FEDERAL UNIVERSITY OF BAHIA

MASTER OF SCIENCE THESIS

OBJECT DETECTION, LOCALIZATION, AND GRASPING WITH VISUAL SENSORS APPLIED TO ROBOTIC MANIPULATORS

Daniel Moura de Oliveira

Graduate Program In Electrical Engineering

Salvador
2019

DANIEL MOURA DE OLIVEIRA

OBJECT DETECTION, LOCALIZATION, AND GRASPING WITH VISUAL SENSORS APPLIED TO ROBOTIC MANIPULATORS

This Master of Science Thesis was presented to the Graduate Program in Electrical Engineering of the Federal University of Bahia as a partial requirement for completion of the Master's Degree in Electrical Engineering.

Advisor: Prof. Dr. André Gustavo Scolari Conceição

Salvador
2019

Ficha catalográfica.

Moura de Oliveira, Daniel

OBJECT DETECTION, LOCALIZATION, AND GRASPING WITH VISUAL SENSORS APPLIED TO ROBOTIC MANIPULATORS/ Daniel Moura de Oliveira– Salvador, 2019.

107p.: il.

Advisor: Prof. Dr. André Gustavo Scolari Conceição.
Master Thesis– FEDERAL UNIVERSITY OF BAHIA, POLYTECHNIC SCHOOL, 2019.

1. Visão Computacional. 2. Grasping. 3. Pick and Place. 4. Robotic Manipulators. 5. PBVS..
I. Gustavo Scolari Conceição, André. II. FEDERAL UNIVERSITY OF BAHIA. POLYTECHNIC SCHOOL. III Título.

Daniel Moura de Oliveira

"Object Detection, Localization, and Grasping with Visual Sensors Applied to Robotic Manipulators"

Dissertação apresentada à Universidade Federal da Bahia, como parte das exigências do Programa de Pós-Graduação em Engenharia Elétrica, para a obtenção do título de *Mestre*.

APROVADA em: 10 de Setembro de 2019.

BANCA EXAMINADORA

Prof. Dr. André Gustavo Scolari Conceição
Orientador - UFBA

Prof. Dr. Tiago Trindade Ribeiro
UFBA

Prof. Dr. Eduardo Telmo Fonseca Santos
IFBA

ACKNOWLEDGEMENTS

First, i want to dedicate this work to to my mother, my father and my brother. I would like to thanks my advisor prof. Andre Gustavo Scolari Conceição for his support and patience in guiding me to write this document.

I would like to thank the people in the LaR laboratory, for helping solving and discussing some problems found a long the way. Lastly, i would like to thanks European Union's Horizon 2020 research and innovation programme for found this project.

RESUMO

Esta dissertação de mestrado tem como objetivo principal desenvolver um sistema de *pick and place* para braços robóticos com o uso de visão computacional. Serão utilizadas duas abordagens: *eye in hand*, onde a câmera fica fixa em uma posição próxima a garra do rôbo e *eye to hand*, onde a câmera fica fixa em uma posição próxima a base do robô. Para detecção de objetos serão implementados e avaliados dois algoritmos: por tags, utilizando *AprilTags*, e por features, utilizando *Oriented FAST and Rotated BRIEF*(ORB).

Para validação do sistema proposto serão utilizados os braços robóticos UR5 e JACO em conjunto de um sensor RGB-D. Resultados experimentais comparando os algoritmos de visão utilizado e realizando tarefas de *pick and place* serão feitos para demonstrar a eficiência do sistema.

Palavras-chave: UR5, JACO, AprilTag, ORB, Eye in Hand, Visão Computacional.

ABSTRACT

This Master's thesis aims to develop a pick and place system for robotic arms with the use of computer vision. Two approaches will be used: eye in hand, where the camera is fixed on the robotic manipulator and eye to hand, where the camera is near the robot base. For the detection of objects two algorithms will be implemented and evaluated: detection by tags, using AprilTags, and by features, using Oriented FAST and Rotated BRIEF(ORB).

For the validation of the proposed system, the UR5 and JACO robotic arms will be used in conjunction with an RGB-D sensor. Experimental results comparing both computer vision algorithm and pick and place tasks will be done to shown the efficiency of the system.

