kinematics, collision checking, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications. [50]

The *MoveIt!* package can be used through a easy to use interface in RViz, figure 3.22 or through programming. However, by using this package through programming there will also be an interface that displays the motion plan computed, due to the associated topics published.



Planning Library
OMPL
RRTConnectkConfigDefault

Planner Parameters

longest_valid_segment...	0,05
projection_evaluator	joints(joint_1,joint_2)
range	0
type	geometric::RRTConnect

Warehouse
Host: 127.0.0.1 Port: 33829

Kinematics
☑ Use Collision-Aware IK
☐ Allow Approximate IK Solutions

Figure 3.22: RViz interface of the MoveIt package.

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

3.2.4 ROS *fanuc*

ROS *fanuc* is the metapackage that supports the integration of FANUC manipulators in a ROS platform. It currently contains packages that provide nodes for communication with FANUC industrial robot controllers and URDF models for various FANUC manipulators. The packages used in this project were the *fanuc_ driver* and the 3 packages that focus in the LR Mate 200iD manipulator. As a matter of fact, the last 3 packages are still integrated in the *fanuc_ experimental* metapackage which contains experimental packages that will be moved to the *fanuc* metapackage once they've received sufficient testing and review.

The *fanuc_ driver* package contains all the drivers to install in any FANUC controller for its interfacing with the ROS-Industrial program. This package contains different files designed to be compiled, through a FANUC Roboguide simulator, in order to generate executable files which will be then send to a FANUC controller.

The 3 packages that focus in the LR Mate 200iD are a *support package*, a *moveit_ plugins package* and a *moveit_ config package*. These include all the configuration files with information about the hardware, algorithms for computation of the robot's kinematics, STL models of all the robot's components for the recreation of the URDF model, etc. Since the robot's URDF model used for this project will also include the frames and the STL models of both the Kinect and the laser sensor it is necessary to create a *moveit_ config package* for this specific model, thus providing collision-aware path planning for the used robot. This can be done by following the tutorial in [51]. This tutorial walks through the steps of filling the *MoveIt* Setup Assistant wizard, explains how to update various configuration/launch files to provide additional robot-specific data that allows *MoveIt* to communicate with the robot interface node and also explain the process for connecting with a real robot. The second step of this tutorial also explains how to create the planning and execution launch file which will be the one used to invoke the ROS Industrial tools, to launch the planning environment and all the communications to control the manipulator. This communication is triggered by nodes and topics which are responsible for keeping track of the robot state, communicate with the controller, establishing the workspace, etc.

3.2.5 PCL - Point Cloud Library

Figure 3.23: Point Cloud Library logo.

The Point Cloud Library or PCL is a large scale, open project for 2D/3D image and point cloud processing, started by Willow Garage in 2010. PCL is available for Windows, macOS, Linux and Android and since it is included in ROS, it can be used in robotic projects and there is no need to install it.

A point cloud is a data structure used to represent a collection of data points defined by a given coordinates system and is commonly used to represent 3D data. With a 3D point cloud it is possible to define the shape of some real or created physical system. When color information is present the point cloud becomes 4D. Point clouds can be generated from a computer program synthetically or acquired from sensors such as 3D scanners and stereo cameras.

Willow Garage's purpose for creating this library was to accelerate 3D algorithmic work in perception for use in various robotic applications [52]. This library contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, object recognition, model fitting and segmentation. These algorithms can be used, for example, for perception in robotics to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract keypoints and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them. PCL is free for commercial and research use.

To simplify development, PCL is split into a number of modular libraries that can be compiled separately. These are: filters, features, keypoints, registration, kdtree, octree, segmentation, sample_consensus, surface, recognition, io and visualization. This way, the PCL distribution on platforms with reduced computational power or size constrains becomes easier [53].

3.2.6 OpenNI (Open Natural Interaction)

The OpenNI framework is an open source SDK, which is a set of software development tools that allows the creation of applications for a certain software framework, in this case ROS. OpenNI was primarily developed by PrimeSence, the company behind Kinect's depth sensor technology. It is used for the development of 3D sensing middleware libraries and applications [54]. OpenNI allows the user to access color and depth images from the camera and use its hands or body to interact with different digital devices. It was designed to work with a broader range of sensors in different operating systems.

In order to use the Kinect, it was necessary to install OpenNI (openni_camera) and Kinect driver by following the steps in [55]. Afterwards it is possible to run Kinect on ROS and visualise its data in Rviz.

When launching, the *openni.launch* ensures that the *device_ id* corresponds to the correct device name used.

3.2.7 Arduino IDE

The Arduino Integrated Development Environment (IDE) is a cross-platform application that is written in the programming language Java. This IDE is based on the open-source softwares Processing and Wiring. It runs on Windows, macOS X, and Linux and can be used with any Arduino board and other microcontrollers boards, such as NodeMcu.

This open-source software provides a simple one-click mechanisms to compile and upload programs to an Arduino board making this a simple process. It also contains a message area, a text console and a serial monitor and plotter [56].

Chapter 4

Calibrations

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

Until now the hardware of this project was all used separately. In order to integrate the robotic manipulator, the sensors and their controllers into a global system, it is important to determine the transformations between all the components. The robot, the Kinect, the laser sensor and the gripper all contain 3D coordinate frames that change over time. To keep track of all these frames the `tf` package, from ROS, was used. `tf` maintains the relationship between coordinate frames in a tree structure buffered in time and lets the user transform points and vectors between any two coordinate frames at any desired point in time [57].

4.1 FANUC Robot and End Effector

In order to include the coordinate frames of all the robot's joints, the URDF specification of the FANUC LR Mate 200iD was included to launch the global system. The specifications of the FANUC manipulators can be downloaded from the git repository in [58].

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

In order to incorporate the frame of the tool tip (`eef_tool_tip`), the gripper's exact length had to be determined. This was done by moving the end effector closer to the table and by visualizing the coordinates of the frames sent from the robot through a TCP/IP connection. By doing so, it was possible to determine the difference on the OZ axis between the base of the robot (`robot_base_link`) and the last frame of the robot (*eef_ base_ link*). This difference was later used to do the transformation to represent the frame of the tool tip (`eef_tool_tip`). This process is represented in figures 4.1 and 4.2 and in figure 4.2 it is possible to see that after the transformation, the `eef_tool_tip` is almost equal to zero on the Z direction, as desired.

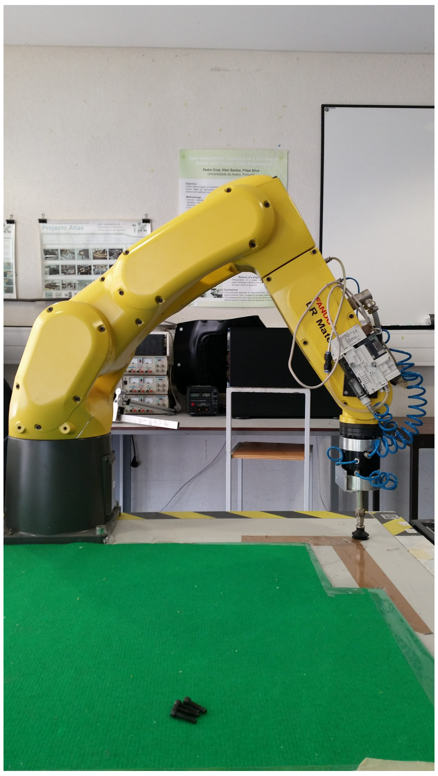
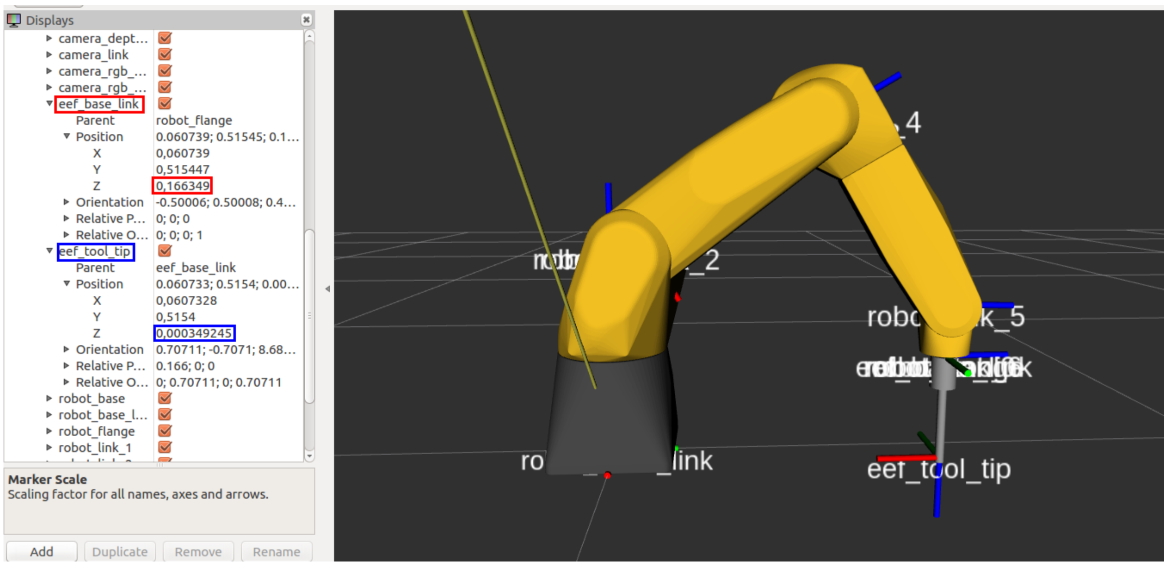


Figure 4.1: Robot's position for the end effector calibration.



```
Displays
▸ camera_dept...      ✓
▸ camera_link         ✓
▸ camera_rgb_...      ✓
▸ camera_rgb_...      ✓
▾ eef_base_link       ✓
    Parent            robot_flange
  ▾ Position          0.060739; 0.51545; 0.1...
      X               0,060739
      Y               0,515447
      Z               0,166349
  ▸ Orientation       -0.50006; 0.50008; 0.4...
  ▸ Relative P...     0; 0; 0
  ▸ Relative O...     0; 0; 0; 1
▾ eef_tool_tip        ✓
    Parent            eef_base_link
  ▾ Position          0.060733; 0.5154; 0.00...
      X               0,0607328
      Y               0,5154
      Z               0,000349245
  ▸ Orientation       0.70711; -0.7071; 8.68...
  ▸ Relative P...     0.166; 0; 0
  ▸ Relative O...     0; 0.70711; 0; 0.70711
▸ robot_base          ✓
▸ robot_base_l...     ✓
▸ robot_flange        ✓
▸ robot_link_1        ✓

Marker Scale
Scaling factor for all names, axes and arrows.
```

Figure 4.2: Coordinates of the frames in the end effector calibration.

The frame of tip of the gripper was represented with an offset of 165.5mm. This information made it possible to design a simple gripper to represent the real one, by incorporating the mesh of the gripper and the description of the last coordinate system (`eef_tool_tip`) in the URDF specifications.

The visualization of the robot with the end effector, in the respective position, and all coordinate frames, so far, is now possible using the following command:

```
$ roslaunch bin_picking globalvisualization.launch
```

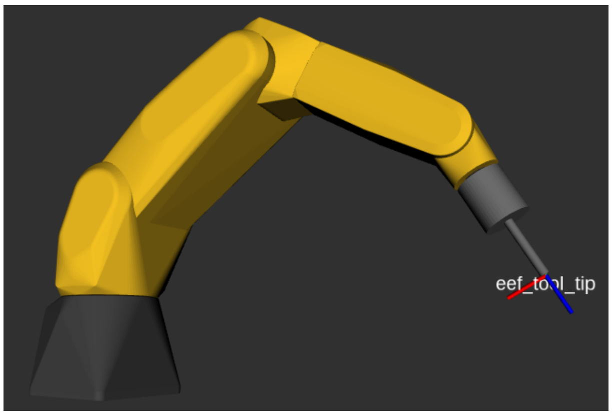


Figure 4.3: Visualization of the robot model with the end effector and is coordinate frame.

4.2 Laser Sensor

In order to correctly calibrate the laser sensor and incorporate its coordinate frame in the global system, the transformation from the tip of the gripper (`eef_tool_tip`) to the emitter of the laser had to be determined. Since the sensor is going to be used to create a bin-picking process with great accuracy, the calibration has to be precise and 4 variables have to be correctly determined to accomplish this task. These 4 variables are the difference in the X, Y and Z direction between the `eef_tool_tip` and the frame which overlaps with the emitter of the laser, as well as the rotation between the laser beam and the gripper. The last one is fundamental to confirm that the laser beam is parallel to the end effector, thus ensuring that both have the same orientation when pointing to an object's centroid.

The last variable was determined by placing the robot's end effector aligned exactly 90 degrees with the table. In order to ensure that the 5th joint of the robot was exactly 90 degrees withthe table, the robCOMM language was used to send a command to the robCOMM server to move the end effector to a desired cartesian position and pose specification. To do so, a library developed by a former student, João Peixoto, was used [59]. This library allows the user to communicate with the FANUC robot using a TCP/IP connection by sending and receiving commands in the robCOMM language. To put this to work a ROS node called `fanuc_control` was created to move the robot to a desired joint or Cartesian position specified in the program `fanuc_control.cpp`.

Having the robot's 5th joint exactly 90.000 degrees, the robot was moved from one position to another, moving only in the Z direction. The movement started in the position represented in figure 4.4 where the tool tip is 10.527 mm from the base of the robot. The center of the red dot, reflected by the laser, was then marked. The 5th joint state (90.000 degrees) and the Z coordinate of the end effector (10.527 mm) where both acquired through a robCOMM command that returns the manipulator's current cartesian position with a precision of three decimal digits.

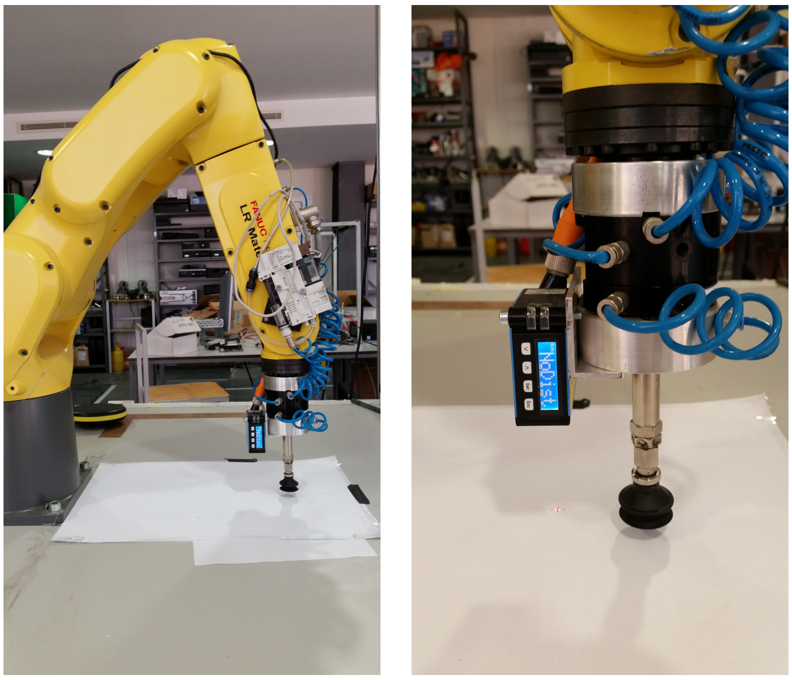


Figure 4.4: First position for testing the angle of the laser beam.

The movement ended in the following position, which is 528.93 mm from the global frame (robot_base_link). This coordinate was also acquired with the robCOMM command that returns the robot's current cartesian position.

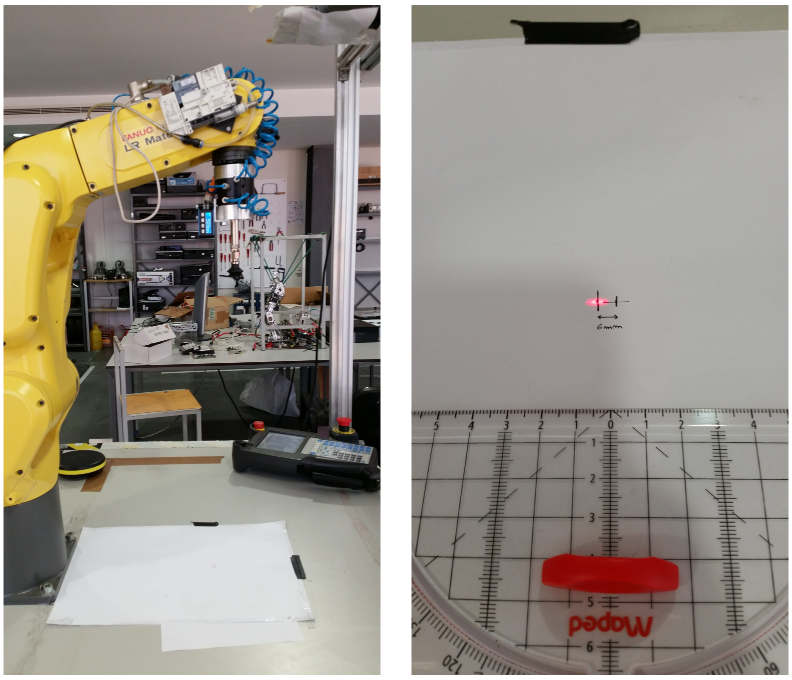


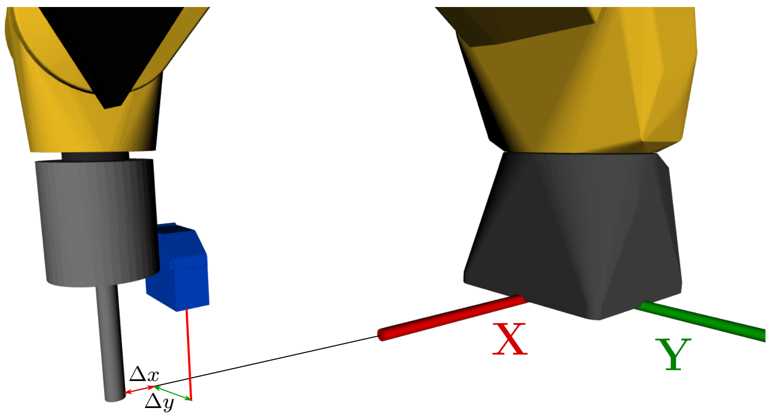
Figure 4.5: Second position for testing the angle of the laser beam.

Therefore, when moving the end effector 518.403 mm the laser moved 6.0±0.5 mm (measured with a ruler) and this represents that the laser shifted 0.66 degrees. This offset is less than 1 degree so it was not taken into account. If this offset had a relevant value there would be a rotation in the transformation between the frames of the tip of the gripper and the laser's emitter.

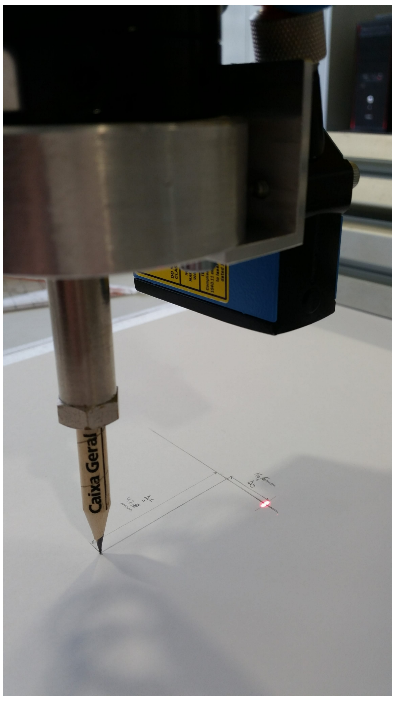
Afterwards the translation in the X,Y and Z directions were calculated. To calculate the exact difference from the tip of the gripper to the laser on the OZ axis some readings from the laser were gathered through the serial port, when it was vertically pointing to the table, thus knowing the difference to the base of the robot. The end effector's z-coordinate, read in RViz, was then subtracted from the laser sensor reading. After gathering the differences for different distances the average was calculated, hence having a 63.6 mm offset from the sender of the laser to the tip of the gripper in the z direction.

In order to calculate the translation in the X and Y directions the gripper was replaced by a pencil and the difference to the red dot projected by the laser was evaluated (figure 4.6b). This process was done by taping a paper sheet in parallel to the base of the robot and by moving it so the pencil could mark the sheet. Having the sheet in this position it was possible to draw parallel lines from the robot's base, thus having the exact offset in the X and Y directions between the tip of the pencil to the laser beam.

In the figure 4.6 it is possible to see the schematics, in 4.6a and the real difference in the X and Y direction, in figure 4.6b, which is 47.8 mm and 16.6 mm , respectively.



(a) Schematics of the process.



(b) Real image of the process.

Figure 4.6: Calibration of the laser sensor in the X and Y direction.

Having found the components for the transformation, the URDF files that describe the robot were changed. In order to incorporate the coordinate frame of the emitter of the laser sensor the `binpicking_robot_macro.xacro` file was altered and the `binpicking_gripper_macro.xacro` file was created to represent the sensor. The CAD model of the device was also included in the `binpicking_gripper_macro.xacro` file for a better visualization.

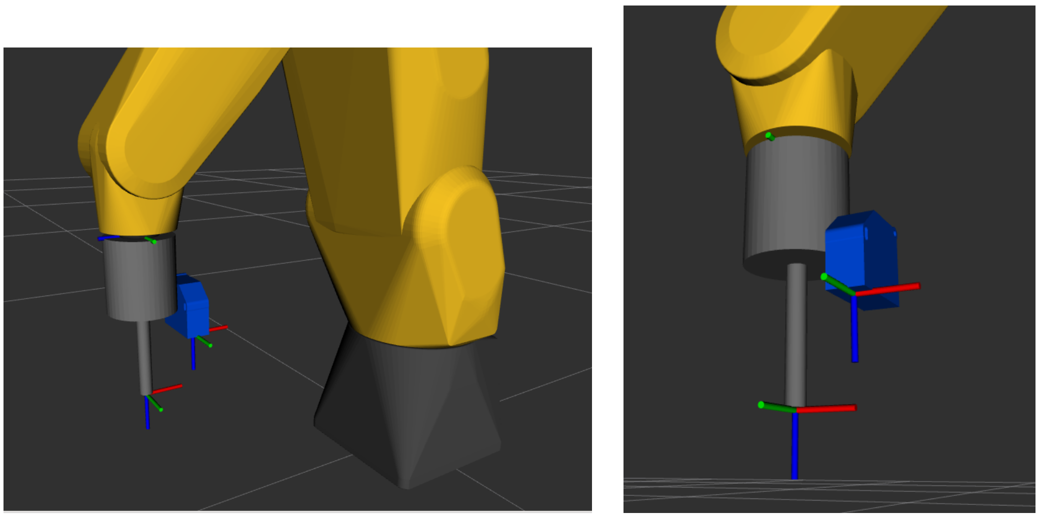


Figure 4.7: Incorporation of the frame of the laser sensor's sender and the CAD of the device.

4.3 Kinect

The calibration of a RGB-D sensor consists of two parts, the calibration of the intrinsic and extrinsic parameters. The intrinsic parameters represent a projective transformation from the 3D camera's coordinates into the 2D image coordinates. The extrinsic parameters represent a rigid transformation from 3D world coordinate system to the 3D cameraâĂŹs coordinate system. The camera calibration parameters are represented in figure 4.8 for a better understanding.

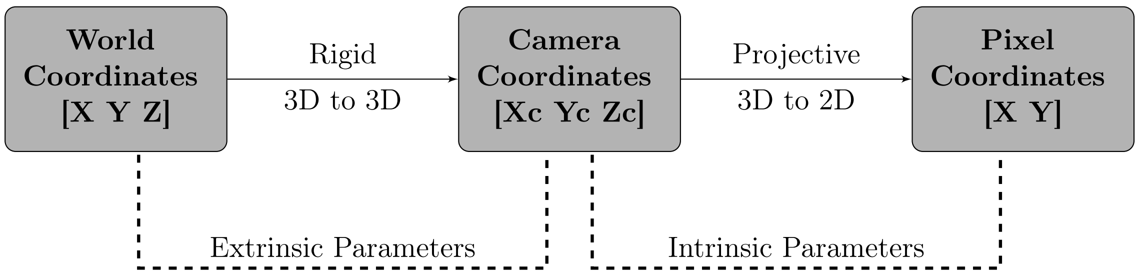


Figure 4.8: Camera calibration parameters.

4.3.1 Calibration of the Intrinsic Parameters

The intrinsic calibration is responsible for describing the internal parameters of both RGB and IR(depth) cameras. Those parameters are the focal length, the optical center, also known as the principal point, the skew coefficient and the coefficients that describe the lens distortion [60] [61].