Keywords: UR5, JACO, AprilTag, ORB, Eye in Hand, Computer Vision.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Since the birth of the industry, big companies are always trying to find a way to increase their production and reduce costs. When the Industrial Revolution started, most processes were made by people in terrible working conditions, resulting in several deaths, leading to the creation of the first labor rights. With the technological advances and the birth of robots, most of the manual work could be replaced by machines over the years, reducing costs with labor, more humane working conditions, as repetitive and dangerous tasks can be done by robots, and increase in productivity, making automation very attractive for the industry. Robots not only take part in industrial tasks, they can work in environments that are not possible or undesirable by humans like space and undersea exploration, tasks in a radioactive environments and even defusing explosive devices.

Nowadays, thanks to the evolution of technology, the industry has two important concepts: internet of things(IoT) and industry 4.0. IoT is the concept of basically connecting any device to the Internet or to each other through other means. It is a concept not just growing in the industry, but in people daily life also, be it through smart phones or smart houses. While the concept of smart houses and devices isn't new, as seen in (COOK et al., 2003), where it is proposed an agent to control and automate a house, the emergence of small and powerful devices like Raspberry Pi and high-level communication protocols like Zigbee helped with the evolution of the concept of smart technology. In Han et al. (HAN; LIM, 2010), ZigBee is used for management of energy of a house while in Chen-Yen Peng and Rung-Chin Chen (PENG; CHEN, 2018) a Raspberry Pi is used to control house devices using voice commands. In the future, it is estimated that cities will become "smart" through IoT, as shown in Figure 1.1.

Industry 4.0 is the fourth revolution that has occurred in manufacturing. It uses IoT, Machine Learning, multiple devices and data to improve automation processes. It improves greatly the efficient and production of the industry, reducing the need for human intervention in the production process. One of the examples of great success in using Industry 4.0 is Amazon, where they had record sales and profit while having less workers (MERCHANT, 2019).

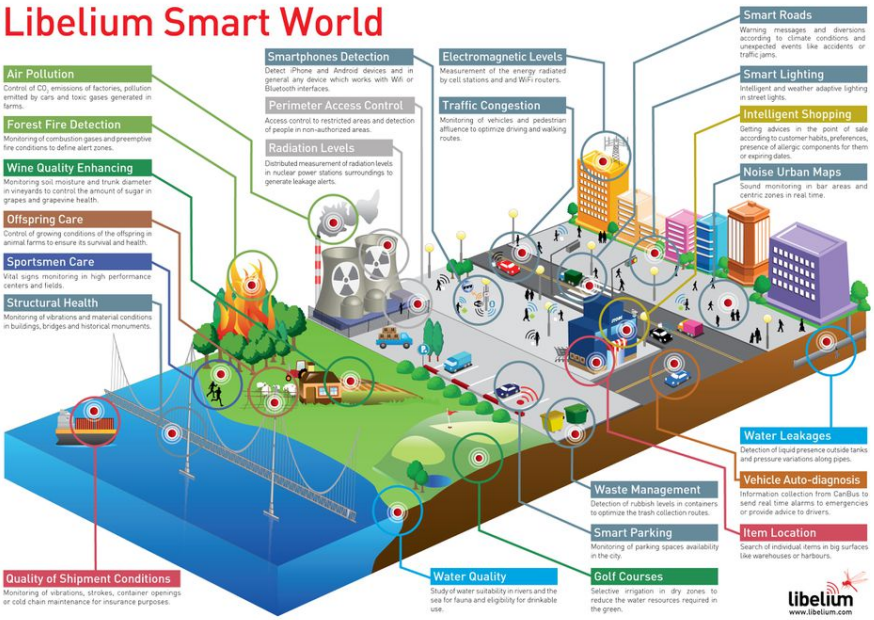


Figure 1.1: Smart World.
Source: Forbes, 2019.

When it comes to robots, that are diverse types of robots available in the market like mobile robots, robotic manipulators and humanoid robots. Those robots can be used in the industry in diverse tasks like carrying objects, use simple tools, reach difficult places and automation. The use of computer vision in robotics, called robotic vision, helped in the automation process greatly, since the robots can map, localize it self and find objects in the environment. The robotic manipulator is a robot that looks like an arm, as shown in Figure 1.2, and it is used mostly on pick and place tasks. Before automation, these kind of robots used to pick objects while being controlled by an operator or could be automated to do simple repetitive tasks, like pick an object in a specific position, but with the advance in technology and computer vision, these task can be completely autonomous and flexible.

With this in mind, this work will develop a autonomous pick and place system using computer vision. The objects will be detected using two approaches: Features and Tags. While some works, like (TEKE, 2018), use fully RGB-D image to detect and track, we are using the RGB image for object detection and the RGB-D image for object localization and orientation, where the set of position and orientation can be simply called *pose*, in space. This way, we reduce computational costs and improve the system performance.

The feature approach, using the Oriented FAST and Rotated BRIEF (ORB) algorithm, will require an image of the object to detect it, and with this image, extract features from the image of the object and the image from the visual sensor, compare those features and detect the object, as seen in Figure 1.3. The tag approach won't require previous acknowledgement of the object, but will require acknowledgement of the tag message, since each tag has an numerical id assigned to it. The tags used, AprilTag,

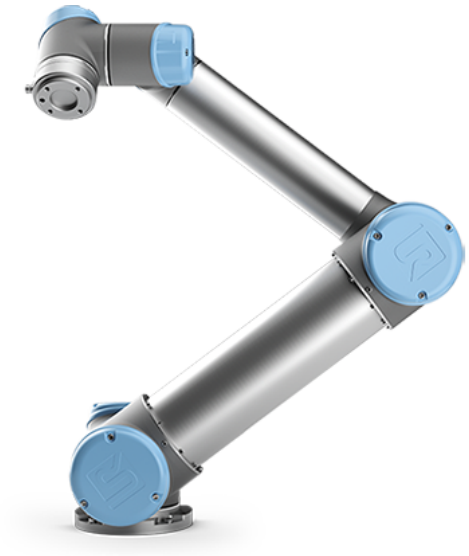


Figure 1.2: Manipulator Robot.
Source: Universal Robots, 2019.

can be seen in Figure 1.4.

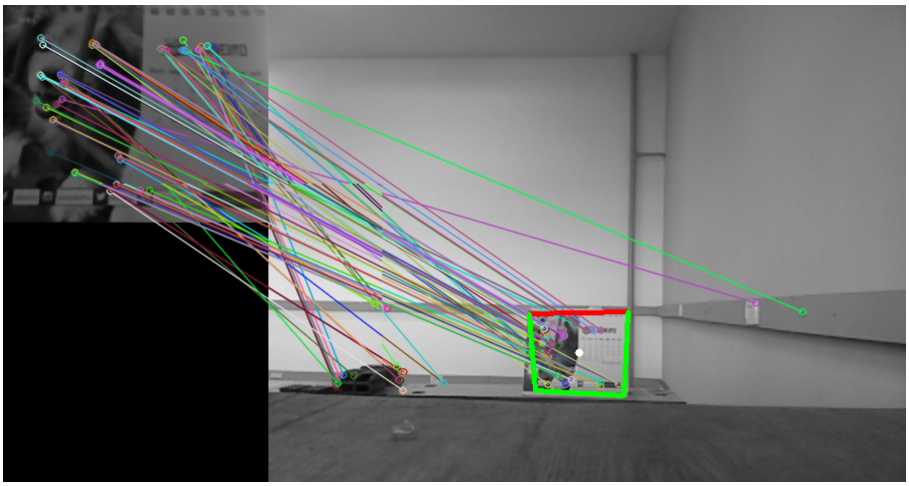


Figure 1.3: Object detection by features.

The manipulator robots used to execute the pick and place tasks were the UR5, from Universal Robots (ROBOTS, 2019b), JACO from Kinova Robots (KINOVA, 2019), the RGB-D sensor Intel Realsense D435 (INTEL, 2019), seen in Figure 1.5, and Microsoft Kinect (MOBINI; FOUMANI, 2013), seen in Figure 1.6. Pick and place tasks of different objects will be made to validate the system in different environments to show the flexibility of the system to grasp objects, like: on a 3D printer, table and inside a box. The control architecture adopted is the Position Based Visual Servoing (PBVS) (CORKE, 2017) using the eye in hand and eye to hand configuration. The visual pick system was delevoped using Robot Operating System (ROS) (QUIGLEY et al., 2009),