Actually, this process is not truly required, since the `openni_camera` driver already provides default camera models with reasonably accurate focal lengths and other parameters definitions. Additionally, even though the driver does not model the lens distortion, the Kinect, fortunately, uses low-distortion lenses, so even the edges of the image will not be displaced by more than a few pixels. However, if the application in which the Kinect is used requires maximum accuracy, performing a rigorous calibrations is helpful [62]. Since the project requires as much accuracy as possible to create a precision bin-picking process, this calibration was performed.

The calibration procedure used for the purpose of calibrating both RGB and IR cameras was the one fully described in the tutorials [62] and [63]. In these tutorials the camera_calibration's `cameracalibrator.py` node is used to calibrate a monocular camera with a raw image. For this, a large checkerboard with known dimensions is required. The one used was a 8x6 checkerboard with 105.3 mm squares, which can be seen in figure 4.9.

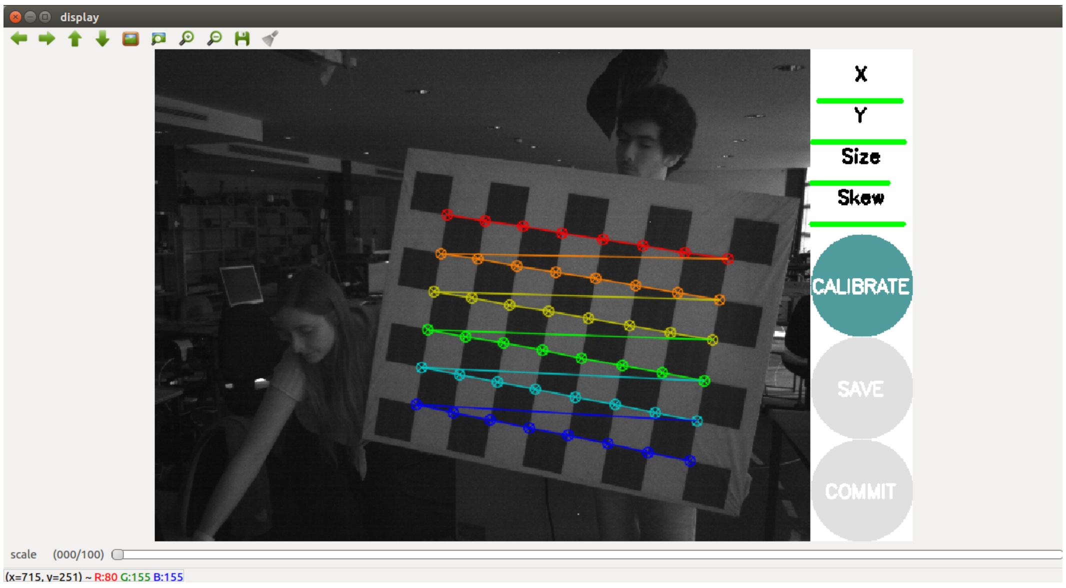


Figure 4.9: Calibration window with the highlighted checkerboard.

In the depth calibration, the speckle pattern emitted by the IR projector makes it impossible to accurately detect the checkerboard corners, because of the clarity the speckles create on the surfaces. The solution is to cover the IR light source, as in figure 4.10, thus blocking the speckles, and provide a separate IR light source, such as the sunlight or incandescent lamps [62].

After the intrinsic calibration, the automatically saved parameters, which are stored in the `rgb_deviceID.yaml` and the `depth_deviceID.yaml` files, were used in the launch file which configures the camera info URLs. The URL (Uniform Resource Locator) expresses the location for getting and saving calibration data [64].

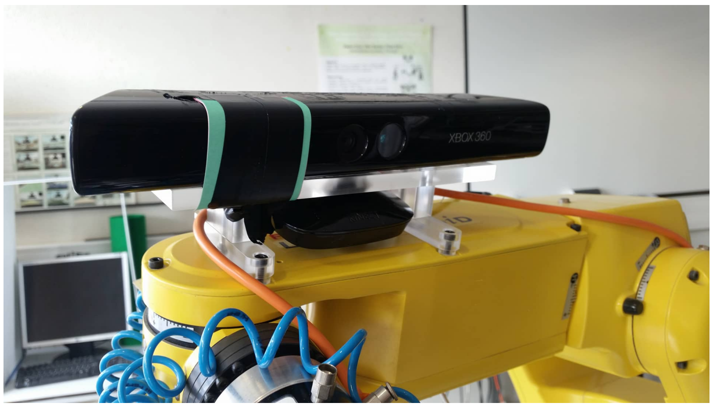
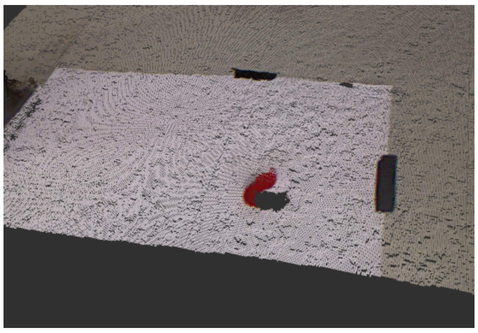
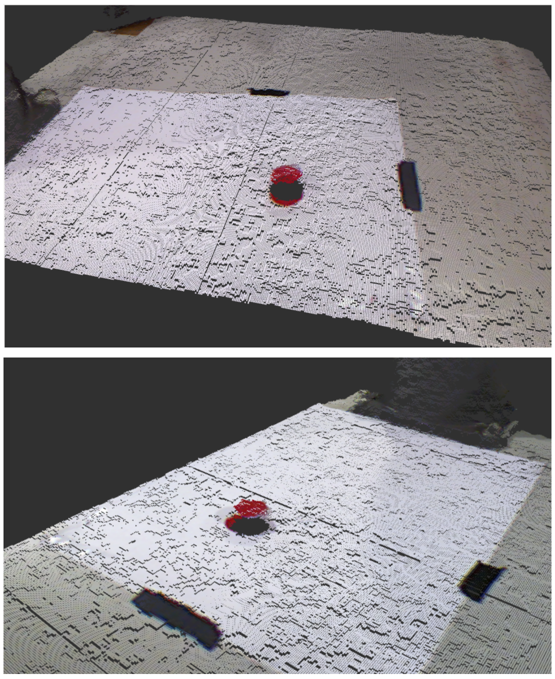


Figure 4.10: Kinect with the IR projector blocked for the intrinsic calibration of the IR (depth) camera.

To evaluate the accuracy of this calibration, it is possible to visualize the obtained point cloud combined with the RGB data (`depth_registered`) and evaluate if they are correctly aligned. Figure 4.11 presents the comparison between two obtained point clouds, one before, (4.11a), and other after, (4.11b), the calibration, to demonstrate the importance of this calibration. The point cloud and the RGB data before the calibration are noticeably more misaligned, since the points in red do not coincide with the points which represent the piece on the table that have a different hight.



(a) Before the calibration.



(b) After the calibration.

Figure 4.11: Point cloud, of an object on the table in front of the manipulator, before and after the Kinect's intrinsic calibration.

4.3.2 Calibration of the Extrinsic Parameters

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

In this case, the camera is mounted on the robot's 4th joint, therefore it is necessary to compute the static transformation from the frame of the robot's joint to the optical frame of the camera. This is referred to as an eye-in-hand type of calibration.

To do this calibration, the `visp_hand2eye_calibration` ROS package [65] was used. This package is used to estimate the camera position with respect to its effector (in this circumstance the robot's arm) using the ViSP library.

To compute the relative transformation between the camera and the hand/arm, it is necessary to feed the calibrator node with the `/world_effector` and the `/camera_object` transformations, which are known. The `/world_effector` represents, in this specific case, the dynamic transformation from the global frame (`robot_base_link`) to the frame of the robot's 4th joint (`robot_link_4`). The `/camera_object` transformation is calculated using an ArUco marker and the `aruco_detect` package [66] and represents the dynamic transformation between the camera and the ArUco frames. To use this last package, the installation of the fiducial software is necessary. This provides a system that allows a robot to determine its position and orientation by looking at a number of fiducial markers, detected by the `aruco_detect` node, that are fixed in the robot's environment [67]. The process of the ArUco detection can be visualized in figure 4.12.

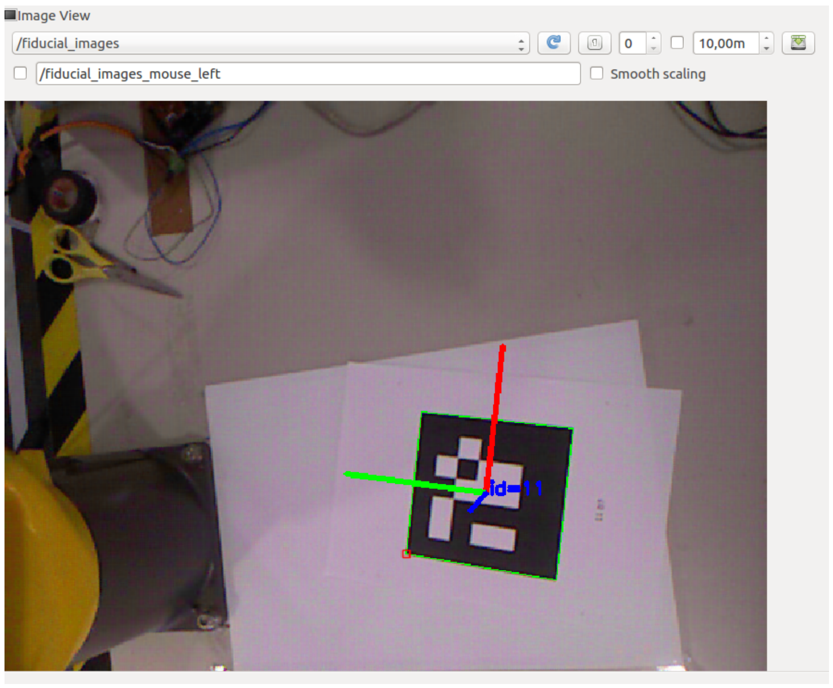


Figure 4.12: ArUco detection.

Figure 4.13 presents a simple schematics of the calibration process. This calibration computes the desired static transformation (`/effector_camera`) through the two known dynamic transformations published in the topics `/world_effector` and `/camera_object` acquired in different poses.

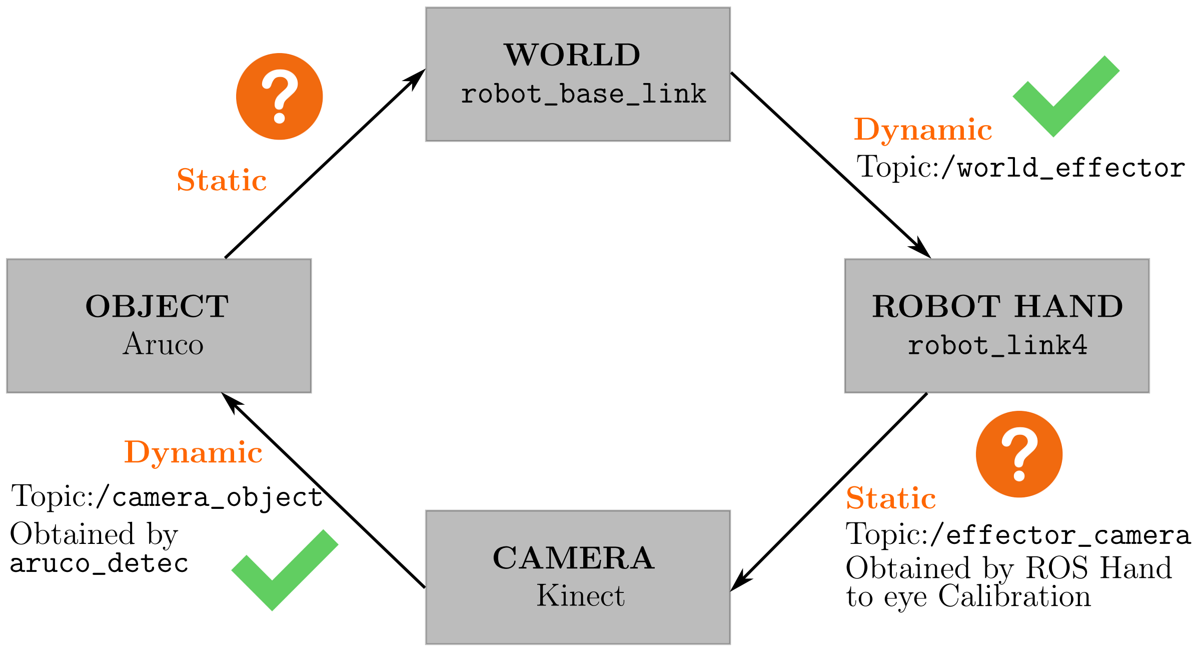


Figure 4.13: The Kinect's extrinsic calibration process.

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

Lastly, the `/effector_camera` transformation and Kinect xacro were included in the `binpicking_macro.xacro` file to create the relation between the robot's 4th joint and the `camera_link` frame, which represents the Kinect.

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

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

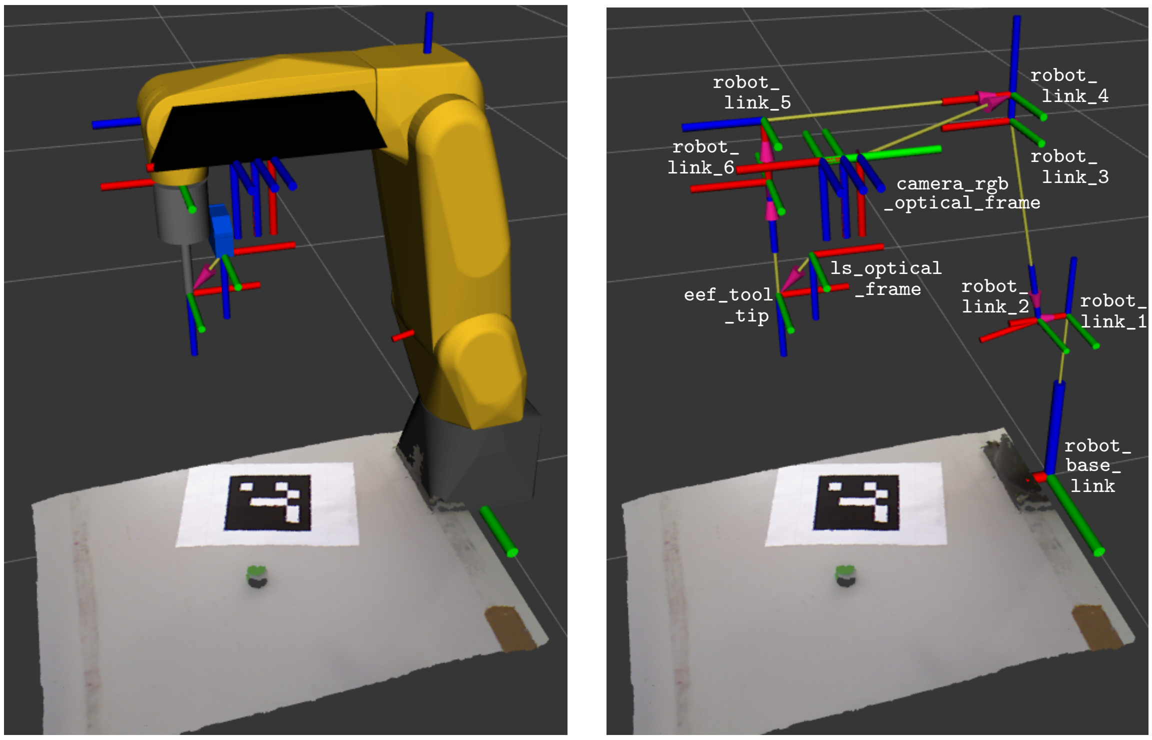


Figure 4.14: Complete system with all the components properly calibrated and all the corresponding frames.

It is also possible to visualize the tf tree with the frame coordinates of the entire system in figure 4.15. The frames that belong to the Kinect, the end effector and the laser sensor are properly highlighted in the figure. The `tfs` with the prefix `robot` are all launched through the FANUC driver.

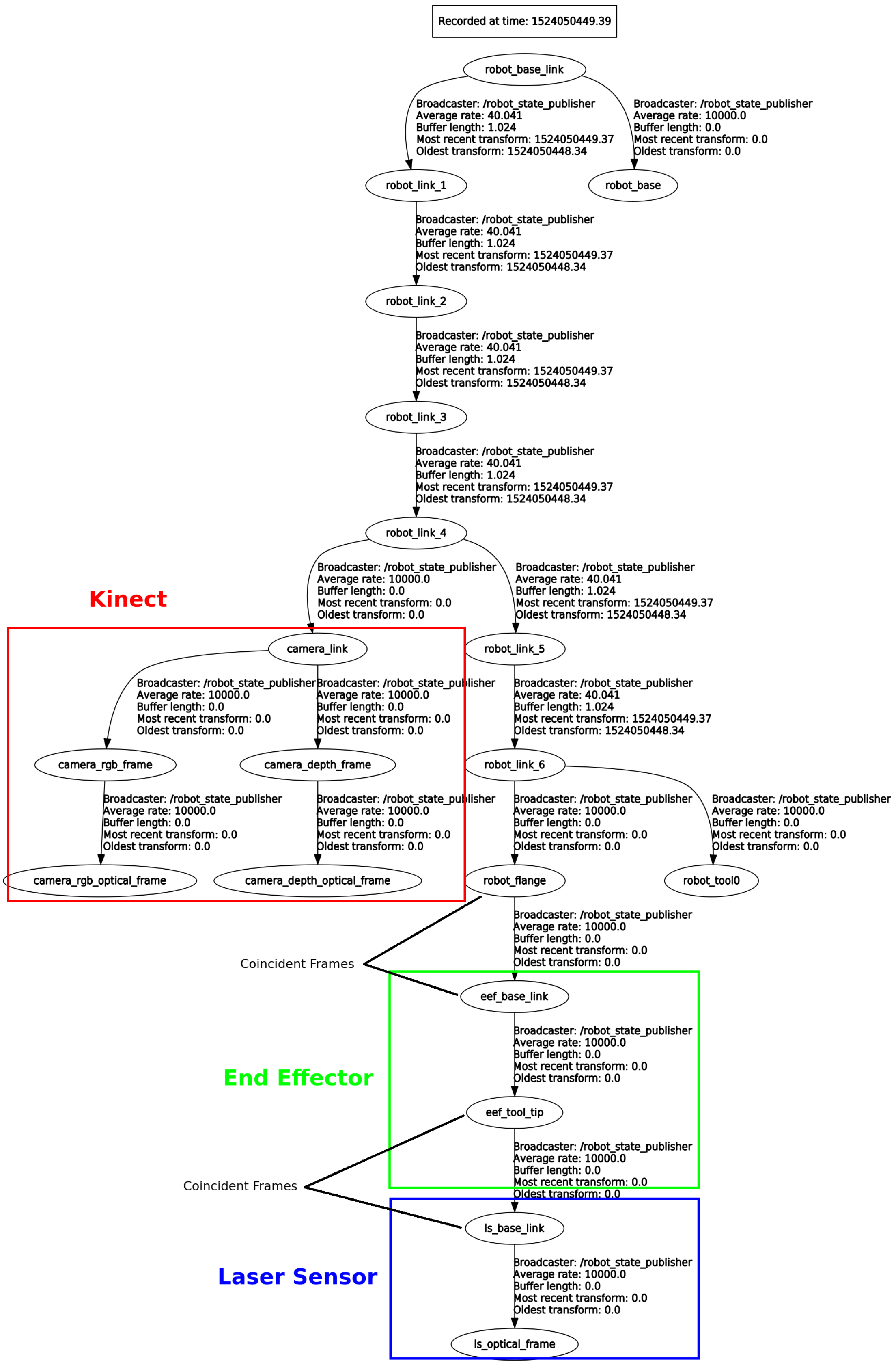

Recorded at time: 1524050449.39
robot_base_link
Broadcaster: /robot_state_publisher
Average rate: 40.041
Buffer length: 1.024
Most recent transform: 1524050449.37
Oldest transform: 1524050448.34
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
robot_link_1
robot_base
Broadcaster: /robot_state_publisher
Average rate: 40.041
Buffer length: 1.024
Most recent transform: 1524050449.37
Oldest transform: 1524050448.34
robot_link_2
Broadcaster: /robot_state_publisher
Average rate: 40.041
Buffer length: 1.024
Most recent transform: 1524050449.37
Oldest transform: 1524050448.34
robot_link_3
Broadcaster: /robot_state_publisher
Average rate: 40.041
Buffer length: 1.024
Most recent transform: 1524050449.37
Oldest transform: 1524050448.34
robot_link_4
Kinect
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 40.041
Buffer length: 1.024
Most recent transform: 1524050449.37
Oldest transform: 1524050448.34
camera_link
robot_link_5
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 40.041
Buffer length: 1.024
Most recent transform: 1524050449.37
Oldest transform: 1524050448.34
camera_rgb_frame
camera_depth_frame
robot_link_6
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_rgb_optical_frame
camera_depth_optical_frame
robot_flange
robot_tool0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Coincident Frames
eef_base_link
End Effector
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
eef_tool_tip
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Coincident Frames
ls_base_link
Laser Sensor
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
ls_optical_frame

Figure 4.15: tf tree of all the coordinate frames.

Chapter 5

Segmentation of the Point Clouds

Having all the hardware correctly calibrated, and being already capable of acquiring Kinect data, the next important step is processing the "registered" point cloud. "Registration", in this case, is the process of aligning the depth with the RGB optical frame; this way, every pixel of the acquired point cloud contains depth and color information. Therefore, this chapter presents a detailed explanation of the Kinect data processing.

In the point cloud generated by the Kinect, the objects are normally represented by surfaces of points as shown in figure 5.2b. In order to perform a precise bin-picking process, the determination of the centroid and the normal of those surfaces is crucial. The centroid represents the destination point for the tip of the robot's gripper, in order to grasp the desired object. The normal represents the gripper's orientation when approaching the object so that the suction cup can be parallel to the surface in order to enable a correct grip. The suction cup of a suction gripper is highlighted in figure 5.1.

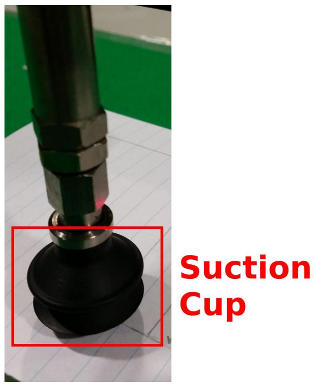


Figure 5.1: Suction cup.

The steps of the process used to acquire the centroid and normal information of each object are listed below:

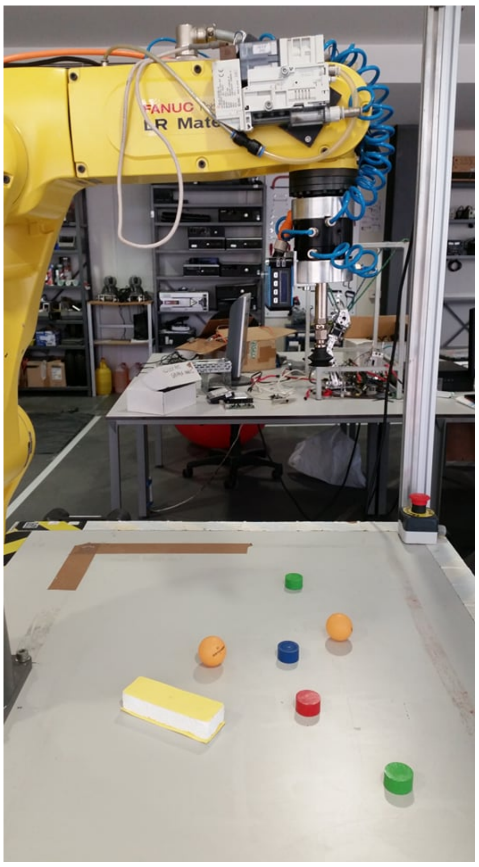
1. Removal of undesired points from the point cloud, such as, points on the floor or points that represent the robot's base;
2. Identification and removal of points from the table where the objects lay;
3. Clustering of the point cloud to separate the different objects;

4. Determination of the centroid and normal for each cluster.

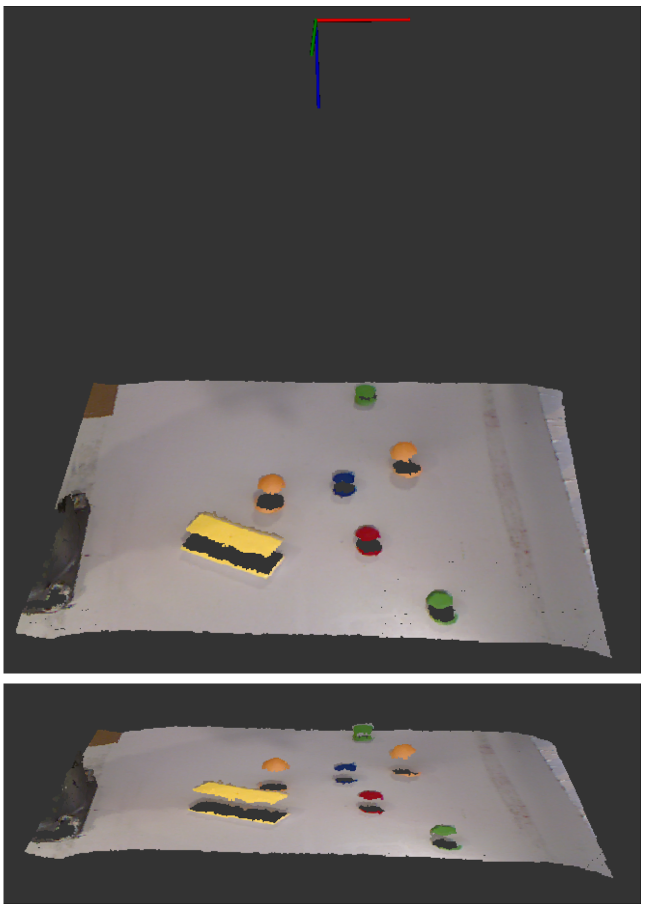
The point cloud used to develop this process was recorded into a *bag* through the rosbag package, so that the code could be written and tested without being always connected to the Kinect sensor. A bag is a file format in ROS for storing ROS message data [68]. The `pieces_segmentation2.bag` file stored in the bagfiles folder was the one used for this purpose.