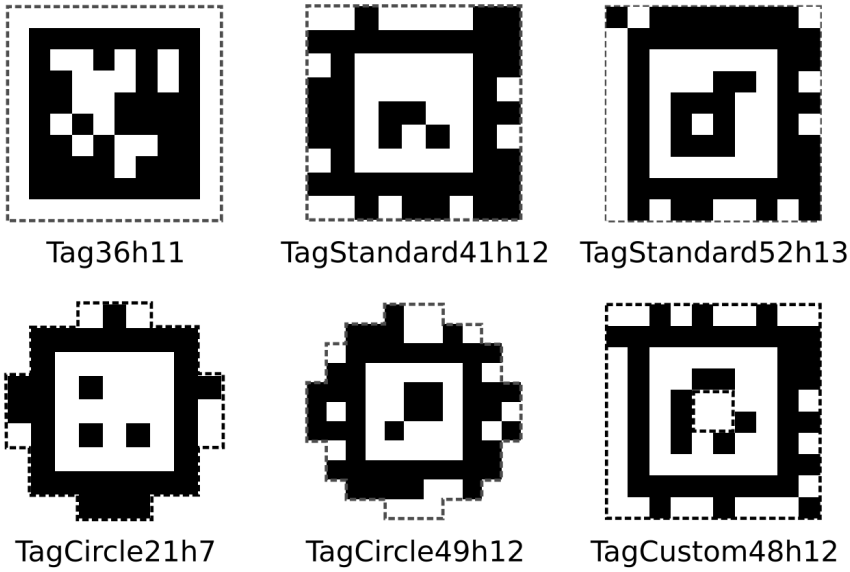


Figure 1.4: April Tags.

Figure 1.5: Intel Realsense D435.

since it is becoming a standard in robotic and make integration between robots easy and intuitive; Open Computer Vision (OpenCV) (BRADSKI, 2000), since it provides the necessary computer vision algorithms, and Eigen (GUENNEBAUD; JACOB et al., 2010) were used.

1.1 OBJECTIVES

The main objective of this research is the development of a control system for robot manipulators to execute pick and place tasks using computer vision, where two approaches will be used: on based on features and another in tags. The features approach allow any object to be detected as long as it has enough features and the tags approach allow an object to be detected, independent of its features or geometry.

Figure 1.6: Microsoft Kinect.

1.1.1 Specific Objectives

The specific objectives are as follows:

- Develop a pick and place system based on computer vision, using features and tags.
- Detect and find the object position using RGB sensor.
- Pose estimation based on the RGB estimation.
- Use Position Based Visual servoing control architecture.
- Evaluate the pick and place system in different environments.

1.2 JUSTIFICATION

This is a practical work, so the realization of this project implies in the use of know and existing algorithms, validating those algorithms in practical situations, inheriting the justificative of the original approaches. This way, expanding the literature to a practical approach of the problem.

This way, this work can be justified by expanding the literature, testing algorithms and techniques in a practical environment, creating a practical system that can be used in the industry, in the context of the Industry 4.0 and its use of robotic vision, since the costs of a visual sensor is cheaper, compared to others sensor, like lasers.

1.3 STRUCTURE OF THE THESIS

The thesis is structured as follows:

- Chapter 2 will shown the used vision algorithms and how they work;
- Chapter 3 will discuss the control system used;
- Chapter 4 shows the results and discussions related to them;
- The conclusion, final considerations and future works will be discussed on chapter 5;

Chapter 2

COMPUTER VISION

In this chapter, it will be discussed the computer vision algorithms used in this work, starting with related works on robotic vision on Section 2.1, the visual sensors used in this work on Section 2.2, the ORB algorithm on Section 2.3 and the AprilTag detection algorithm on Section 2.4.

2.1 RELATED WORK

Robotic vision is a common topic in literature nowadays, since a vision sensor is more accessible than a laser or others sensor, while the computers are getting smaller and more powerful, making the use of complex vision algorithms more viable than ever. Some examples of AprilTags applications on robotics can be seen in (WESTMAN; KAESS, 2018), where AprilTag is used to help improve a SLAM algorithm in a underwater environment and the paper (WANG et al., 2016) uses AprilTag to track and follow a ground vehicle using a unmanned aerial vehicle. Feature detection algorithms, like ORB, are versatile, low computational cost and can be applied in multiple applications like: localization in space using ORB SLAM (MUR-ARTAL; MONTIEL; TARDOS, 2015), detect objects in movement (XIE et al., 2013) and even object detection using a FPGA (KULKARNI; JAGTAP; HARPALE, 2013).