The `objDetection.cpp` program, contained in the source file of the project, was created to subscribe to the `/camera/depth_registered/points` (XYZRGB) topic published by the Kinect, invoking a callback function whenever a new message arrives to this topic. This function is the one responsible for processing the data, publishing and presenting the results.

The initial point cloud before the segmentation process is shown in figure 5.2b acquired from the scene shown in 5.2a in which the objects to catch are 4 checker pieces (2 green, 1 blue and 1 red), 2 orange ping pong balls, and 1 yellow parallelepiped.



(a) The real scene from which the point cloud was taken.



(b) To views of the initial point cloud.

Figure 5.2: The point cloud obtained from the scene on the left, before the segmentation process.

5.1 Extraction of outliers and filtering

In order to correctly identify the objects, the first step is to eliminate, from the initial point cloud, points not belonging to the table in which the objects are on. Those are points that represent the base of the robot and/or points beyond the table, such as the floor, thus leaving only the objects and the background table.

The filter used to do this was the **PassThrough** which filters along a specified dimension, thus removing the undesired points. This filter is capable of identifying points within a specific range of X, Y and Z values and cut off values that are either inside or outside the given range.

The points which were far away from the camera, i.e., a more than 0.9 meters, along the Z direction were removed, thus removing the points beyond the table. Since the process of acquiring the point cloud of the workspace will always be done with the manipulator in the same respective position, the range in the X direction necessary to remove the points from the robot's base will always be the same.

The results after applying this filter can be seen in figure 5.3.

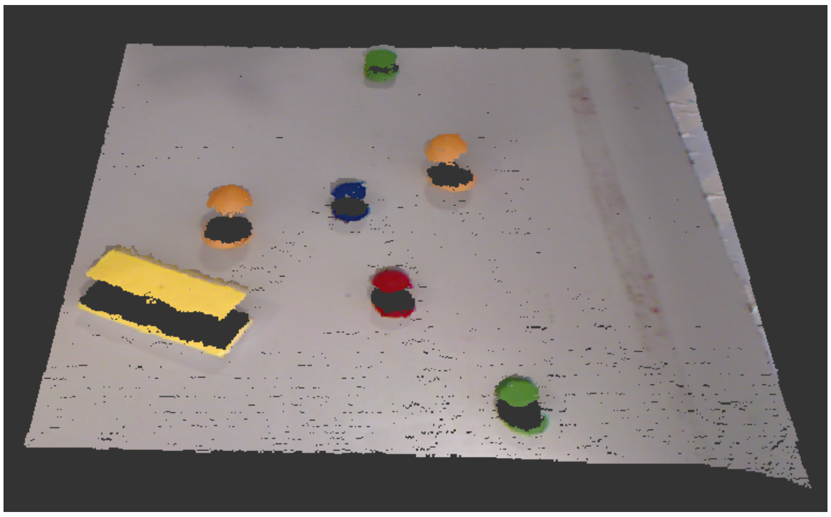


Figure 5.3: Point cloud after the PassThrough filter.

The point cloud acquired by the Kinect has a resolution image stream of 640 × 480 at 30 FPS and since the subscribed topic (`camera/depth_registered/points`) has the RGB and depth information combined, each pixel will have 32 bytes of data. Those 32 bytes consist of 16 bytes for depth information, in which the last 4 bytes are for padding, and 16 bytes which are mostly padding for RGB [69]. Consequently, this means that a point cloud from this topic will have $640 \times 480 \times 32$ bytes of data which is 9.8 MB. By using such a heavy point cloud, the processing process becomes slower; and sometimes even impossible. To save both bandwidth and disk space, and for flexibility in further processing, the number of points of the point cloud has to be reduced.

The filter used to reduce the number of points is called **VoxelGrid**. This filter is used to simplify the cloud, by wrapping the point cloud with a three-dimensional grid and reducing the number of points to the center points within each block of the grid. A function was created to apply this type of filtering called *voxelgrid*. This function has an input parameter which defines the size of the grid used in the *LeafSize* parameter of the VoxelGrid class. A 2 mm $\times$ 2 mm $\times$ 2 mm cubic grid was the one chosen for this filtering, thus maintaining the surface's details of the objects while reducing the exaggerated amount of points. The point cloud resulting from this type of filter is presented in figure 5.4.

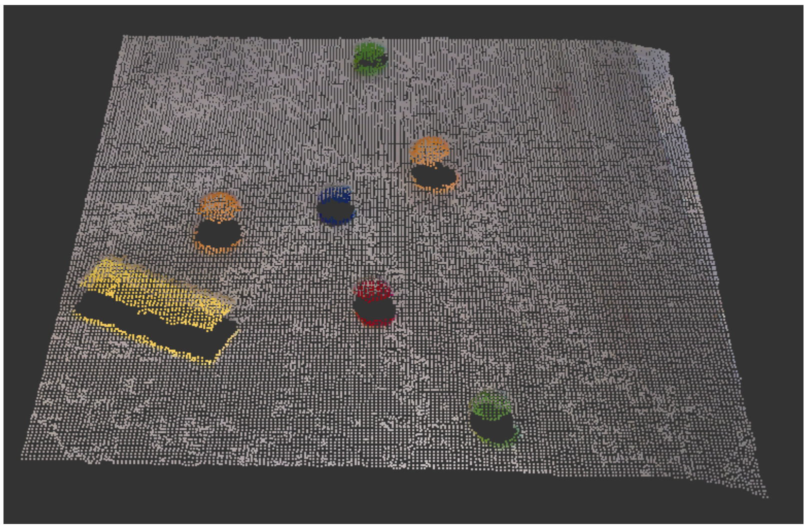


Figure 5.4: Point cloud after the VoxelGrid filter.

The very high density of the point cloud in figure 5.3 is visible in comparison to the one represented in 5.4. From now on, the point cloud requires less computational power, making the following steps easier to process.

5.2 Extraction of the background

The next step consists of identifying and removing the background formed by the table, thus leaving only the part of the point cloud that represents the objects to grab. In order to identify the table, the **SACSegmentation** class does a simple segmentation of a set of points with the purpose of finding the dominant plane in a scene.

In order to do this, the SACSegmentation object was created and the model and the method type were defined. The method used was the *RANdom SAmple Consensus (RANSAC)*. This is an iterative method used to estimate and detect *outliers* from a set of observed data in a robust but simple way [70]. In addition, the "distance threshold", which determines how close a point must be to the model in order to be considered an *inlier* [71], has to be specified. Since the shortest pieces are 16.8 mm in height, the *DistanceThreshold* parameter was set to 15.5 mm in order to guarantee that the points that represent the objects will not be removed.

After the *inliers* that represent the table were identified, they were removed using the **ExtractIndices** filter, thus leaving only the *outliers*. This filter is used to extract a subset of points from a point cloud based on the indices outputted by the segmentation algorithm used in [72].

The point cloud after the extraction of the background can be seen in figure 5.5.

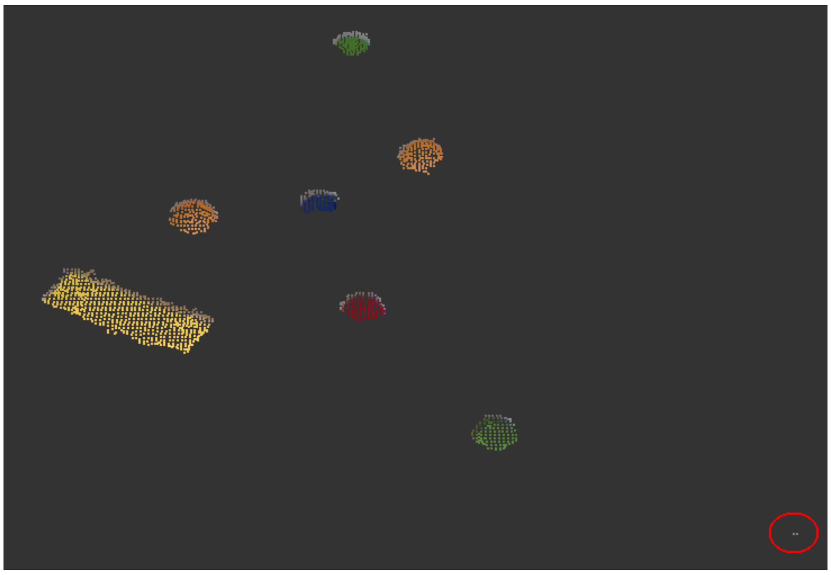


Figure 5.5: Point cloud after the extraction of the background.

In figure 5.5, it is possible to visualize that not all points from the table were correctly removed. Those points are highlighted in the same figure. In order to remove these isolated points the **RadiusOutlierRemoval** filter was used. This is capable of removing all indices in its input cloud that do not have at least some number of neighbors within a certain range [73]. Therefore, points that do not have at least 45 neighborhood points within a radius of 20 mm will be eliminated. The point cloud that results from this filter is presented in figure 5.6 and it is possible to see that the only points now in the resulting point cloud belong to a specific object.

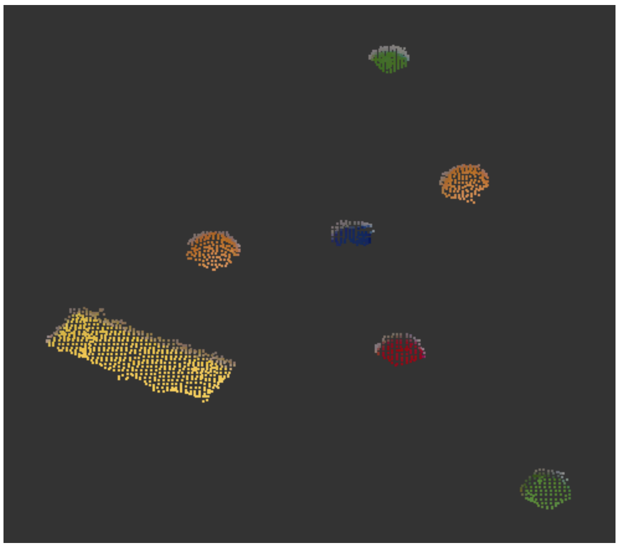


Figure 5.6: Point cloud after the extraction of the background and isolated points.

5.3 Clustering

Having only the points that represent the objects to grab, the following step is to split them in different point clouds, each one representing a specific object. The clustering method used to separate and extract the objects was the **Euclidean Cluster Extraction** with the *pcl::EuclideanClusterExtraction* class. This process uses a plane segmentation algorithm to identify different surfaces, which is the same as the one used in the identification of the background, in the previous section. This clustering method is only used to separate different objects that are not in close contact with each other; in that case another class has to be used, for example the *pcl::RegionGrowing* [74].

Since this segmentation is for a precision bin-picking process, the objects used were small and without complex geometries, so the whole gripper's suction cup is able to lean against the object's surface for a correct grip. This way, these pieces will be represented in the point cloud only as one surface, in contrast to larger objects in which the upper surface and probably one or two surfaces from its side would also appear.

The euclidean clustering essentially groups points that are close together. For this to work, the following parameters must be correctly customized. Firstly the *ClusterTolerance* must be set to define the range in which the points will be considered to belong to the same cluster. This parameter has to be carefully selected to suit each dataset. If the tolerance is set to a very small value, it is possible for an actual object to be seen as multiple clusters. On the other hand, if it is set to a value that is too high, it is possible for multiple objects to be seen as one cluster. For this specific dataset, the *ClusterTolerance* was set to 35 mm because the checker pieces have a diameter of 32 mm.

Another feature to be defined is the minimum and the maximum number of points that a cluster can contain. Having already reduced the number of points and since the smallest objects have normally 200 ore more points, and the larger ones 1500, the *MinClusterSize* and the *MaxClusterSize* were set to 100 and 2500 respectively.

For the search method of the extraction, which is the last defined parameter, a *KdTree* object was created. The *search::KdTree* is a wrapper class which inherits the *pcl::KdTree* class for performing search functions [75], such as finding the K nearest neighbors of a specific point or location, using a KdTree structure. This type of structure is a data structure used in computer science for organizing a number of points in a space with K dimensions [76].

After the extraction of the clusters out from the initial point cloud, the indices for each were saved in *ece_indices*. To separate each cluster it is necessary to iterate through the *ece_indices*; new point clouds for each entry have to be created and all of the points of the current cluster have to be written in the corresponding point cloud [77].

Figure 5.7 shows the different clusters, each with a distinct color, in one viewer.

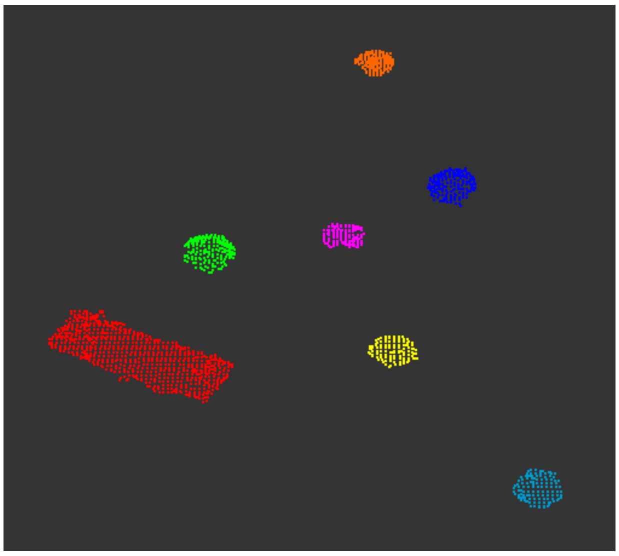


Figure 5.7: The objects already clustered into different point clouds with distinct colors.

5.4 Calculation of each Centroid and Normal

The final step of the segmentation is the determination of the centroid and its normal for each cluster obtained in the previous section. The normal is represented by a vector perpendicular to the plane tangent to the surface of the cluster originated from its centroid. To compute both the centroid and normal, the following process was used for each cluster separately and, simultaneously, the data was stored in two different vectors, which are containers that represent arrays capable of changing size.

1. Firstly, the normals of all of the points of a cluster are computed using the **NormalEstimation** class. This class retrieves the nearest neighbors for each point and fits a plane to them. Afterwards, the normal of the plane is taken as the local surface normal. This method can estimate normals with high detail, regardless of the density of the point cloud because it uses the nearest neighbors independently of their distances [78]. To use this class, some parameters have to be correctly customized. The *SearchMethod* used was the same as the Euclidean Cluster Extraction, so a *KdTree* object was once more created. The *RadiusSearch* parameter was set to use all neighbors inside a sphere of 50 mm diameter. This issue is of extreme importance since the scale has to be selected based on the level of detail required by the application. If the curvature of a surface is important the scale factor needs to be larger. Since only two of the used objects have a curved surface (the ping pong balls) the 35 mm diameter was the one preferred after some analysis with different values in order to find the perfect balance between them.

2. The second main step consists of computing the center of the surface, which will be referred to as the virtual centroid since it does not represent a point of the surface but rather the central point of all the surface's points. To do so, a fixed-size vector of 4 floats from the *Eigen* library was created and the *pcl::compute3DCentroid* class was used. This class estimates the X, Y and Z coordinates of the centroid of a set of points and returns it as a 3D vector.

3. Having the virtual centroid of the surface, the index of the cluster's point which is closer to this virtual centroid was then determined. To do so, algorithm 1 was used. This will loop through all of the index values and store, in the *index_ center* variable, the index of the point which has the shortest distance to the virtual centroid.

```
Input: Cluster's point cloud and its size and Virtual Centroid(ViCenter)
Output: Index of the center point (index_ center)
for Index = 1 : Size of the cluster's point cloud do
    Distance = | CloudPoint.XCoord[Index] - ViCenter.XCoord | + | CloudPoint.YCoord[Index] - ViCenter.YCoord | + | CloudPoint.ZCoord[Index] - ViCenter.ZCoord |;
    if $Distance \leq Distance\_Before$ then
        Distance_Before = Distance;
        index_ center = Index;
    end
end
```

Algorithm 1: Algorithm used to determine the index of the center point.

4. The index of the point which has the shortest distance to the virtual centroid can be now used to determine the centroid of the object's surface. This virtual centroid computed in 2 can not be used as the grasping point because, for curved surfaces like the ones obtained with the ping pong balls, this centroid will be below the interior surface instead of above the external surface. This is better visualized in figure 5.8 in which the white point represents the virtual centroid and the red one represents the centroid of the surface which will be later the grasping point after some adjustments with the laser sensor measurement.

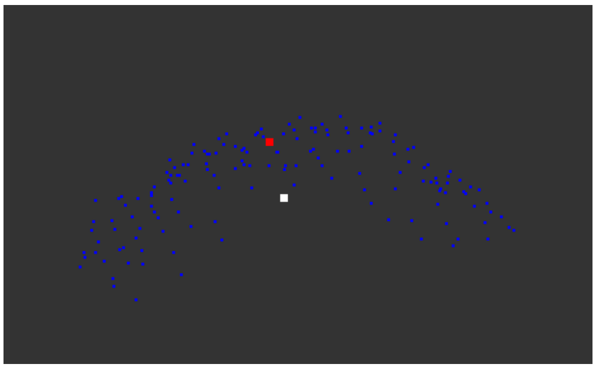


Figure 5.8: Representation of the virtual centroid (white) and the surface's centroid (red) in a curved surface.

5. Lastly, the normal of the surface's centroid was also determined from all of the normals computed in 1 through the index of the point with the shortest distance to the virtual centroid.

In figures 5.9 and 5.10 it is possible to visualize the objects with their corresponding centroids (surface centroids) and their normals.

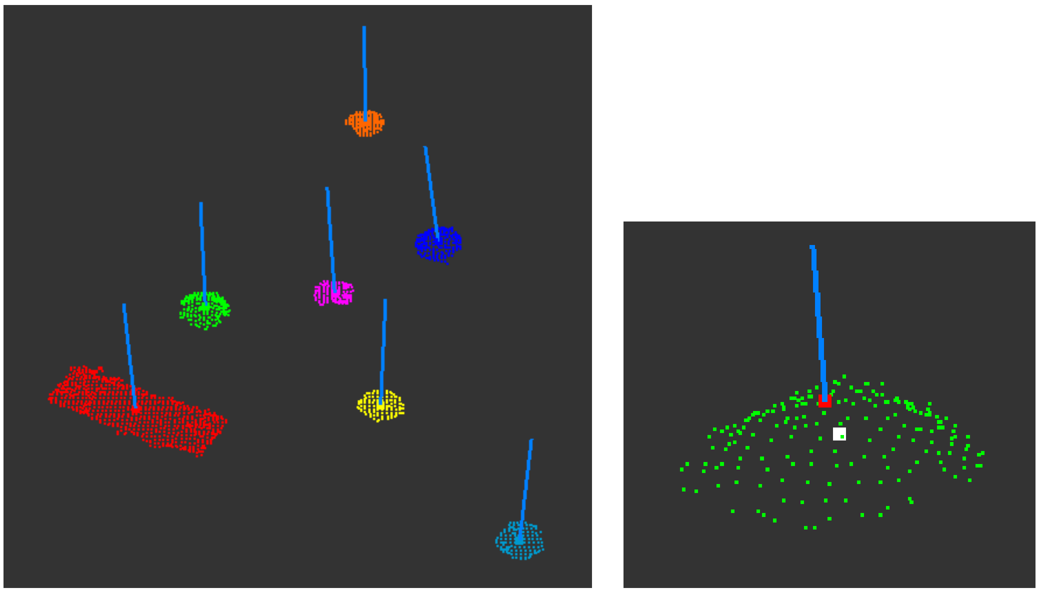


Figure 5.9: Representation of the clusters of the all objects with their corresponding centroids and normals.

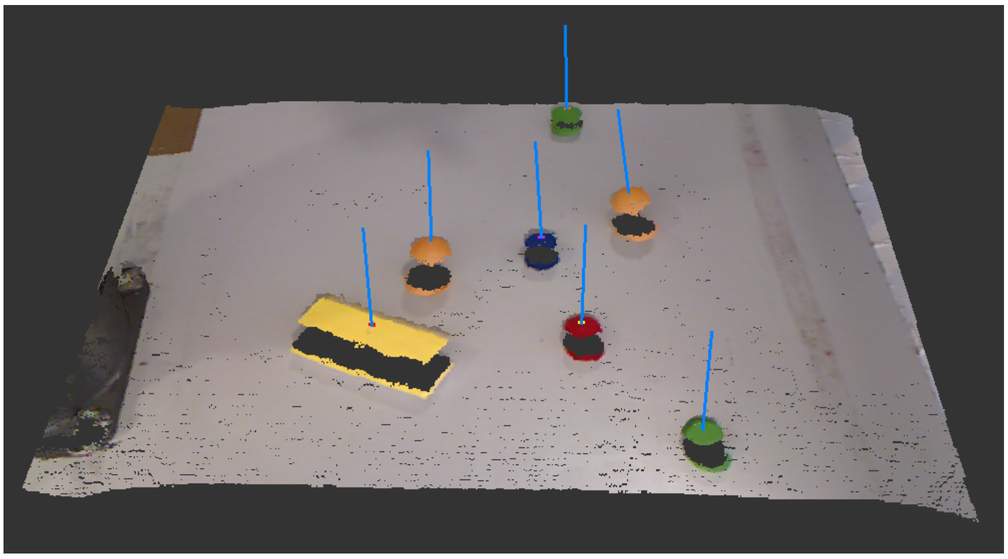


Figure 5.10: Representation of the object's centroid and its normals in the initial point cloud.

The centroid and its normal of the first clustered object were published in the topics `/cloud_centroid` and `/cloud_centroid_normal` respectively. These have their coordinates in relation to the `tf /camera_rgb_optical_frame`, which is one of the frames of the Kinect sensor, and will be later transformed to be in relation to the `/robot_base_link` frame of the manipulator, for future manipulation.

Chapter 6

System Integration

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

In order to elaborate the path carried out by the manipulator to correctly grasp an object, different waypoints have to be determined. To do so, information like the approximation point's coordinates, the surface's normal vector, the Euler angles, the end effector's coordinates for the measurement with the laser sensor and the grasping point's coordinates are required for the determination of those waypoints. The approach used for the acquisition of all this information is explained below from section 6.1 to 6.5.

Consequently, different nodes had to be created for the organization of the process. Section 6.6 describes the purpose of each node, explains the connection among them all and how they communicate with each other and through each topics.

6.1 Approximation Point

To guarantee the safety of the entire bin-picking process, it is important to determine an approximation point closer to the centroid of the object to grab. When moving the manipulator to this point, its end effector will most likely be perpendicular to the object's surface, thus allowing the laser sensor to correctly measure the exact distance to the object. The approximation point (A_p) was calculated through equation (6.1), along with the centroid's coordinates (C) and the normal vector ($\vec{n}$), both in relation to the Kinect's frame `camera_rgb_optical_frame`. In equation (6.1) the k symbolizes the scale which represents the distance from A_p to C. Since the laser sensor readings oscillate more in distances smaller than 200 mm, k was set to 250 mm. Figure 6.1 represents the schematics of equation (6.1).

$$A_p = C + k \times \vec{n} \tag{6.1}$$

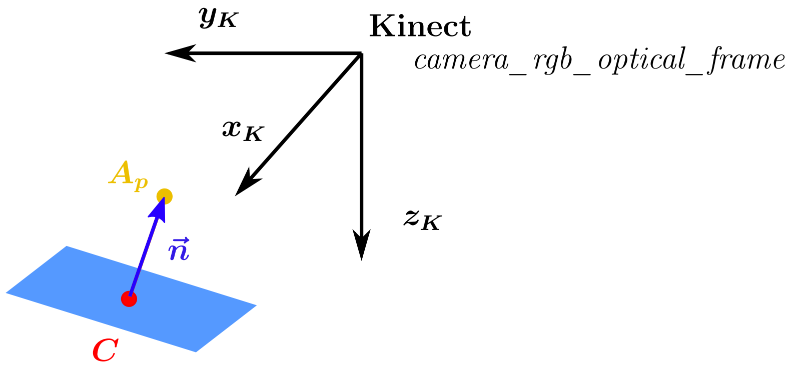


Figure 6.1: Schematics of the determination of the approximation point (A_p) with the centroid (C) and the surface's normal vector ($\vec{n}$).

6.2 Normal vector in the global frame

The surface's normal of an object is calculated through the coordinates of the approximation point (A_p) and the centroid (C), both already defined in relation to the global frame (`robot_base_link`). To do so, the following three equations are used, starting by equation (6.2) which determines the vector between the two points. Afterwards, the components of this vector are used in equation (6.3), which determines the vector's length for the calculation of its unit vector in (6.4). Figure 6.2 presents the schematics of the surface's normal for a better understanding of the equations. The normal vector is now oriented in the opposite direction since it is more convenient for the future determination of the grasping point.

$$\overrightarrow{A_pC} = (x_C - x_{A_p}, y_C - y_{A_p}, z_C - z_{A_p}) \tag{6.2}$$

$$length = \|\overrightarrow{A_pC}\| = \sqrt{{x_{A_pC}}^2 + {y_{A_pC}}^2 + {z_{A_pC}}^2} \tag{6.3}$$

$$\widehat{A_pC} = \frac{\overrightarrow{A_pC}}{length} \tag{6.4}$$

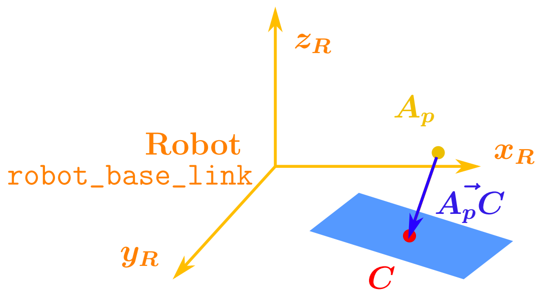


Figure 6.2: Schematics of the surface's normal ($\vec{A_pC}$) of the object to grab in relation to the global frame.

For a better perception of the two variables explained in the previous sections, figure 6.3 displays both the approximation point and the surface's normal, of a real object, in a point cloud acquired by the Kinect.

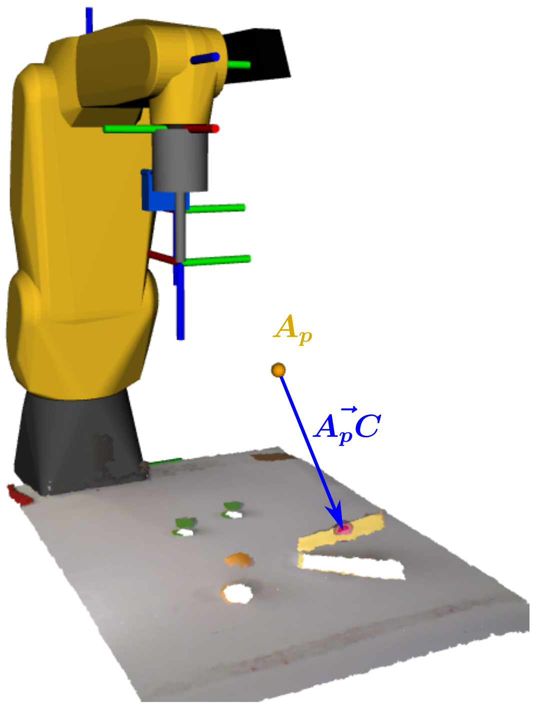


Figure 6.3: The approximation point (A_p) and the surface's normal ($\vec{A_pC}$) of an object detected by the Kinect.

6.3 Calculation of the Euler angles

The Euler angles describe the robot's end effector orientation, which has three rotational degrees of freedom. The coordinate frame of the end effector and the respective Euler angles are represented in figure 6.4. The *Roll*, *Pitch* and *Yaw* angles represent the rotation around the OZ, OY and OX axis, respectively.

In order to grab the objects, the OZ axis of the end effector has to be coincident with the surface's normal vector. However, a vector has only two rotational degrees of freedom, because rotation about the "axis" of the vector has no effect. Therefore, it is only possible to calculate the *Pitch* and *Yaw* angles. Since there is no need to move the end effector around the OZ axis the *Roll* angle will be 180° through the entire process. The *Pitch* and *Yaw* angles can be calculated through the normal vector components and equations (6.6) and (6.7), respectively. Figure 6.4 also presents a schematics of both angles for a better understanding.

$$Roll = \phi = 180^\circ \tag{6.5}$$

$$Pitch = \theta = \arctan\left(\frac{n_x}{\sqrt{{n_y}^2 + {n_z}^2}}\right) \tag{6.6}$$

$$Yaw = \psi = \arctan\left(\frac{-n_y}{n_z}\right) \tag{6.7}$$

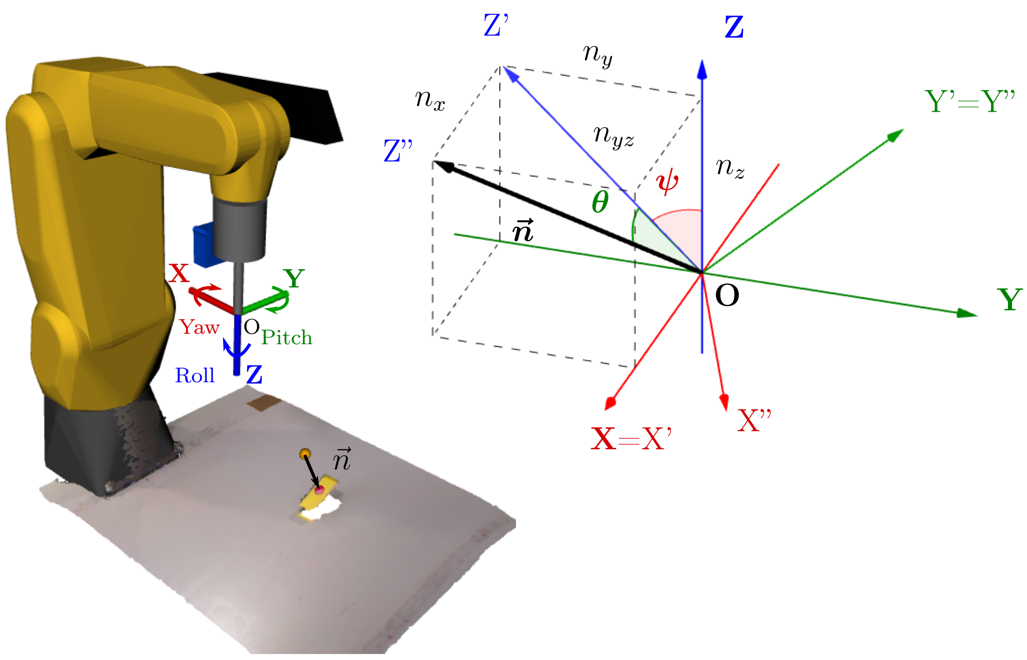
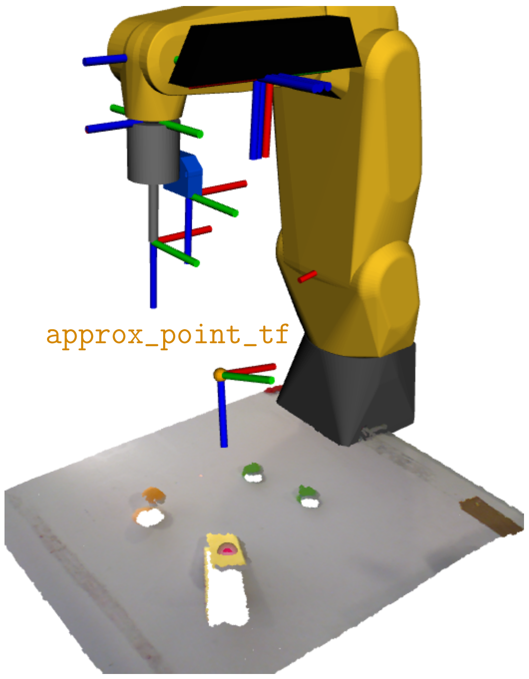


Figure 6.4: Schematics that represent the end effector's Euler angles (Yaw and Pitch).

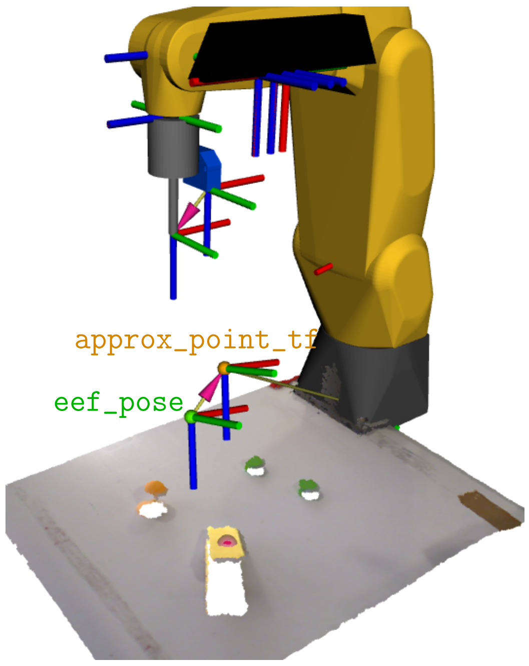
6.4 End effector position for measurement with laser sensor

In order to incorporate the laser sensor distance information, the frame of the laser's emitter has to be coincident with the approximation point, and its OZ axis has to be perpendicular to the object's surface. Therefore, it is necessary to determine where the end effector's frame will be in relation to the global frame (`robot_base_link`) when the laser's emitter is coincident with the approximation point. To do so, the transformation from the robot base to the end effector (${}^{R}T_{E}$) is determined through the static transformation from the end effector to the laser sensor (${}^{E}T_{L}$), the approximation point coordinates (A_p) and the desired Euler angles.

To compute this transformation, first, a `tf` broadcaster is created, called `approx_point_tf`, with the global frame as the parent frame, placed in the approximation point and with the orientation given by the already calculated Euler angles. The approximation point and its frame can be seen in figure 6.5a. Afterwards, another `tf` broadcaster is created, called `eef_pose`, now with the previous `tf`, the `approx_point_tf`, as the parent frame and with its origin at the same coordinates as the translation from the end effector to the laser sensor. This translation was previously determined through the *lookupTransform* function. Figure 6.5b presents the end effector's position and its frame in relation to the `tf` `approx_point_tf`. Finally, the coordinates of the origin of the last `tf` in relation to the global frame is then determined through the *lookupTransform* function from the *tf::Transformer* class, thus obtaining the end effector's position for the laser reading. The end effector's position is represented in relation to the global frame, in figure 6.5b, by a green dot.



(a) Frame centered on the approximation point with the orientation of the Euler angles (`approx_point_tf`).



(b) Frame of the end effector's position for the laser sensor measurement (`eef_pose`).

Figure 6.5: Schematics of the coordinate frames created for the determination of the end effector's position for the measurement with the laser sensor.

Having the end effector in its determined position and with the respective orientation, the laser emitter will coincide with the approximation point and measure the exact distance to the object's centroid through the surface's normal, as in figure 6.6.

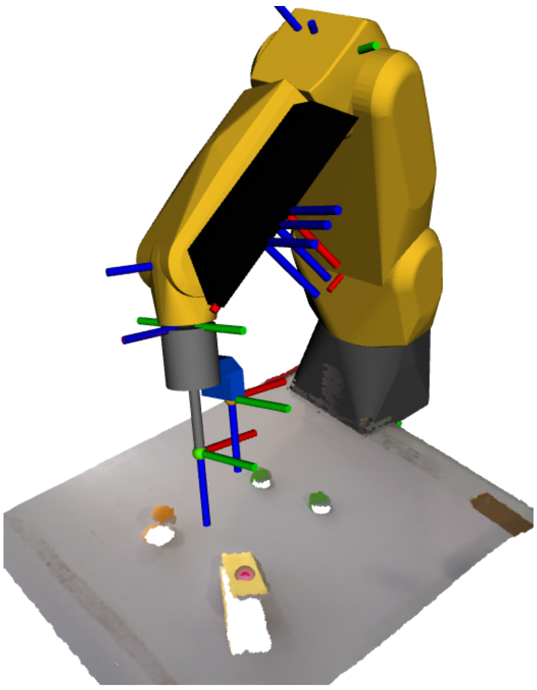


Figure 6.6: End effector in the correct position for the laser sensor to measure the exact distance from the approximation point to the object.

6.5 Determination of the grasping point

As already explained, the point cloud acquired from the Kinect does not describe the grasping point of an object in a very precise way; that is why the laser sensor is used. To calculate the exact grasping point of an object, the process explained in 6.1 used to determine the approximation point, was once more used, but now with the laser sensor information. Therefore, through equation (6.8) together with the approximation point (A_p), the vector surface's normal ($\widehat{A_pC}$) and the laser sensor's reading it is then possible to determine the grasping point (G_p). This process is represented in figure 6.7 for a better understanding.

$$G_p = A_p + laser\_reading \times \widehat{A_pC} \tag{6.8}$$

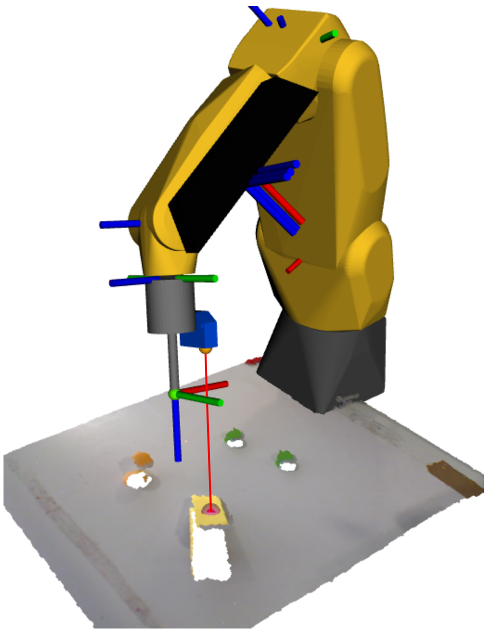


Figure 6.7: Determination of the grasping point through the combination of the coordinates of the approximation point with the laser sensor information.

The grasping point (G_p) represents the real position of the object's centroid and, as can be seen in figure 6.8, it has a small offset from the centroid determined through the segmentation of the Kinect's point cloud.

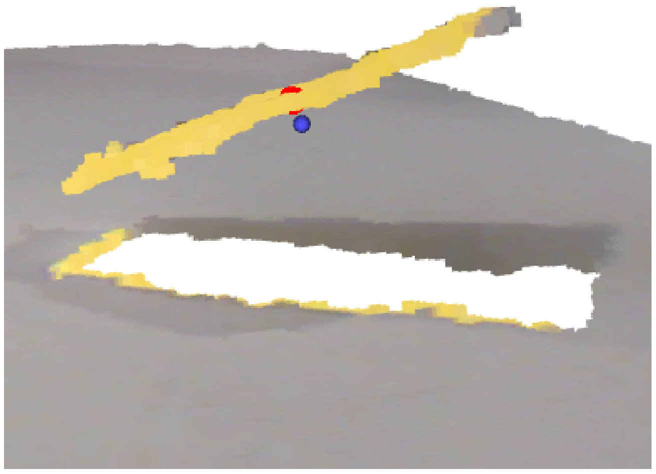


Figure 6.8: Representation of the grasping point (blue dot) and the object's centroid (red dot) for comparison.

6.6 Flow of the global process

In order to combine all the sensor's data and also establish the communication with the robot, different nodes and launch files were created for the development and organization of the entire bin-picking process. The main node is the `move_fanuc.py` and is the one responsible for the communications with the FANUC's server, i.e., send the commands to move the robot and receive its current state. This program is also the one responsible for launching the remaining nodes at the correct moment in time.

The developed bin-picking process is composed of 4 major stages, each corresponding to a different position of the manipulator. In the first stage the workspace is analysed and several information is obtained for the determination of the positions of the following stages. The process proceeds then with the manipulator moving to the previously computed end effector position for the distance measurement with the laser sensor. The third stage will be the positioning at the approximation point and finally, the process is completed with the final stage which is therefore the grasping of the object. There can also be a 5th stage to the process which consists of returning to the approximation point for safety reasons.

These stages are schematized in figure 6.9 and are fully described below.

Appendix D explains which files to launch and which nodes to run in order to launch the planning environment and all the communications to control the manipulator, thus performing the developed bin-picking process.

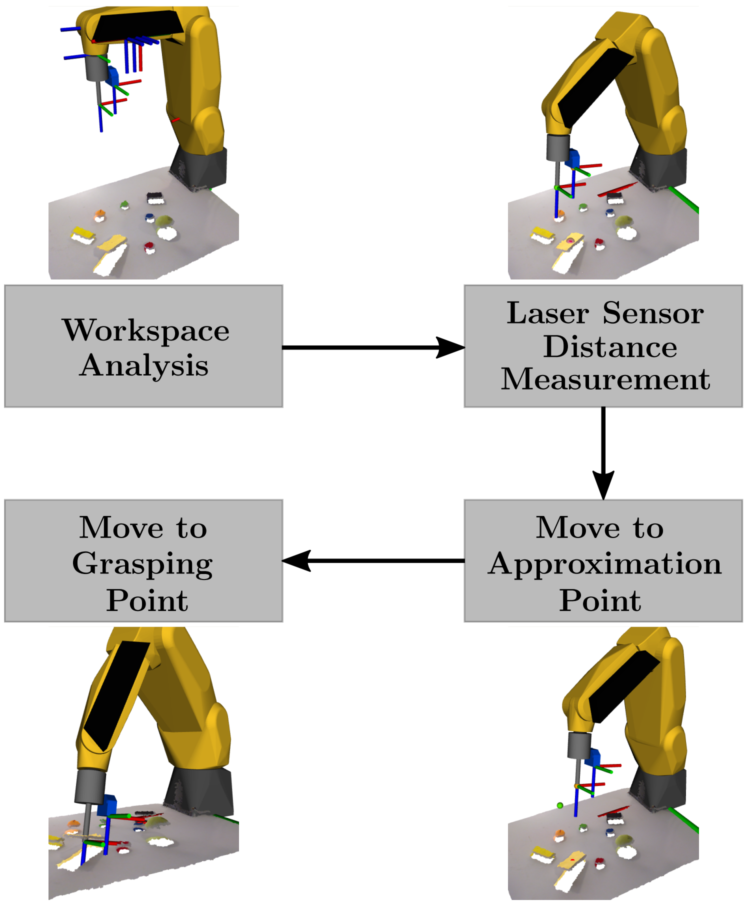
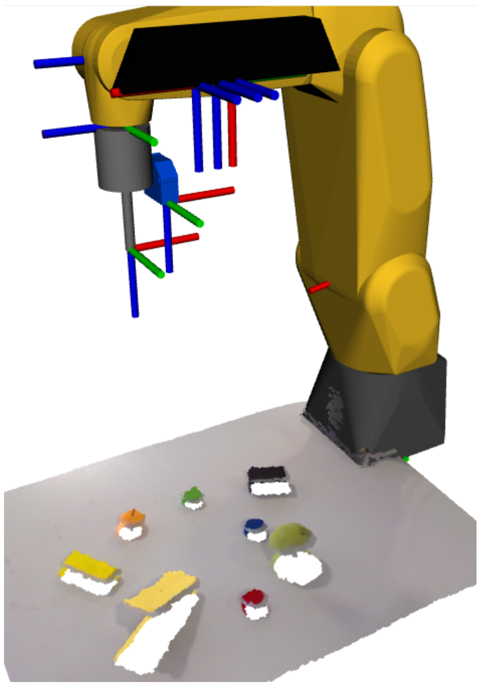

Workspace Analysis
Laser Sensor Distance Measurement
Move to Approximation Point
Move to Grasping Point

Figure 6.9: The major stages of the developed bin-picking process.

The initial commands in the `move_fanuc` node are responsible for moving the robot to the first position. This position, represented in figure 6.10, allows the Kinect to properly see the entire workspace.

First Position: Workspace Analysis


(a) Robot's virtual model.

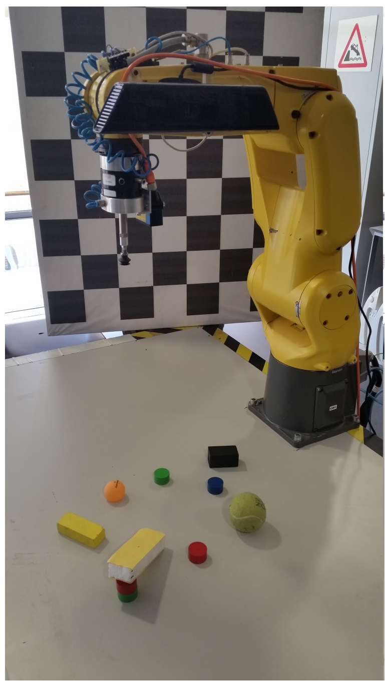

(b) Real robot.

Figure 6.10: Robot's initial position for workspace analyze.

In this first position, the `move_fanuc.py` node launches a file which will run both the `objDetection` and the `pointTFtransfer` nodes. The `objDetection` node, as already explained in the segmentation chapter 5, is the one responsible for computing and publishing the coordinates of the centroid of the object to grab and its normal. The centroid is published in the topic `/cloud_centroid` and its normal in the `/cloud_centroid_normal`. Both are defined in the Kinect's frame, more specifically the `camera_rgb_optical_frame`. Figure 6.11 presents the robot's virtual model with the centroid of the first object to grab, marked with a red dot in the point cloud.

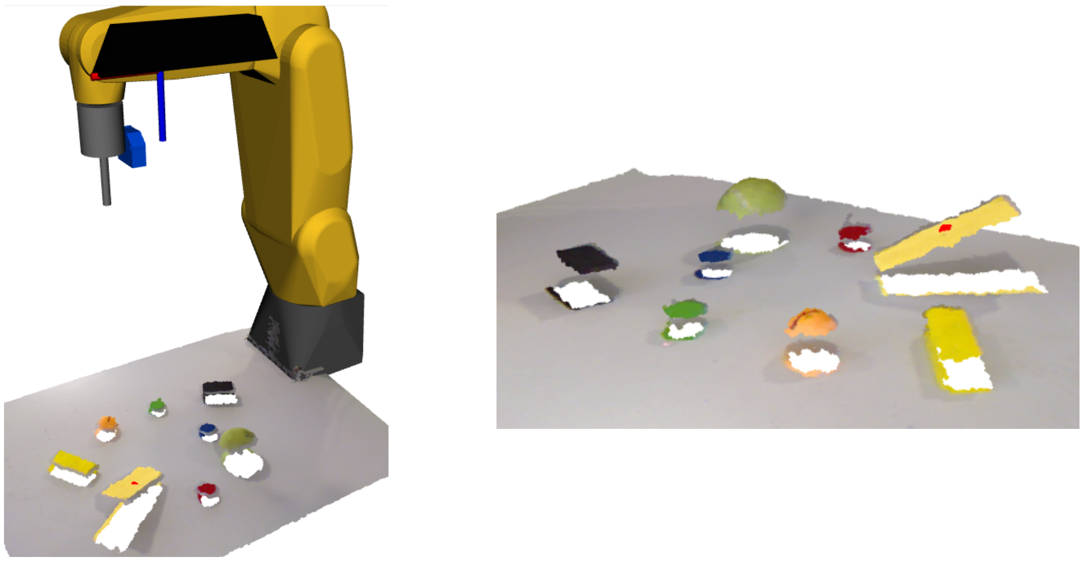


Figure 6.11: Representation, with a red dot, of the centroid of the object to grab in relation to the Kinect's frame.

To move the robot to a desired point, the `move_fanuc.py` node has to know the point's exact coordinates and the end effector's orientation, in order to then send it to the manipulator's server. To do so, the centroid's coordinates have to be defined in the global frame, which is the `robot_base_link`. Consequently, a program that transforms the coordinates from the Kinect's frame, `camera_rgb_optical_frame`, to the robot's frame, `robot_base_link`, had to be created. The `pointTFtransfer` node is the one responsible for this task. This subscribes to the topics published by the `objDetection` node, calculates the transformation between the frames `robot_base_link` and `camera_rgb_optical_frame` and calculates the point's coordinates in relation to the first frame.

This node is also the one responsible for the determination of the approximation point coordinates, the end effector position for the measurement with the laser sensor, and the Euler angles for the end effector's orientation. As explained in section 6.1, the approximation point is previously determined with the centroid coordinates and the normal vector in relation to the Kinect's frame, and was also transformed to be defined in the global frame. The centroid and the approximation point were then used, as described in section 6.2, to determine the unit vector between those two points, thus having the surface's normal already in relation to the global frame as well. This normal vector is then used to compute the Euler angles as explained in section 6.3. The surface's normal will also be later used for determining the grasping point coordinates together with the laser sensor reading. Finally, the end effector's position, for the measurement with the laser sensor of the exact distance, is determined as explained in section 6.4. The approximation point, the surface's normal, the Euler angles and the end effector's position are all then stored in a ROS message called `TargetsPose` which contains three *Vector3*, for the 3 points and the normal vector, and one *Pose2D* for the Euler angles, since only the yaw and the pitch angles are computed.

Figure 6.12 presents the robot's virtual model with the approximation point marked with a yellow dot and the end effector's position with a green dot, both already defined in the global frame. There is also a bigger pink dot, which represents the centroid of the object defined in the global frame, to confirm the frame conversion since it is coincident with the centroid represented by the red dot, which is defined in the Kinect's frame. It is possible to see in figure 6.13 an amplified image of both centroids mentioned in the previous sentence.

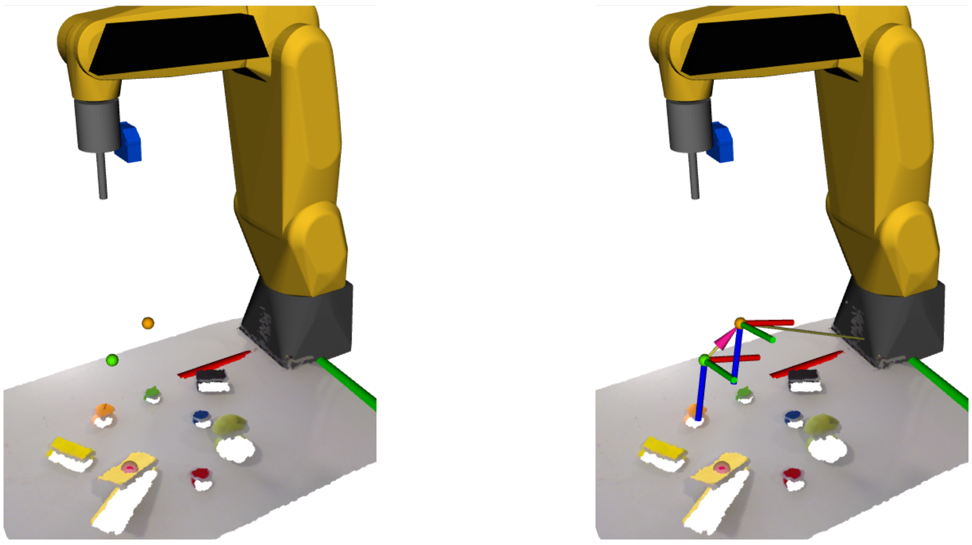


Figure 6.12: Representation of the approximation point (yellow dot) and the end effector's position (green dot) in the point cloud.

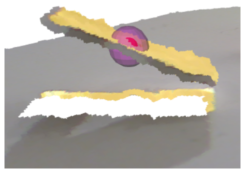
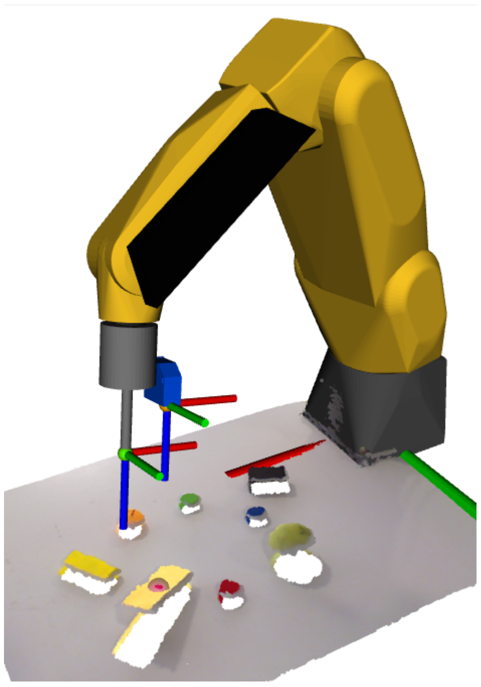


Figure 6.13: Representation of both centroids, one in relation to the Kinect's frame (red dot) and the other in relation to the global frame (bigger pink dot) for the frame conversion confirmation.

The `move_fanuc.py` node will subscribe to the ROS msg published by the `pointTFtransfer` node and after storing the information in the corresponding variables, it will terminate both the `objDetection` and `pointTFtransfer` nodes. The subscribed information is then used to compute the path the manipulator will perform from the current position to the calculated end effector's position. This is done by computing several *waypoints* to follow between the current state and the desired pose. Those points are then used to compute the Cartesian path, through the *compute_ cartesian_ path* function, with an interpolation resolution of 1 cm and a jump threshold of 0.0, thus effectively disabling any jumps. The *compute_ cartesian_ path* function returns a fraction and the path to be executed. The returned fraction represents the percentage of success of the generated path. The path will only be executed if the fraction is equal to 1.0 and the manipulator will then move to the second position, figure 6.14.

Second Position: Laser Sensor Distance Measurement


(a) Robot's virtual model.

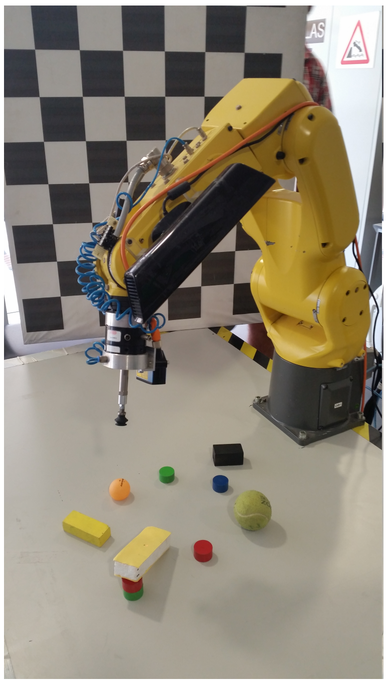

(b) Real robot.

Figure 6.14: End-effector's positioning for laser measurement of the distance from the approximation point to the object's surface.

After commanding the manipulator to position itself in the second position, the `move_fanuc.py` node will launch the file that runs the `sensorRS232` node. This is the one responsible for opening a serial connection with the laser and reading the data that comes through the port. Since the laser's readings always fluctuate between two values, as previously explained in section 3.1.3, the data will be stored in a vector of 10 measurements and the average of those measurements will be then determined. These 10 readings were collected in order to eliminate undesired readings since they sometimes oscillate between more than two values. The `sensorRS232` node will publish a ROS msg with the average of the measurements which will be then subscribed by the `move_fanuc.py` node.

After having the laser sensor reading stored in a variable, the `move_fanuc.py` node will terminate the `sensorRS232` node. This variable is then used to determine the coordinates of the grasping point as previously explained in section 6.5. The grasping point is represented in the point cloud of figure 6.15 with a blue dot.

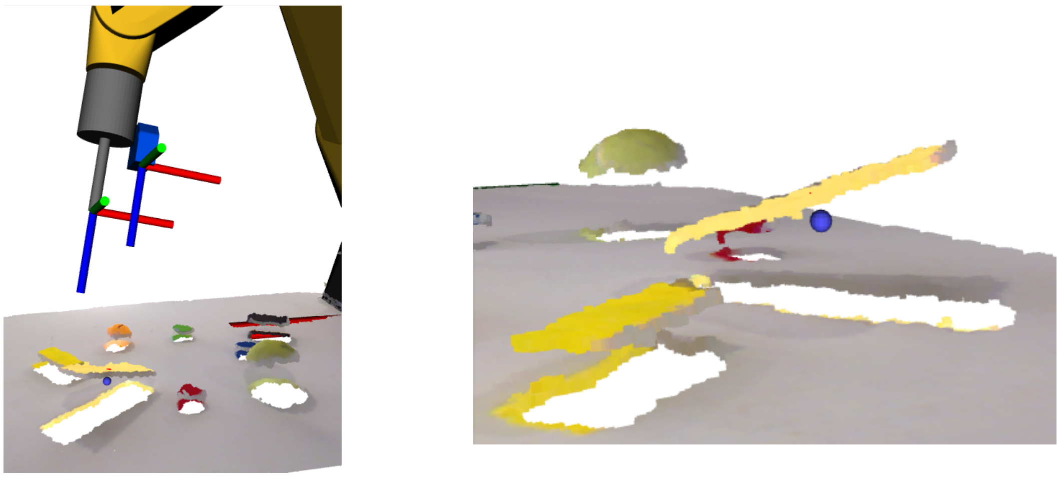
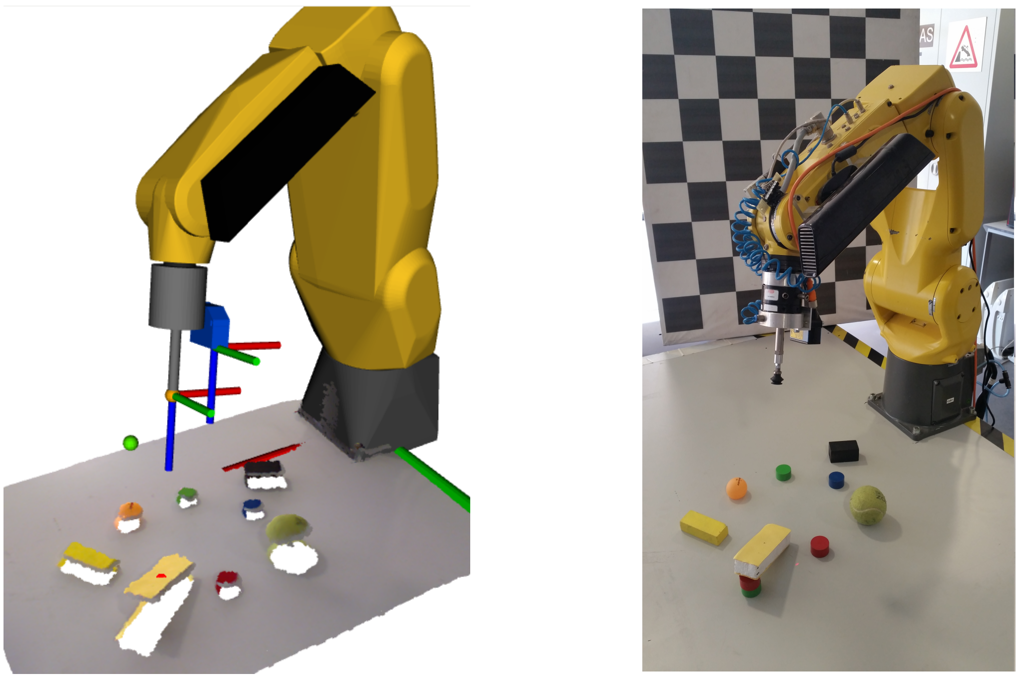


Figure 6.15: Representation of the grasping point in the point cloud as a blue dot.

To ensure that the end effector moves closer to the grasping point already with the correct orientation and through the surface's normal, the following end effector's position will be the approximation point, to only then move correctly to the grasping point. Therefore, the third robot's position of this bin-picking process will be with the tool tip at the approximation point. Once more, the `move_fanuc.py` node will compute the Cartesian path and execute the plan if it was accomplished with a success of 100%. The 3rd position of the manipulator is represented in figure 6.16.

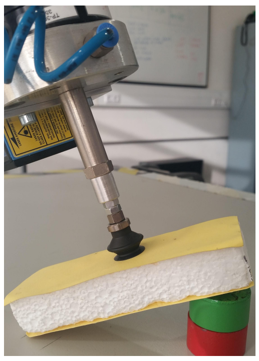
Third Position: Approximation Point



(a) Robot's virtual model. (b) Real robot.

Figure 6.16: Robot's end effector at the approximation point (yellow dot).

Having the gripper already in the approximation point, it can now move to the grasping position through a perpendicular path to the object's surface, thus ensuring that the suction cup will be parallel to its surface or the surfaces' tangent plane in case of a curved object as in figure 6.17.


(a)

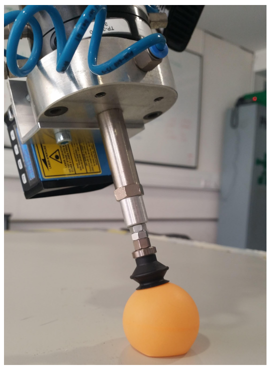
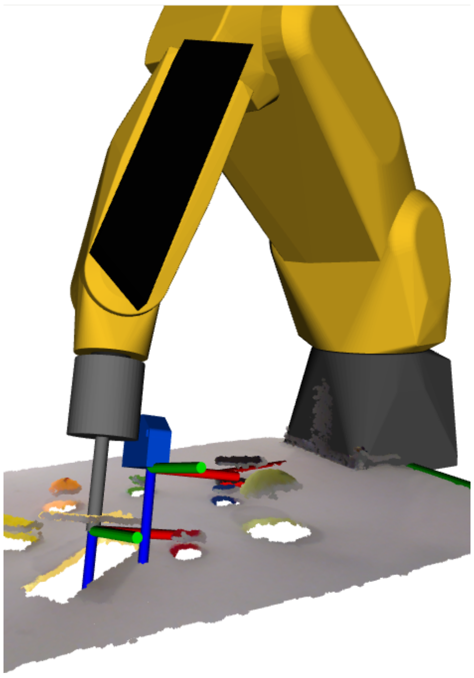

(b)

Figure 6.17: Gripper properly positioned to grasp different objects, one a flat surfaced object, (a), and the other a curved object, (b).

In the manipulator's 4th position, which is represented in figure 6.18, the tooltip is coincident with the grasping point and the gripper is ready to correctly grab the object without deforming it.

Fourth Position: Grasping Point


(a) Robot's virtual model.

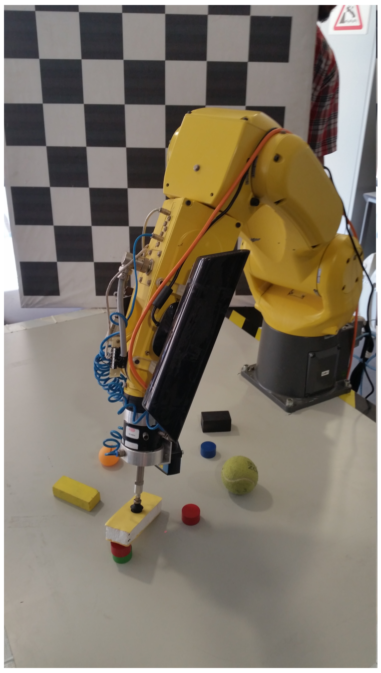

(b) Real robot.

Figure 6.18: Robot's end effector at the grasping point (blue dot) ready to catch the object.

In order to grab an object, the suction I/O (Input/Output) has to be activated. However, the meta-packages which interact with the FANUC manipulator (ROS-Industrial and MoveIt!) are not yet prepared for the control of the robot's I/Os. Therefore, the search for a viable solution to solve this problem was necessary. The ex-student Vítor Silva has already developed, in [79], a functional approach to solve the problem of the I/O control and that was also the one used in this project. The developed solution lies in the utilization of a microcontroller, connected with the computer, which, through a circuit, commands relays to actuate the manipulator's digital inputs which will in turn dispute the digital outputs. Figure 6.19 presents the I/O's control unit created by Vítor Silva.

The microcontroller used is an Arduino Leonardo ETH and is responsible for the initialization of the Ethernet connection, thus allowing an exchange of messages under the TCP/IP protocol. An application server is also initialized to interact with a TCP/IP client which sends the commands for motorizing and turning an output on or off.

The `vs_IO_client` node is the one responsible for the communication with the I/O's control unit. This node is a TCP/IP client which connects with the Arduino through the same IP address and Port defined in the server, thus allowing a bidirectional communication between the two running processes in different devices (socket communication). This client receives, decodes and processes a ROS msg, sent by the `move_fanuc.py` node, with the commands to correctly activate the I/Os. This message will change/monitor the state of the suction I/O, which is number 8, by turning it on, through function 2, turning it off, through function 3, or just monitor the output, through function 1.

However, the valve that controls the compressed air in the gripper is bi-stable so, in order to change its state correctly, an output has to be turned on and the other turned off. Therefore, Vítor Silva also created a TP program that controls two opposite state outputs through a single input, thus avoiding the unnecessary use of other inputs. This TP program has to be continuously executed in parallel with the other ROS programs, required for running the server. To do so, the Background Logic option, available in the FANUC controllers, was used. This option allows the continuous execution of a program, in the background, without interfering with other main programs already running. So, it is necessary to run, with the administrator login, the TP program called `MONIO` in the Background Logic.

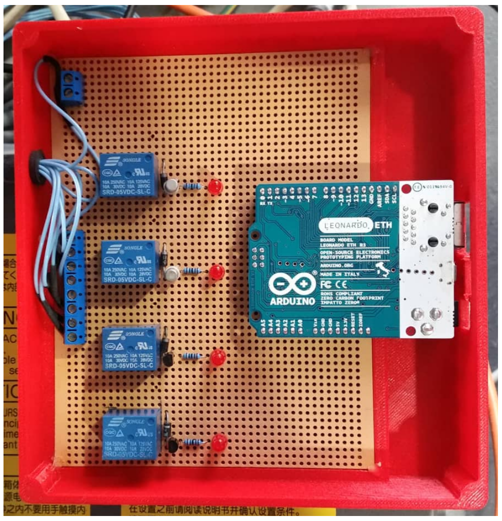


Figure 6.19: I/O's control unit.

Having already grasped the desired object the manipulator can then return to the approximation point to guarantee that the grasped object does not collide with the other items.

Fifth Position: Return to Approximation Point

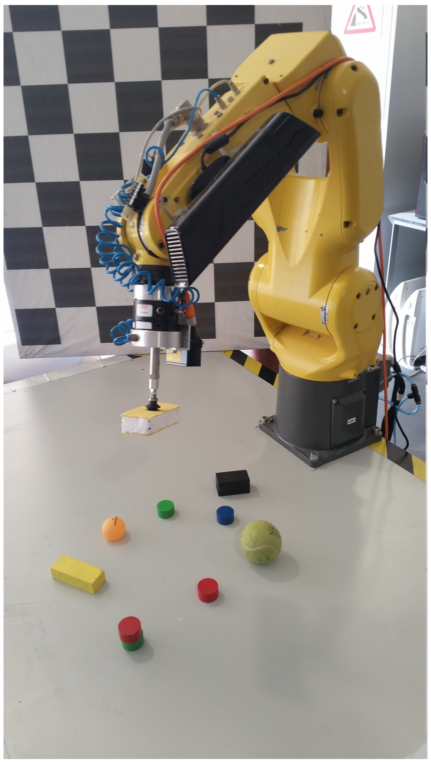


Figure 6.20: Robot returns to the approximation point with the grasped object.

From here, the manipulator can now place the object in a desired predefined position, like a box or a treadmill.

The overall node communication and the topics published and subscribed through the entire process are briefly presented in figure 6.21 for a better understanding of how the full system works.

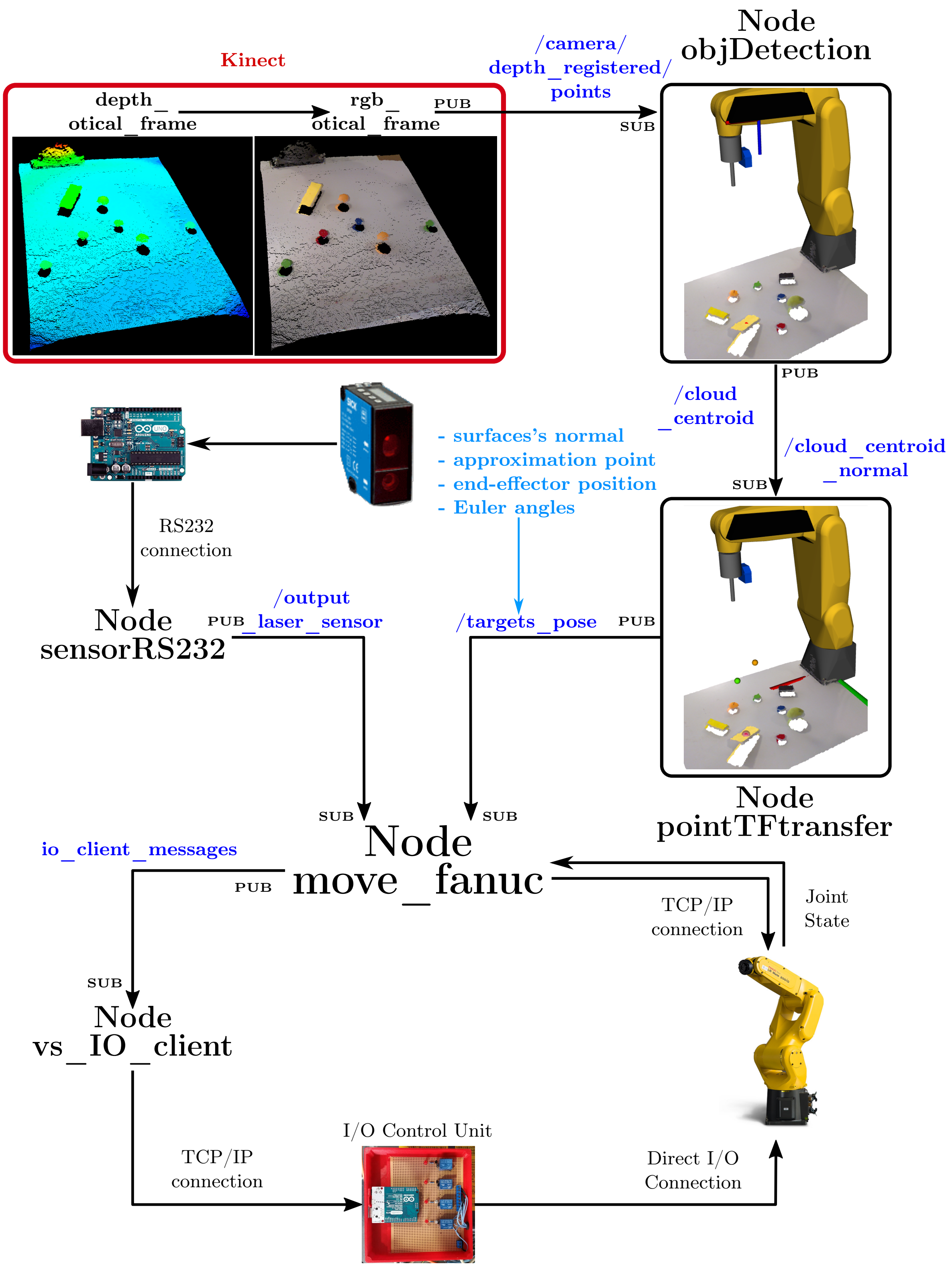


Figure 6.21: Overall node communication.

Chapter 7

Experiments and Results

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

For this particular project, this sort of assessment could be done by evaluating the number of successes in the identification and grasping of different objects. Consequently, three tests were taken with three distinct objects each and a fourth one with a variety of distinctive shaped objects. All of these tests consist of detecting, grabbing and placing the objects in a small box, following the steps presented in the chapter 6 of the System Integration.

7.1 Performed Tests

The objects used in the first test were small cylindrical pieces, with a diameter of 37mm, as the ones in figure 7.1. By using these small objects, it is possible to check the precision and accuracy of the process. Some of the pieces were placed in the workspace table with a inclination to force the manipulator to position itself with different Euler angles. Figure 7.1 presents the first test's starting position, where 20 small pieces would be supposedly grabbed and placed inside the box on the left of the figure.

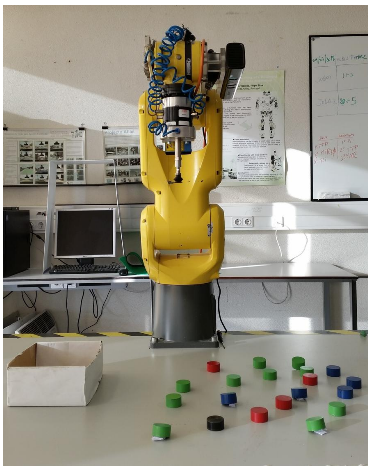
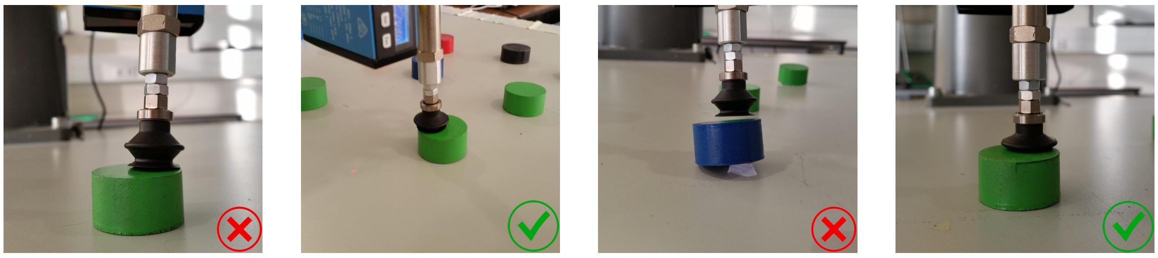


Figure 7.1: First test with small cylindrical pieces.

In the 20 tested pieces 6 of them were not successfully grabbed, which represents a percentage of success of 70%. This low percentage is caused by several easily visible problems. One cause of error is in the determination of the surface's normal, which sometimes is not properly computed. By positioning the end effector with the Euler angles computed with the wrong surface's normal, the suction cup will not be parallel to the object's surface, thus not allowing the object to be gripped, as in figure 7.2a.

Another cause of error is in the determination of the objects centroid, explained in the chapter 5 of the Segmentation of the Point Clouds. Since the Kinect has a low precision, the centroid's coordinates can vary up to 3mm from the correct centroid. This flaw is not quite relevant for the main purpose, which is for playing games by identifying and tracking individual players; nonetheless, for the purpose of developing this precise bin-picking process, the error could prevent the correct grasp of smaller objects. In this specific test, this error was not one of the reasons for not catching those 6 pieces, however the piece in figure 7.2b was almost not grabbed because of this, which means that objects smaller than the ones used in this test could have failed more often because of this flaw.

Apparently, the main cause of error in the picking of objects derives from the oscillation in the laser sensor readings. This, then, can cause two opposite situations, the tool tip could be too far from the object when grabbing it, as in figure 7.2c, but also too close, as in figure 7.2d. The piece in the last case was obviously grabbed but if it had been a more fragile object it could have broken it.



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

Figure 7.2: Main causes of error in the correct picking of an object in the first test. (a)-Error caused by the incorrect determination of the surface's normal. (b)-Error caused by the incorrect determination of the object's centroid given the lack of precision in Kinect's depth sensor. (c)-Error caused by the incorrect laser sensor distance reading, which was inferior than the real distance. (d)-Error caused by the incorrect laser sensor distance reading, which was greater than the real distance.

In the second test, the objects used were ping pong balls. With this test it is possible to demonstrate the versatility of this process by having the capability of grabbing objects with curved surfaces. Figure 7.3 presents the initial setup of the second trial, where 5 ping pong balls were scattered in front of the manipulator. The process held in the first test was replicated four times with the ping pong balls since there were only 5 balls to pick, thus having also a sampling of 20 objects for the experiment.

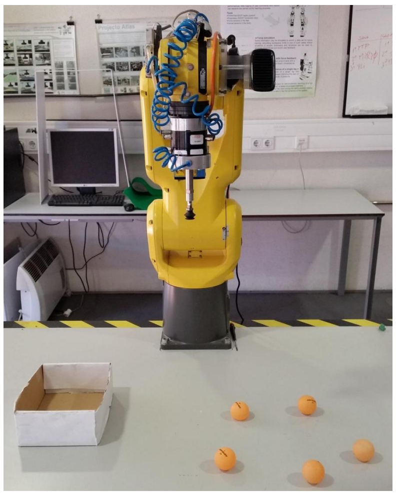


Figure 7.3: Second test with ping pong balls.

Coincidentally, in the second trial, 6 ping pong balls were also not grabbed. However, in this case, the main cause of error resulted from the incorrect determination of the surface's normal. This is explained by the fact that normal vectors in curved surfaces vary along the object's surface, unlike flat surfaces in which the normal vectors have all almost the same orientation. From the 6 pieces not grabbed, 3 of them occurred because of this specific issue, like the one in figure 7.4a . Nevertheless, in other times the surface's normal was poorly determined but since the tool tip got close enough to the surface, the gripper was able to grab the object, as in the case presented in figure 7.4b.

The other two issues observed in the first trial were, as expected, also reported in this second experiment. One object was not grabbed because of the incorrect determination of the object's centroid, presented in figure 7.4c, and the other 2 were due to the incorrect measurement of the distance with the laser sensor, as in figure 7.4d.

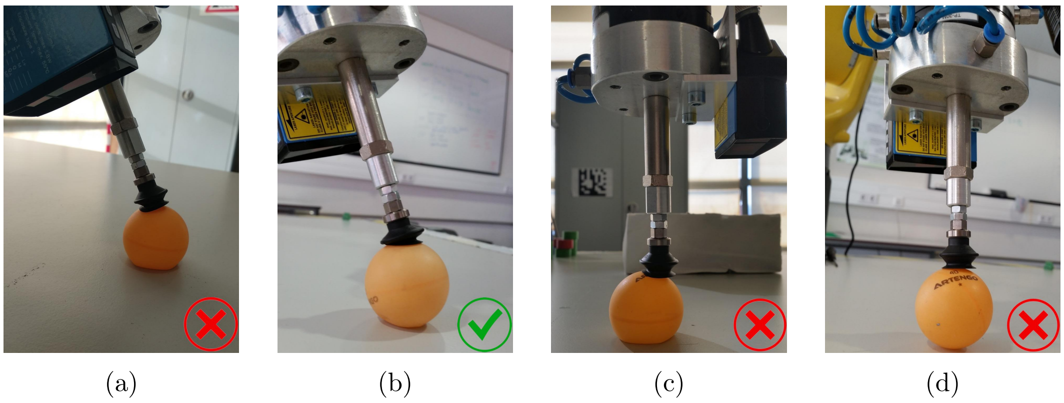


Figure 7.4: Main causes of error in the correct picking of an object in the second test. (a)-Error caused by the incorrect determination of the surface's normal. (b)-Object grabbed despite the incorrect determination of the surface's normal. (c)-Error caused by the incorrect determination of the object's centroid given the lack of precision in Kinect's depth sensor. (d)-Error caused by the incorrect laser sensor distance reading.

In order to solve the errors in the laser sensor's readings, firstly, the Arduino, the circuit and all the connections, used to receive and interpret the sensor's output, were placed in a box, as in figure 7.5, thus protecting all the connections. Afterwards the laser sensor was re-calibrated with another process to ensure that the calibration curve, that links the sensor's distance displayed in its monitor and the reading received by the Arduino, was as precise as possible. Since the laser sensor was now attached to the manipulator, contrarily to the first time calibrating, it was possible to use the robot's accuracy to preform a better calibration. This time the calibration was held not only with the sensor's distance displayed in its monitor but also with the exact position of the robot's end effector. So, two linear regressions were then computed to evaluate the relation between both these informations with the readings received by the Arduino. Both curves are presented in figure 7.6. Through the Z coordinate of the robot's end effector it is possible to ensure that the collected data is always equally spaced, thus obtaining a more precise slope for the calibration curve. Therefore, the equation that converts the reading to a distance was altered to include the scope of the linear regression that uses the end effector's coordinates (blue curve in figure 7.6) and the intercept of the curve that uses the distance displayed in the sensor's monitor (red curve in figure 7.6). Consequently the Arduino's code was changed and uploaded.

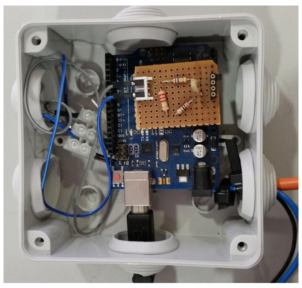


Figure 7.5: Box with the Arduino and all the connections properly stored.

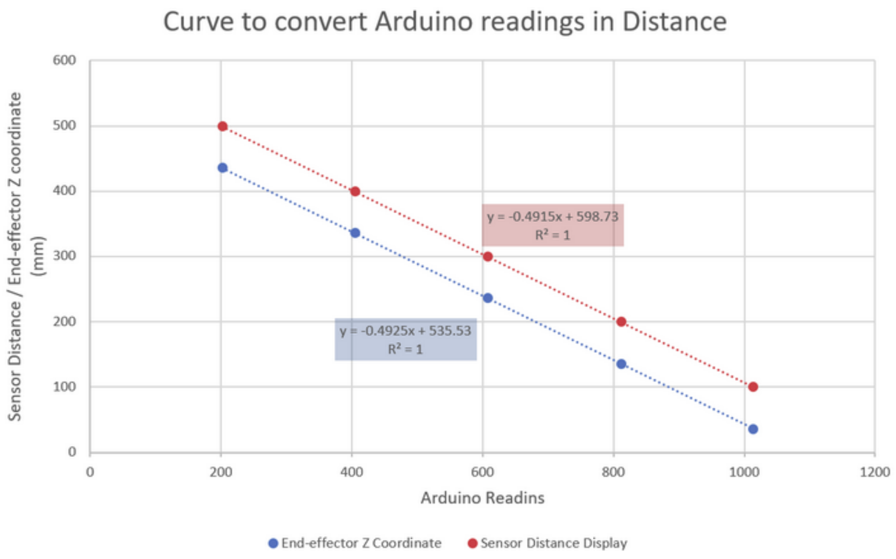


Figure 7.6: Graph with the two linear regressions used for the re-calibration of the laser sensor's readings .

After the alteration in the laser sensor's calibration curve a third test was conducted. In this third trial the objects used were eggs, thus demonstrating that this process is sufficiently precise to grab fragile objects without deforming or breaking them. The eggs used were previously emptied in order to prevent the egg's content to enter the pipes with compressed air, used for the suction, in case of breaking. Consequently, by using empty eggs these will be even more fragile and harder to grab without breaking. Figure 7.7 presents the initial setup of the third trial in which the procedure used in the second trial was replicated since there were only 5 eggs as well.

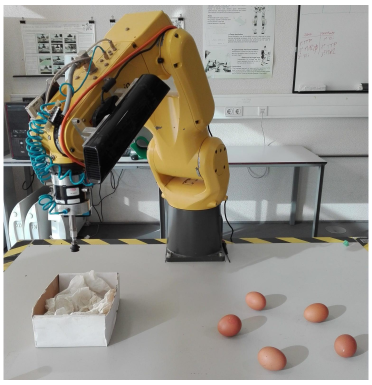


Figure 7.7: Third test with eggs.

Now, from the 20 eggs used in this test 18 were successfully grabbed, and figures 7.8a, 7.8b and 7.8c present some of those successes. From the two unsuccessfully grabbed eggs, one was due to the incorrect determination of the surface's normal, presented in figure 7.8d, and the other was caused by something not yet witnessed in these trials. Since the eggs become lighter when emptied, at the moment the end effector moves closer to more distant objects, the suction cup, sometimes, can push those objects when approaching them. This will cause situations has the one visualized in figure 7.8e. To solve this problem the suction can be activated when the manipulator is approaching the object and not after.

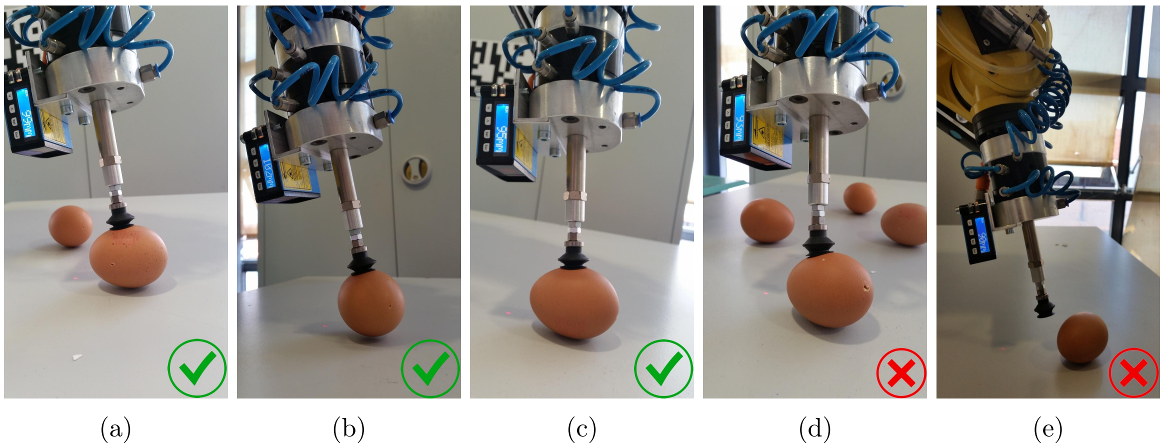


Figure 7.8: Successes and failures of the third trial with eggs. Figure (a), (b) and (c) present three successfully grabbed objects. (d)-Error caused by the incorrect determination of the surfaces's normal. (e)-Object not grabbed because it was pushed by the tool tip.

The last and fourth test was performed now with a variety of distinctive shaped objects. Figure 7.9 presents the last test's starting position, where cobbles, eggs, ping pong balls, duct tapes and small cylindrical objects where scattered in front of the manipulator, within the limits marked on the table. Those limits represent the intersection between the area observed by the Kinect and the space in which the robot can safely move. Once more, the process applied in the other tests was used, and the objects were picked from the largest object to the smallest, i.e., objects represented by point clouds with more points were the first to be picked.

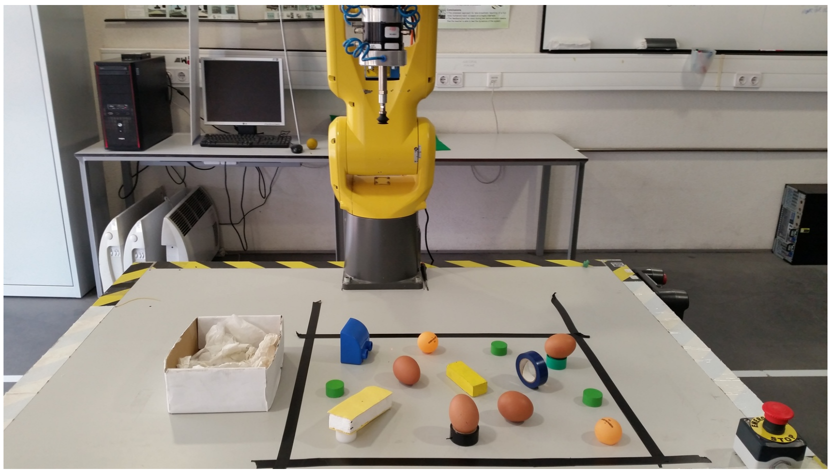
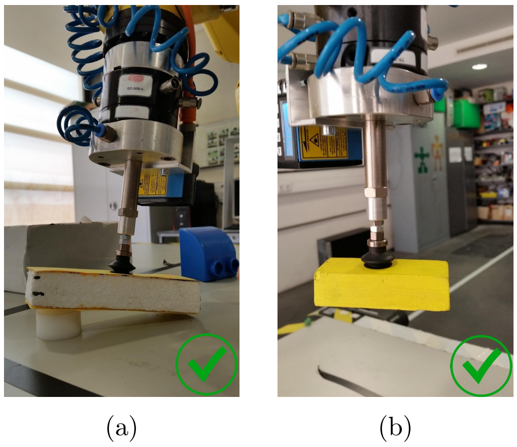


Figure 7.9: Fourth test with a variety of distinctive shaped objects.

The two objects with flat surfaces were the first successfully grabbed at the first attempt. Both successes are presented in figure 7.10.



(a) (b)

Figure 7.10: Flat-surfaced pieces successfully grabbed in the fourth test.

The following largest object in the bin-picking process was the Lego piece. Since the surface of the piece captured by the Kinect was too inclined the determination of the correct surface's normal kept failing, as can be seen in figure 7.11a in which the object was not grabbed because of this problem. However, after moving the Lego to a less inclined position, presented in figure 7.11b, the object was then grabbed at the first attempt, figure 7.11c.

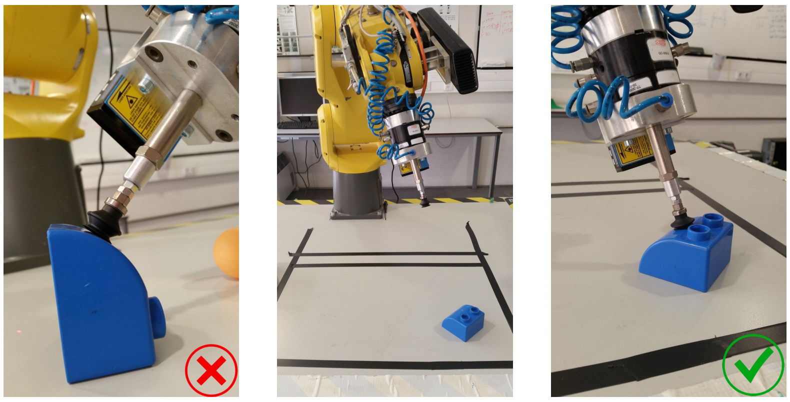

(a) First attempt. (b) New position. (c) Second attempt.

Figure 7.11: Experiment performed on the fourth test with a Lego piece.

The next object to be picked was the duct tape, which despite being represented by a big point cloud it is a thin object and the grasping area must be properly centered for the suction cup to correctly grab it. Therefore, the first attempt, presented in figure 7.12a, was unsuccessful. However, it took only one other try to successfully pick up the duct tape, which was recorded in figure 7.12b.

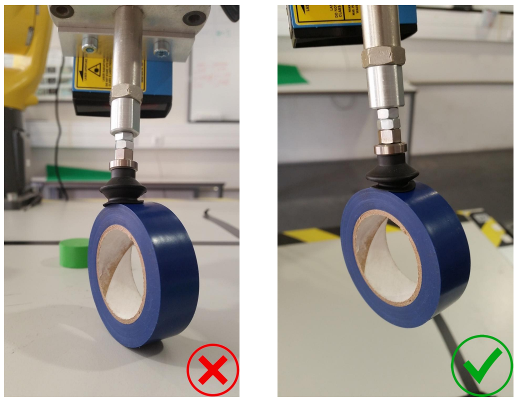

(a) First attempt. (b) Second attempt.

Figure 7.12: Experiment performed on the fourth test with a duct tape.

Afterwards the eggs were all correctly grabbed on the first attempt except for one. The moments of the grasping of the 3 eggs were captured and are presented in figure 7.13. The other egg was grabbed after two attempts, however, this one was positioned differently from the others, as can be seen in figure 7.14. In this position the observed surface of the egg will be its top which has a more accentuated curvature, making the determination of the correct surface's normal even harder, which was the cause of error in the first unsuccessful attempt, presented in figure 7.14a. The second attempt, shown in figure 7.14b, was unsuccessful due to the fact that the suction cup is required to be closer to the object in this conditions even though the laser reading had been correct.

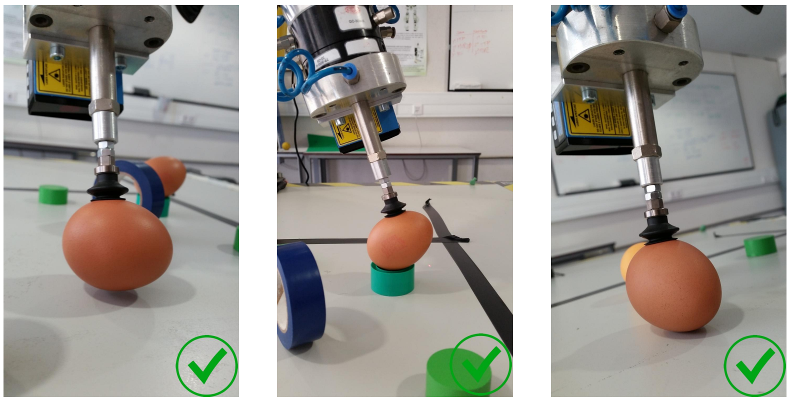

(a) (b) (c)

Figure 7.13: Eggs successfully grabbed in the fourth test.

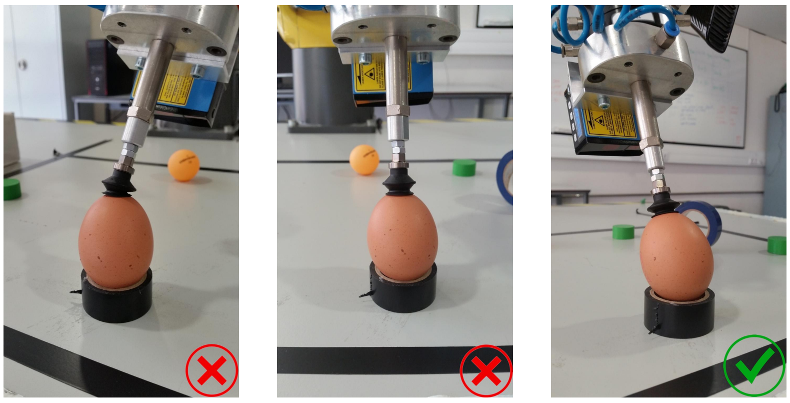

(a) First attempt. (b) Second attempt. (c) Successful attempt.

Figure 7.14: Experiment performed on the fourth test with an egg vertically positioned.

The test proceeded to the grasping of the smaller objects, starting with the ping pong balls. The first ball was grabbed only at the second attempt, moment recorded in figure 7.15b, once more due to a incorrect determination of the surface's normal. Figure 7.15a shows the first attempt to pick the first ball. The second ball could not be picked at the current position since it was too close to the manipulator and the *MoveIt!* planner was never able to generate a 100% successful plan to grab the object in those conditions. For this reason a second marking was used to better visualize the manipulators limits, which can be seen in figure 7.15c near the ping pong ball. However, the *MoveIt!* planner was able to generate the movement to the centroid of the cylindrical piece, present in the same figure. This happened because, in oder to grab this flat object which is parallel to the table, the end effector would have to be perpendicular instead of reclined, which would be the necessary position to grab the ball.

Finally, the last small cylindrical pieces were all successfully grabbed on the first attempt.

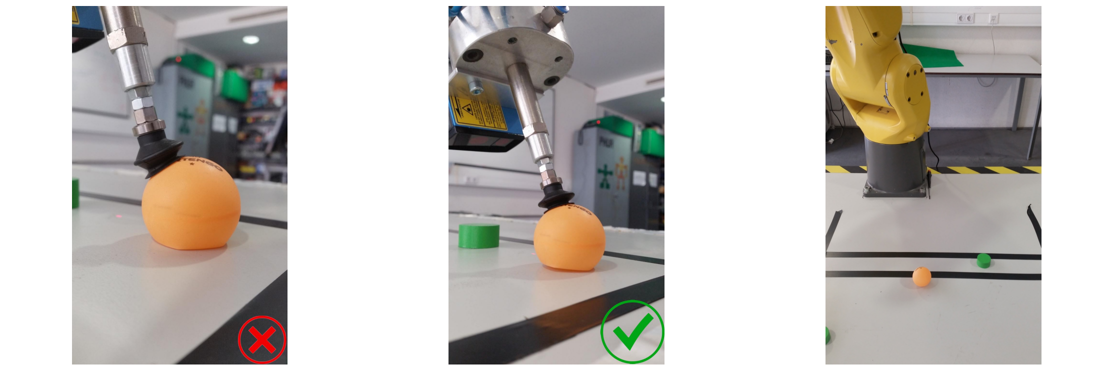

(a) First attempt of the first ping pong ball.
(b) Second successful attempt of the first ping pong ball.
(c) Second ping pong ball's repositioning.

Figure 7.15: Experiment performed on the fourth test with ping pong balls.

In conclusion, from the 13 objects used in this fourth test, 8 were grabbed on the first attempt and the remaining after one or two other attempts. It should also be noted that through all the tests with fragile objects, as the eggs, none of those were ever deformed or broken in the grasping process, at most they were not collected.

7.2 Demonstration

Since one of the main objectives of this project is the creation of an active perception unit for the detection of objects with high accuracy for further manipulation, it is fundamental to demonstrate the capability of replicating the bin-picking process in different settings. To do so, the robot together with the Kinect and the laser sensor were placed on a mobile platform which belongs to the ROBONUC platform, [79]. The ROBONUC, presented in figure 7.16a, was developed in the Laboratory for Automation and Robotics (LAR) for the combination of a mobile robot with a manipulator. Therefore, this platform is composed by the Robuter II which is a mobile platform and a FANUC manipulator. Figure 7.16b shows the manipulator with the developed vision system already on the the Robuter II.

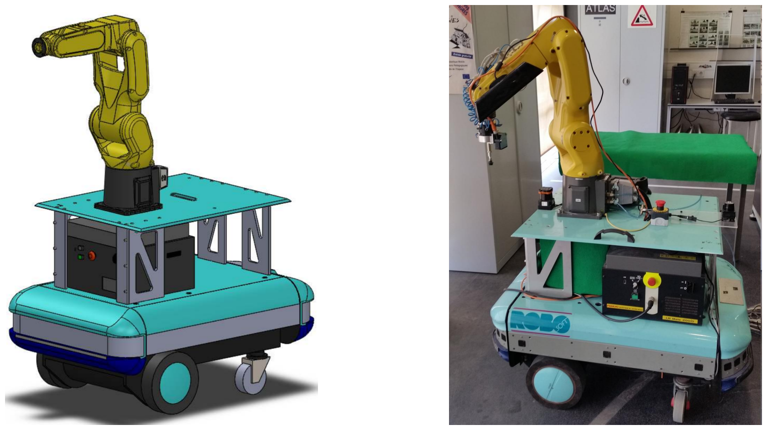

(a) Model created by Vítor Silva
(b) Real system.

Figure 7.16: ROBONUC Platform.

The demonstration was performed similarly to the previous tests after the mobile platform moved itself closer to a table with different scattered objects, placing the manipulator in the correct position to perform the bin-picking process of the objects, as in figure 7.17. The results obtained in this demonstration were similar to the ones from the fourth test, thus demonstrating the potential of this bin-picking process in different settings.

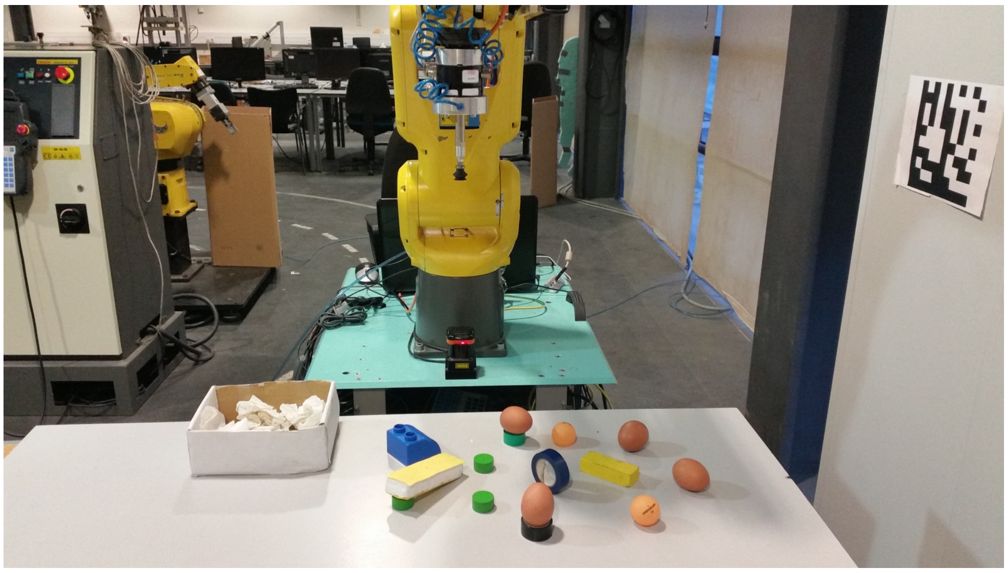


Figure 7.17: Demonstration starting position.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

During the course of this project, a solution for the execution of a precise bin-picking process capable of grasping small and fragile objects without breaking or deforming them was accomplished, thus achieving the main objective. This was accomplished through the creation of a bin-picking system that combines the information of a 3D sensor (Kinect) with the measurement of a precise 1D laser sensor.

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

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

Upon obtaining and processing the Kinect's data, one of the main objectives of this project was achieved through the combination of the already obtained information with the laser sensor's measurement. While the Kinect's point cloud provides the coordinates of the object on the table, the laser sensor complements with a more accurate Z coordinate, thus obtaining a more exact grasping position.

By combining all of the acquired information a possible bin-picking solution was created which falls upon the combination of four major stages: workspace analysis, positioning for the laser distance measurement, positioning at the approximation point and finally grasping of the object. The grasping position is computed in the second stage and represents the most prominent and innovative feature of this project.

At last, the successfully performed tests demonstrated the viability and the reliability of the developed bin-picking process. However, it should be noted that more favorable results could be achieved with the combination of the precise laser sensor with a 3D sensor more robust capable of providing a more accurate depth data. Therefore reducing the majority of errors caused by the incorrect determination of the surface's normal, since the laser sensor is already capable of aiding in the determination of the exact coordinates of the grasping point. Additionally, this type of hardware is not recommended to be used in an industrial environment since the Kinect is not prepared to work through long periods of time.

8.2 Future Work

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

The main intervention should be in the improvement of the segmentation of the Kinect's point cloud. Since the program developed to cluster and identify the objects in the workspace only works if they are not in close contact with each other, it should be refined in order to be able to identify different objects when they are near each other or even when they are piled up inside a bin. The algorithm that computes the surface's normal should also be thoroughly examined or replaced by a better one. Additionally, instead of processing a point cloud of only one position the robot could move above the workspace and combine different views of the environment for a better identification of the objects centroid, which will later be useful for the determination of a more adequate grasping point.

In the determination of the grasping position, there is also the possibility of using a gripper with two suction tool tips, thus allowing for a more stable manipulation of bigger objects. In this case, the determination of both grasping points can still be determined with one laser sensor or two equally-spaced as the tool tips.

For safety reasons, there should be an enhancement in collision avoidance for more than just self-collisions, which is the case at the moment.

In the identification/determination of the final position of the bin-picking process, i.e., the place were the object will be dropped, the Kinect can once more be used to determine the final destination point.

Finally, since this system was integrated in the ROBONUC platform at the final stage of this project there is still the need to organize all of the connections and power supplies for all of the used hardware, in order to allow the platform to be mobile. This includes also the installation on the platform of an air compressor for the activation of the suction gripper.

References

[1] The Economist. *The growth of industrial robots - Daily chart.* 2017. URL: https://www.economist.com/blogs/graphicdetail/2017/03/daily-chart-19 (visited on 03/09/2018).

[2] RobotWorx. *Advantages and Disadvantages of Automating with Industrial Robots.* URL: https://www.robots.com/blog/viewing/advantages-and-disadvantages-of-automating-with-industrial-robots (visited on 03/06/2018).

[3] Alex Owen-Hill. *Robot Vision vs Computer Vision: What's the Difference?* 2016. URL: https://blog.robotiq.com/robot-vision-vs-computer-vision-whats-the-difference (visited on 03/06/2018).

[4] Inc EPIC Systems. *Quick History of Machine Vision.* 2017. URL: https://www.epicsysinc.com/blog/machine-vision-history (visited on 03/06/2018).

[5] Blumenbecker. *Robot applications | Blumenbecker Robotics.* URL: https://www.robotics.blumenbecker.com/robot-applications/ (visited on 03/06/2018).

[6] teqrma - Vision Guided Robotics. *Bin Picking - The revolutionary technique for industry 4.0.* 2017. URL: https://teqram.com/solutions/bin-picking/ (visited on 03/06/2018).

[7] RIA - Robotic Industries Association. *Robotics Industry Insights - The Pervasive Relevance of Bin Picking in Nature and Business.* 2011. URL: https://www.robotics.org/content-detail.cfm?content_id=3080 (visited on 03/07/2018).

[8] FANUC. *Vision functions for industrial robots.* URL: https://www.fanuc.eu/es/en/robots/accessories/robot-vision (visited on 06/01/2018).

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

[10] FANUC. *Intelligent offline 3D robot simulation with ROBOGUIDE - Fanuc.* URL: https://www.fanuc.eu/uk/en/robots/accessories/roboguide (visited on 06/02/2018).

[11] Pick-it. *Wide variety of products | Pick-it - Advantages.* 2017. URL: https://www.pickit3d.com/advantages/wide-variety-of-products (visited on 06/02/2018).

[12] Pick-it. *Bin picking with Pick-it.* 2017. URL: https://www.pickit3d.com/applications/bin-picking (visited on 06/02/2018).

[13] Pick-it. *3D Camera Products.* 2017. URL: https://www.pickit3d.com/product/pick-it-3d-camera (visited on 06/02/2018).

[14] Pick-it. *Pick-it 3D camera manual-m.pdf - Google Drive.* URL: https://drive.google.com/file/d/0B87wvNgWersoYk5BejBJTnl3eWUtcjZWLUYxdGdQbTh3bndJ/view (visited on 06/02/2018).

[15] Pick-it. *Pick-it software | Pick-it.* 2017. URL: https://www.pickit3d.com/product/pick-it-software (visited on 06/02/2018).

[16] Pick-it. *3D vision processor | Pick-it.* 2017. URL: https://www.pickit3d.com/product/pick-it-vision-processor (visited on 06/02/2018).

[17] Pick-it. *Pick-it product: Robot vision made easy.* 2017. URL: https://www.pickit3d.com/product (visited on 06/02/2018).

[18] Luís Filipe Pinto Rodrigues. "Bin-picking de objectos toroidais". MSc Thesis. Aveiro, 2010. URL: http://lars.mec.ua.pt/public/LARProjects/BinPicking/2010_LuisRodrigues/.

[19] Lutz Plümer Jan Dupuis, Stefan Paulus, Jan Behmann and Heiner Kuhlmann. "A Multi-Resolution Approach for an Automated Fusion of Different Low-Cost 3D Sensors". PhD thesis. 2014, p. 17. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4029635/pdf/sensors-14-07563.pdf.

[20] the free encyclopedia Wikipedia. *David Laserscanner.* 2018. URL: https://en.wikipedia.org/wiki/David_Laserscanner (visited on 06/03/2018).

[21] Mohammad Biglarbegian Kevin Tai, Abdul-Rahman El-Sayed, Mohammadali Shahriari, Mahmud, and Shohel. "State of the Art Robotic Grippers and Applications". 2016. URL: http://www.mdpi.com/2218-6581/5/2/11.

[22] S. I. Cho; S. J. Chang; Y. Y. Kim; K. J. An. "Development of a Three-degrees-of-freedom Robot for harvesting Lettuce using Machine Vision and Fuzzy logic Control". 2002. URL: https://www.sciencedirect.com/science/article/pii/S1537511002900619.

[23] T.J. Dodd A. Pettersson , T. Ohlsson, S. Davis, J.O. Gray. "A hygienically designed force gripper for flexible handling of variable and easily damaged natural food products". 2011. URL: https://www.sciencedirect.com/science/article/pii/S1466856411000427.

[24] Darwin G. Caldwell Davis, S., J.O. Gray. "An end effector based on the Bernoulli principle for handling sliced fruit and vegetables". 2006. URL: https://www.sciencedirect.com/science/article/pii/S0736584506001347?via{\%}3Dihub{\#}fig6.

[25] *Soft grippers improve bin picking | ProFood World.* URL: https://www.profoodworld.com/articles/soft-grippers-improve-bin-picking (visited on 09/28/2018).

[26] FANUC. *FANUC LRMate 200iD/SH industrial Robot.* URL: https://www.fanuc.eu/de/en/robots/robot-filter-page/lrmate-series/lrmate-200-id (visited on 05/14/2018).

[27] Pedro Hernandez. *Microsoft Pulling the Plug on Popular Kinect Sensor.* 2017. URL: http://www.eweek.com/pc-hardware/microsoft-quits-manufacturing-kinect-motion-controller (visited on 09/28/2018).

[28] William Wong. *How Microsoft's PrimeSense-based Kinect Really Works.* 2011. URL: http://www.electronicdesign.com/embedded/how-microsoft-s-primesense-based-kinect-really-works (visited on 05/14/2018).

[29] Matt Rosoff. *The Story Behind Kinect, Microsoft's Newest Billion Dollar Business - Business Insider*. 2011. URL: http://www.businessinsider.com/the-story-behind-microsofts-hot-selling-kinect-2011-1#a-small-incubation-team-figured-out-the-challenges-2 (visited on 05/14/2018).

[30] Ahmed Ben Jmaa et al. "A New Approach For Hand Gestures Recognition Based on Depth Map Captured by RGB-D Camera". In: *Computación y Sistemas* 20.4 (2016). ISSN: 2007-9737. DOI: 10.13053/cys-20-4-2390. URL: https://www.researchgate.net/publication/316560415_A_New_Approach_For_Hand_Gestures_Recognition_Based_on_Depth_Map_Captured_by_RGB-D_Camera.

[31] Microsoft Corporation. *Kinect Sensor*. 2012. URL: https://msdn.microsoft.com/en-us/library/hh438998.aspx (visited on 05/14/2018).

[32] *SICK*. URL: https://www.sick.com/ag/en/ (visited on 05/15/2018).

[33] *DT20-P224B | Distance sensors | SICK*. URL: https://www.sick.com/de/en/distance-sensors/displacement-measurement-sensors/dt20-hi/dt20-p224b/p/p215239 (visited on 09/28/2018).

[34] Sensor Inteligence SICK. *DISPLACEMENT MEASUREMENT SENSORS*. Tech. rep. 2017. URL: https://www.sick.com/media/pdf/9/39/239/dataSheet_DT20-P224B_1040405_en.pdf.

[35] OMRON Corporation. *Displacement Sensors / Measurement Sensors*. 2007. DOI: DOI10.4207/PA.2011.ART40Francescod'ErricoatCNRS-UniversityofBordeaux. arXiv: /ieeexplore.ieee.org/stamp/stamp.jsp?tp={\&}arnumber=6957576 [http:]. URL: https://www.ia.omron.com/support/guide/56/introduction.html (visited on 05/18/2018).

[36] Sensor Inteligence SICK. *DISPLACEMENT MEASUREMENT SENSORS*. Tech. rep. 2017. URL: https://www.sick.com/media/pdf/9/39/239/dataSheet_DT20-P224B_1040405_en.pdf.

[37] Sensor Inteligence SICK. *Distance Sensors - The complete product portfolio*. Tech. rep. 2008. URL: http://www.samey.is/_pdf/_sick/SICK_Lengdarnemar_Optical_and_Ultrasonic_Distance_Sensors.pdf.

[38] SICK. *Triangulation Receiver Technologies*. URL: http://sickusablog.com/wp-content/uploads/2016/09/Triangulation-Receiver-Technologies-SICK-White-Paper-1.pdf.

[39] Scott Rosenberger. *Understanding âĂIJTrue AnalogâĂİ Resolution | AUTOMATION INSIGHTS*. URL: https://automation-insights.blog/2010/03/02/understanding-âĂœtrue-analogâĂİ-resolution/ (visited on 05/16/2018).

[40] Arduino Store. *Arduino UNO Rev3*. 2014. URL: https://store.arduino.cc/usa/arduino-uno-rev3 (visited on 05/16/2018).

[41] the free encyclopedia Wikipedia. *Arduino Uno*. 2018.

[42] Open Source Robotics Foundation. *About ROS*. 2016. URL: http://www.ros.org/about-ros/ (visited on 05/17/2018).

[43] Generation Robots. *Robot Operating System (ROS)*. 2017. DOI: 10.1007/978-3-319-54927-9. URL: https://www.generationrobots.com/blog/en/2016/03/ros-robot-operating-system-2/ (visited on 05/17/2018).

[44] Dave Coleman. *urdf - ROS Wiki.* 2014. URL: http://wiki.ros.org/urdf (visited on 05/18/2018).

[45] Rethink Robotics. *Rviz - sdk-wiki.* 2015. URL: http://sdk.rethinkrobotics.com/wiki/Rviz (visited on 05/17/2018).

[46] Open Source Robotics Foundation. *ROS.org | Core Components.* 2015. URL: http://www.ros.org/core-components/.

[47] Dirk Thomas, Dorian Scholz, and Aaron Blasdel. *rqt - ROS Wiki.* URL: http://wiki.ros.org/rqt (visited on 05/17/2018).

[48] Matthew Robinson. *Industrial - ROS Wiki.* 2018. URL: http://wiki.ros.org/Industrial (visited on 06/24/2018).

[49] *Industrial/Install - ROS Wiki.* 2017. URL: http://wiki.ros.org/Industrial/Install (visited on 06/24/2018).

[50] Ioan A. Sucan and Sachin Chitta. *MoveIt! Motion Planning Framework.* 2017. URL: https://moveit.ros.org/ (visited on 06/18/2018).

[51] GvdHoorn. *Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot - ROS Wiki.* 2017. URL: http://wiki.ros.org/Industrial/Tutorials/Create_a_MoveIt_Pkg_for_an_Industrial_Robot (visited on 07/01/2018).

[52] *Point Cloud Library | Willow Garage.* URL: http://www.willowgarage.com/pages/software/pcl (visited on 05/17/2018).

[53] Point Cloud Libary. *About - Point Cloud Library (PCL).* 2016. URL: http://pointclouds.org/about/ (visited on 05/17/2018).

[54] OpenNI. *Open-source SDK for 3D sensors.* 2015. URL: http://www.openni.ru/ (visited on 05/17/2018).

[55] *Robots/evarobot/Tutorials/indigo/Kinect - ROS Wiki.* URL: http://wiki.ros.org/Robots/evarobot/Tutorials/indigo/Kinect/ (visited on 05/17/2018).

[56] Arduino Software. "Arduino Software". In: *Processing* February (2010), pp. 1–12. DOI: 10.1007/978-1-4302-3883-6_3. URL: http://arduino.cc/en/Main/Software.

[57] I. Saito. *"tf - ROS Wiki".* 2015. URL: http://wiki.ros.org/tf (visited on 05/18/2018).

[58] G.A. van der Hoorn. *fanuc - ROS Wiki.* 2017. URL: http://wiki.ros.org/fanuc (visited on 05/18/2018).

[59] João Peixoto. *Library to communicate with the robCOMM server.* 2016. URL: http://lars.mec.ua.pt/public/LARProjects/Humanoid/2016_JoaoPeixoto/05ROSscripts/ROSCode/fanuc_control/src/ (visited on 06/18/2018).

[60] Yannick Morvan. "Pinhole Camera Model". In: *Computer Vision.* 2014, pp. 610–613. ISBN: 9780387307718. DOI: 10.1007/978-0-387-31439-6_472. URL: http://www.epixea.com/research/multi-view-coding-thesisse8.html.

[61] MATLAB. *What Is the Genetic Algorithm? - MATLAB & Simulink.* URL: https://www.mathworks.com/help/vision/ug/camera-calibration.html (visited on 05/21/2018).

[62] Alexander Reimann. *openni_launch/Tutorials/IntrinsicCalibration - ROS Wiki.* 2015. URL: http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration (visited on 05/21/2018).

[63] Adam Allevato. *camera_calibration/Tutorials/MonocularCalibration - ROS Wiki*. 2017. URL: `http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration` (visited on 05/22/2018).

[64] Jack Quin. *camera_info_manager_py - ROS Wiki*. 2013. URL: `http://wiki.ros.org/camera_info_manager_py` (visited on 06/18/2018).

[65] Fabien Spindler. *visp_hand2eye_calibration - ROS Wiki*. 2016. URL: `http://wiki.ros.org/visp_hand2eye_calibration` (visited on 05/22/2018).

[66] Jim Vaughan. *aruco_detect - ROS Wiki*. 2018. URL: `http://wiki.ros.org/aruco_detect` (visited on 05/22/2018).

[67] Saw Yer. *fiducials - ROS Wiki*. 2018. URL: `http://wiki.ros.org/fiducials` (visited on 05/22/2018).

[68] Isaac Saito. *Bags - ROS Wiki*. 2015. URL: `http://wiki.ros.org/Bags` (visited on 05/25/2018).

[69] Shobhit Chaurasia. *openni_launch/Tutorials/BagRecordingPlayback - ROS Wiki*. 2015. URL: `http://wiki.ros.org/openni_launch/Tutorials/BagRecordingPlayback` (visited on 05/28/2018).

[70] the free encyclopedia Wikipedia. *Random sample consensus*. 2018.

[71] Pcl. *Documentation - Point Cloud Library (PCL) - Plane model segmentation*. URL: `http://pointclouds.org/documentation/tutorials/planar_segmentation.php` (visited on 05/28/2018).

[72] Pcl. *Documentation - Point Cloud Library (PCL) - Extracting indices from a PointCloud*. URL: `http://pointclouds.org/documentation/tutorials/extract_indices.php#extract-indices` (visited on 05/28/2018).

[73] Pcl. *Documentation - Point Cloud Library (PCL) - Removing outliers using a Conditional or RadiusOutlier removal*. URL: `http://pointclouds.org/documentation/tutorials/remove_outliers.php` (visited on 05/28/2018).

[74] Pcl. *Documentation - Point Cloud Library (PCL) - Region growing segmentation*. URL: `http://pointclouds.org/documentation/tutorials/region_growing_segmentation.php` (visited on 05/28/2018).

[75] Point Cloud Library 1.8.1-dev (PCL). *pcl::search::KdTree< PointT, Tree > Class Template Reference*. 2018. URL: `http://docs.pointclouds.org/trunk/classpcl_1_1search_1_1_kd_tree.html` (visited on 05/28/2018).

[76] Pcl. *Documentation - Point Cloud Library (PCL) - How to use a KdTree to search*. URL: `http://pointclouds.org/documentation/tutorials/kdtree_search.php` (visited on 05/28/2018).

[77] Pcl. *Documentation - Point Cloud Library (PCL) - Euclidean Cluster Extraction*. URL: `http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php` (visited on 05/29/2018).

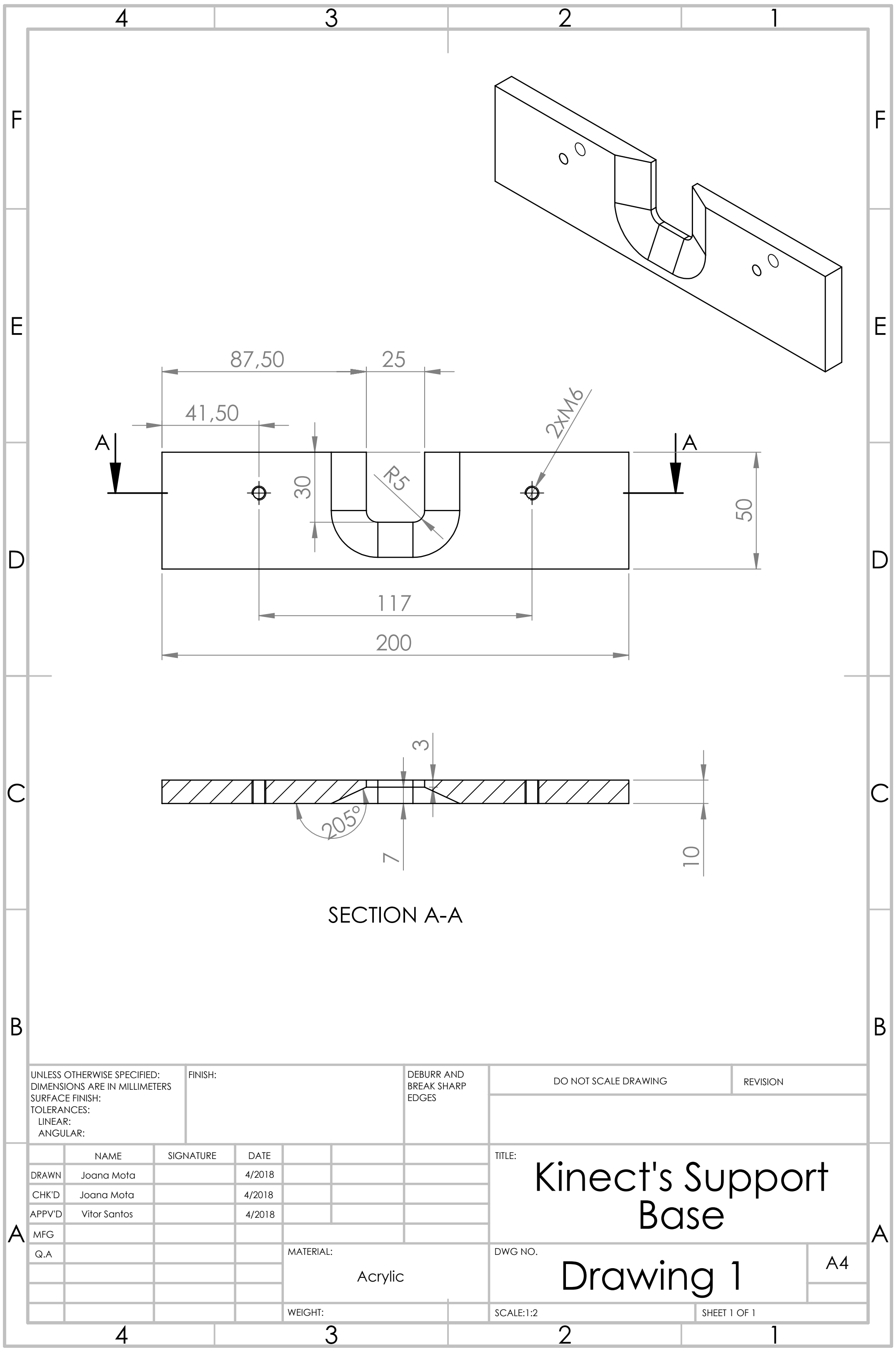
[78] Florian Echtler Jeff Kramer, Matt Parker, Daniel Castro, Nicolas Burrus. *Hacking the Kinect*. Ed. by Apress. magazine. 2012, p. 268.

[79] Vítor Silva. "Integração de Manipulador FANUC na Plataforma Robuter para Manipulação Móvel". MSc Thesis. Universidade de Aveiro, 2017.

Appendices

Appendix A

Definition drawings of the Kinect's support.



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND BREAK SHARP EDGES

DO NOT SCALE DRAWING

REVISION

	NAME	SIGNATURE	DATE
DRAWN	Joana Mota		4/2018
CHK'D	Joana Mota		4/2018
APPV'D	Vitor Santos		4/2018
MFG			
Q.A			

TITLE: Kinect's Support Base

MATERIAL: Acrylic

DWG NO. Drawing 1

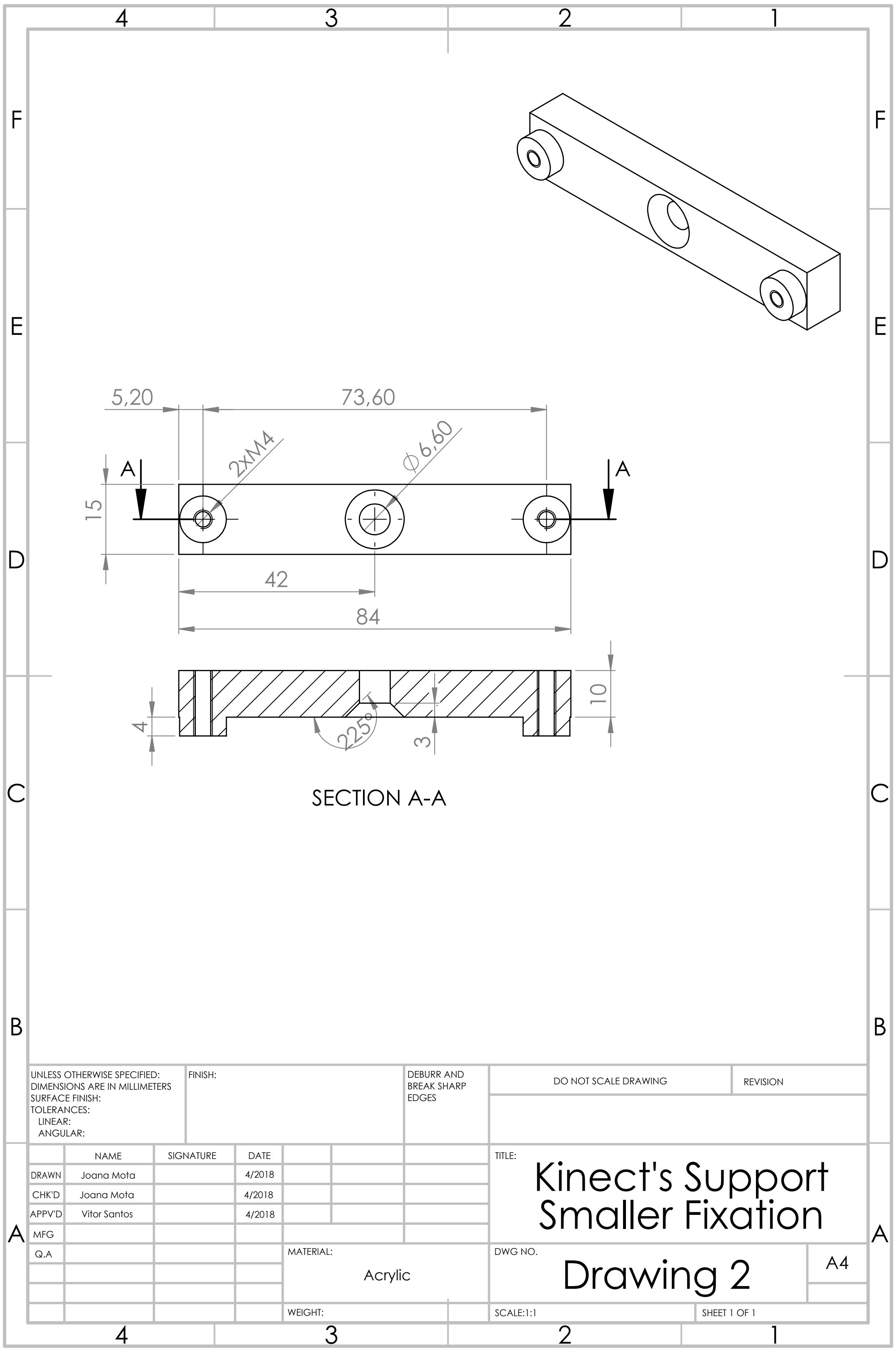
A4

WEIGHT:

SCALE:1:2

SHEET 1 OF 1

Figure A.1: Definition drawing of the base of the Kinect's support.



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND BREAK SHARP EDGES

DO NOT SCALE DRAWING

REVISION

	NAME	SIGNATURE	DATE
DRAWN	Joana Mota		4/2018
CHK'D	Joana Mota		4/2018
APPV'D	Vitor Santos		4/2018
MFG			
Q.A			

TITLE: Kinect's Support Smaller Fixation

MATERIAL: Acrylic

DWG NO. Drawing 2

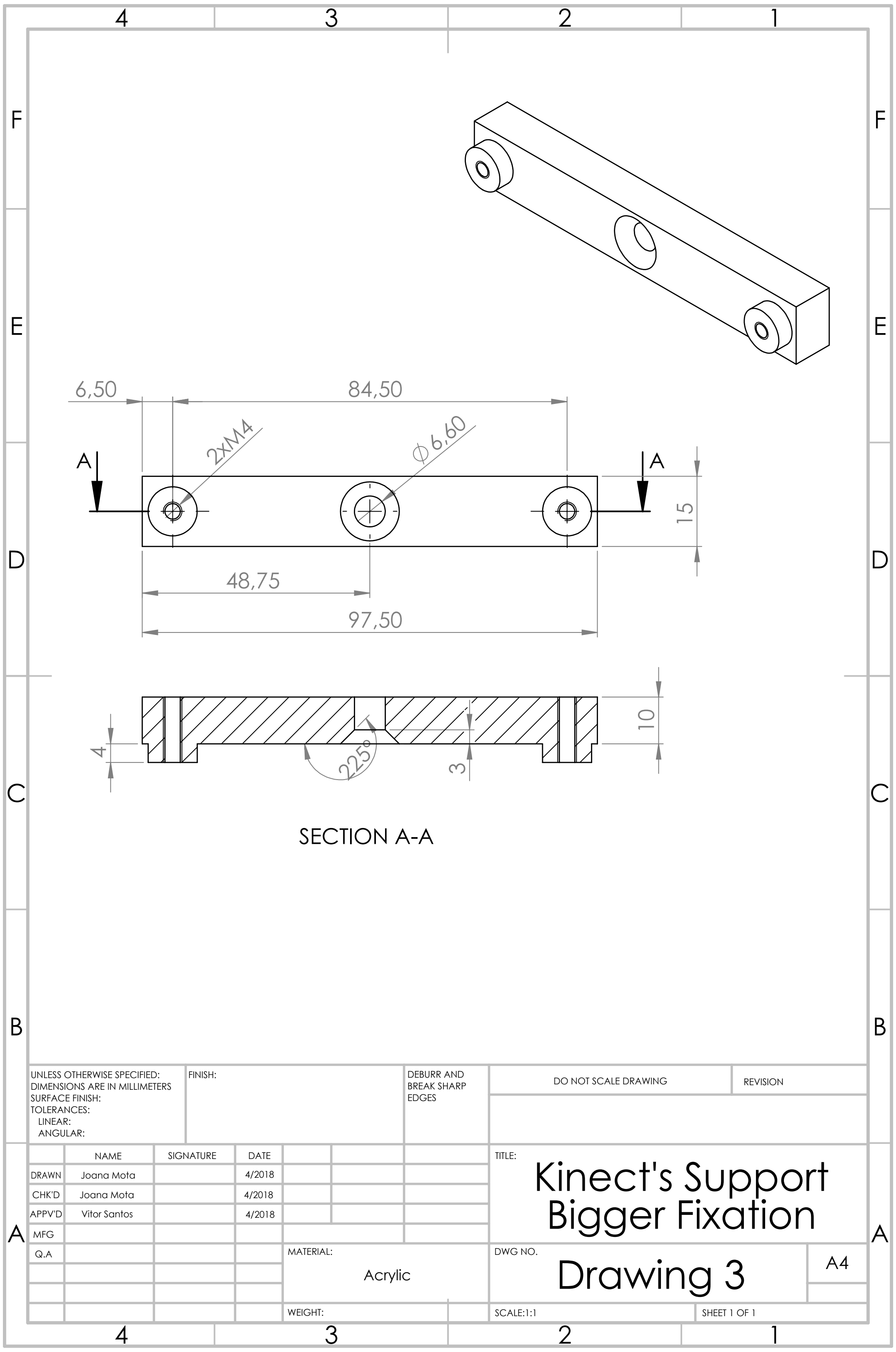
A4

WEIGHT:

SCALE:1:1

SHEET 1 OF 1

Figure A.2: Definition drawing of the smaller fixation part that fixates the support to the robot's arm.



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE: Kinect's Support Bigger Fixation	
DRAWN	Joana Mota		4/2018		
CHK'D	Joana Mota		4/2018		
APPV'D	Vitor Santos		4/2018		
MFG					
Q.A				MATERIAL: Acrylic	DWG NO. Drawing 3	A4
				WEIGHT:	SCALE:1:1	SHEET 1 OF 1

Figure A.3: Definition drawing of the bigger fixation part that fixates the support to the robot's arm.

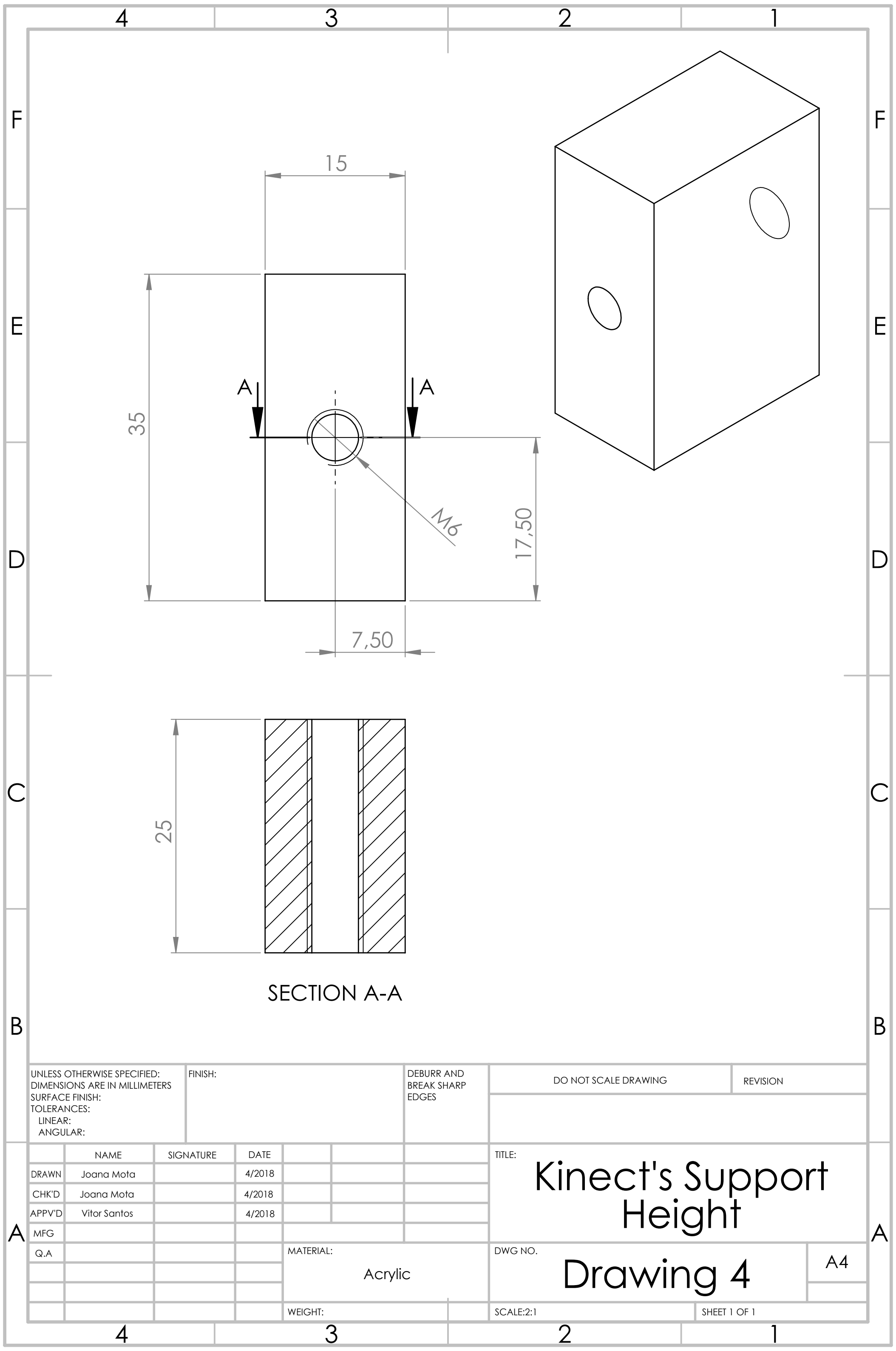


Figure A.4: Definition drawing of the part of the Kinect's support that gives it some height, thus providing the space needed for the motorized tilt of the Kinect.

Appendix B

Table with readings and the corresponding distance presented in the sensor's display.

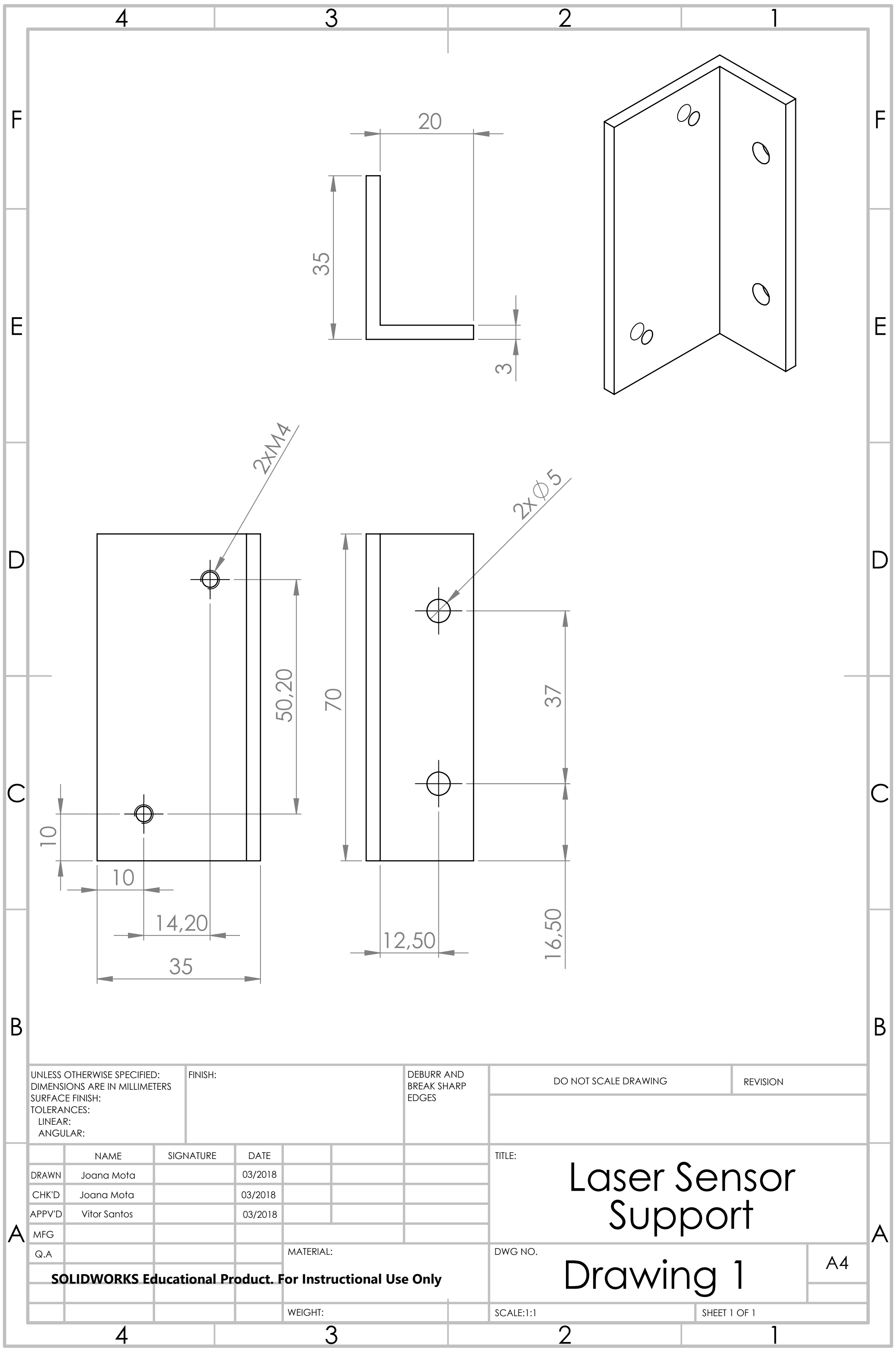
Sensor Distance (mm)	**ADC Reading (Arduino UNO)**
100	1003
110	984
120	963
130	943
140	923
150	903
160	883
170	863
180	843
190	823
200	803
210	783
220	763
230	743
240	723
250	703
260	682
270	663
280	641
290	622
300	601
310	582
320	562
330	542
340	522
350	502

360	481
370	461
380	441
390	421
400	399
410	380
420	360
430	340
440	320
450	299
460	279
470	259
480	239
490	219
500	198

Table B.1: Arduino Readings and the corresponding real distance.

Appendix C

Definition drawing of the laser sensor's support.



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND BREAK SHARP EDGES

DO NOT SCALE DRAWING

REVISION

	NAME	SIGNATURE	DATE
DRAWN	Joana Mota		03/2018
CHK'D	Joana Mota		03/2018
APPV'D	Vitor Santos		03/2018
MFG			
Q.A			

MATERIAL:

TITLE: Laser Sensor Support

DWG NO. Drawing 1

A4

SOLIDWORKS Educational Product. For Instructional Use Only

WEIGHT:

SCALE:1:1

SHEET 1 OF 1

Figure C.1: Definition drawing of the laser sensor support.

Appendix D

Execution of the developed bin-picking process.

Before executing the bin-picking process it is important to ensure that the initial setup, presented in section D.1, was correctly performed.

The `move_fanuc` is the main node which plans and controls all the manipulators movements. There are, in the *move* folder of the *bin_picking* package, 3 types of `move_fanuc` nodes. There is a `move_fanuc_debug` node which, as its name suggests, is used for debugging throughout the entire bin-picking process. This node was created because, in spite of always having RViz displaying all the robot's movement plans it is sometimes important to confirm the coordinates of the final position or even sometimes comprehend why some generated plans fail. The node `move_fanuc` should be the one used for trials when the manipulator is installed on a specific working table. Finally, the node `move_fanuc_demo` is the one used for executing the bin-picking process on the ROBONUC platform. In all of these programs, whenever the manipulator is prepared to move to the following position of the bin-picking process, it will wait for the user's permission to execute the process which should only been given after the confirmation, in RViz, of the generated plan.

In order to launch the planning environment and all the communications to control the manipulator the launch file `move_fanuc_bringup_all.launch` was created and is presented in section D.2.1. This will launch 3 important things for the process to work: 1. the planning and execution launch file (`moveit_planning_execution.launch`) which will be the one used to invoke the ROS Industrial tools, to launch the planning environment and all the communications to control the manipulator; 2. the Kinect drivers; 3. the node `vs_IO_client` which controls the activation of the robot's I/Os.

Besides launching the `move_fanuc_bringup_all.launch` it is also necessary to run one of the `move_fanuc` nodes, in another terminal window, to execute the bin-picking process.

D.1 Initial Setup

1. Turn on the compressed air for the activation of the suction gripper;
2. Turn on the robot's controller in AUTO mode, which is used to set the manipulator in continuous cycle;
3. Run the MONIO program in background logic, present in the Setup option of the console's Menu;
4. Run the ROS server and press the cycle start button;
5. Power the SICK laser sensor (if the manipulator is installed on the ROBONUC platform simply turn on the platform's circuit breaker and activate the Robuter II);
6. Power up the Kinect and connect it and the arduino UNO which interprets the laser sensor's readings to the computer;
7. Connect to the switch on the Robuter II which makes the Ethernet connection with the robot's controller and the arduino responsible for controlling the I/Os activation.

D.2 Launch Files

D.2.1 `move_fanuc_bringup_all.launch`

```
<?xml version="1.0"?>
<launch>
    <arg name="ip_str" value="192.168.0.231" />

    <!-- Planing and execution -->
    <group>
      <include file="$(find fanuc_moveit_config)/launch/moveit_planning_execution.launch">
        <arg name="sim" value="false" />
        <arg name="robot_ip" value="$(arg ip_str)" />
      </include>
    </group>

    <!-- Kinect Drivers -->
    <include file="$(find bin_picking)/launch/kinect.launch" />

    <!-- Node to control I/Os -->
    <node name="vs_IO_client" pkg="robonuc" type="vs_IO_client" />

</launch>
```

D.2.2 `kinect_bag.launch`

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

```xml
<?xml version="1.0"?>
<launch>
    <!-- Kinect Point Cloud Bag path -->
    <arg name="path" default="/home/joana/catkin_ws/src/Bin-picking/bin_picking
    /bagfiles/" />
    <arg name="bag" default="pieces_segmentation2" />
    <!-- Play Point Cloud Bag -->
    <node pkg="rosbag" type="play" name="player" output="screen" args="$(arg
    path)$(arg bag).bag -l" />

    <!-- start RViz configuration-->
    <node pkg="rviz" type="rviz" name="kinect_bag_rviz" args="-d $(find
    bin_picking)/config/kinect_bag_rviz.rviz"/>

    <!-- Kinect Drivers -->
    <include file="$(find openni_launch)/launch/openni.launch">
        <arg name="load_driver" value="false" />
    </include>

    <!-- Run objDetection node centroid and normal determination -->
    <node name="bin_picking_objDetection" pkg="bin_picking" type="objDetection"
     />
</launch>
```

D.2.3 `globalvisualization.launch`

Visualization of the virtual model of the robot with the vision system already calibrated. This is just a virtual representation of the robot and it is not used to represent the manipulators current state.

```xml
<?xml version="1.0"?>
<launch>
  <!-- Load the virtual model (URDF) of Robot + Vision System -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(
    find bin_picking)/urdf/binpicking.xacro'" />
  <param name="use_gui" value="true"/>

  <!-- robot_state : publishes current joint positions and robot state data -->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="
    joint_state_publisher" />
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="
    state_publisher" />

  <!-- start RViz configuration-->
  <node name="robot_rviz" pkg="rviz" type="rviz" required="true"  args="-d $(
    find bin_picking)/config/robot_rviz.rviz" />

</launch>
```

D.2.4 `global_state_visualize.launch`

Visualization of the virtual model of the robot with the vision system already calibrated in its current state. To use this launch file ensure the robot and the Kinect are both properly connected to the computer.

```xml
<?xml version="1.0"?>
<launch>

    <arg name="robot_ip" default="192.168.0.231" doc="IP of controller" />
    <arg name="J23_factor" default="1" doc="Compensation factor for joint 2-3
    coupling (-1, 0 or 1)" />
    <arg name="use_bswap" default="true" doc="If true, robot driver will byte-
    swap all incoming and outgoing data" />

    <!-- robot_state : publishes current joint positions and robot state data
    -->
    <rosparam command="load" file="$(find bin_picking)/config/joint_names.yaml"
     />

    <include file="$(find fanuc_driver)/launch/robot_state.launch">
        <arg name="robot_ip" value="$(arg robot_ip)" />
        <arg name="J23_factor" value="$(arg J23_factor)" />
        <arg name="use_bswap" value="$(arg use_bswap)" />
    </include>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="
    robot_state_publisher" />

    <!-- Load the virtual model (URDF) of Robot + Vision System -->
    <include file="$(find bin_picking)/launch/load_bin_picking.launch" />

    <!-- start RViz configuration-->
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find bin_picking)/
    config/robot_state_visualize.rviz" required="true" />

    <!-- Kinect Drivers -->
    <include file="$(find bin_picking)/launch/kinect.launch" />

</launch>
```