2.2 INTEL REALSENSE AND MICROSOFT KINECT

The vision sensors used in this work were the Intel Realsense D435 and Microsoft Kinect. Both are RGB-D sensors featuring RBG, RGB-D and point cloud image. The Kinect specifications can be seen below:

- Depth stream range: 4–11.5 ft (1.2–3.5 m)
- Viewing angle: 43 vertical by 57 degrees horizontal field of view
- Frame rate (depth and color stream): 30 frames per second (FPS)

- Depth Resolution: 320x240
- Color Resolution: 640x480

The intel Realsense D435 specifications can be seen below:

- Depth Field of View (FOV)—(Horizontal × Vertical) for HD 16:9: 85.2° x 58° (+/- 3°)
- Depth Stream Output Resolution: Up to 1280 x 720
- Depth Stream Output Frame Rate: Up to 90 fps
- Minimum Depth Distance (Min-Z): 0.11 m
- Maximum Range: Approximately 10 meters
- RGB Sensor Resolution and Frame Rate: 1920 x 1080 at 30 fps
- RGB Sensor FOV (Horizontal × Vertical): 69.4° x 42.5° (+/- 3°)

While both look similar, the Intel D435 has some better features, including important categories like resolution and minimum depth distance. On eye on hand applications, where the minimum depth distance is an important and limiting factor, since we can't create a huge distance between the camera and the end effector, the D435 works better. For this reason, the Intel visual sensor was on eye on hand applications and the Kinect on eye to hand applications.

2.3 ORIENTED FAST AND ROTATED BRIEF

The Oriented FAST and Rotated BRIEF (ORB) is a feature detection algorithm like SIFT (LOWE, 2004) and SURF (BAY; GOOL, 2006). While SIFT and SURF have a patent, ORB is under the same license as opencv (BRADSKI, 2000), the BSD License, so it is free to use in any kind of application. ORB, and others features detection algorithms, are ideal to detect common day to day objects like books, notebooks, calendars, etc. since most of those items have a lot of features.

ORB works by improving the FAST (ROSTEN; DRUMMOND, 2006) and BRIEF (CALONDER V. LEPETIT; FUA, 2010) algorithm, by adding a rotation component to FAST and a learning method for de-correlating BRIEF features under rotational invariance, leading to better performance in nearest-neighbor applications (RUBLEE et al., 2011).

2.3.1 FAST

FAST algorithm is a keypoint detector used in ORB. It is known for its performance, being faster than the classic algorithms, like SIFT, HARRIS and SUSAN (ROSTEN; DRUMMOND, 2006). It uses a machine learning algorithm, for corner detection, and

Non-maximal suppression, to remove corners which have an adjacent corner with a higher score function (ROSTEN; DRUMMOND, 2006). It was augmented with pyramid schemes for scale (KLEIN; MURRAY, 2008) and Harris corner filter (HARRIS; STEPHENS, 1988) to reject edges and provide a reasonable score (RUBLEE et al., 2011). Since FAST does not include a orientation operator, a modification was made to using *intensity centroid* (ROSIN, 1999).

2.3.2 oFAST

oFAST is the fast modification to include a orientation operator using *intensity centroid*. The *intensity centroid* assumes that a corner's intensity is an offset from its center and this vector may be used to impute an orientation. Consider *(p,q)* a pair of non-negative integers, *r=p+q* is called the order of the moment and I(x,y) the pixel intensities, the moments can be defined by:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{2.1}$$

Using this equation, the centroid can be defined with:

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{2.2}$$

The patch orientation from the corner's center to the centroid can be defined as:

$$\theta = atan2(m_{01}, m_{10}); \tag{2.3}$$

To improve the rotation invariance of this measure, it was made sure that the moments are computed with x and y remaining within a circular region of radius r. If C approaches 0, the measure may become unstable, but in the FAST algorithm that is rarely the case (RUBLEE et al., 2011).

2.3.3 BRIEF

The BRIEF is a binary feature descriptor. Binary descriptors are faster to compute and the similarity can easily be measured by the Hamming Distance algorithm (CALONDER V. LEPETIT; FUA, 2010). It is a bit string description of an image patch constructed from a set of binary intensity tests (RUBLEE et al., 2011). Consider a patch P, a binary test τ of size *SxS* can be defined as:

$$\tau(p;x;y) = \begin{cases} 1 & \text{if } p(x) < p(y), \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

where *p(x)* is the intensity of p at a point x. The feature is defined as a vector of n binary tests:

$$f_n(p) := \sum_{1 \le i \le n} 2^{i-1} \tau(p;x;y) \tag{2.5}$$

To increase its stability and repeatability, a Gaussian Smoothing Filter distribution was used around the center of the patch.

2.3.4 rBRIEF

Since BRIEF suffers performance issues with in-plane rotation, the rBRIEF descriptor was developed to solve this. The rBRIEF steers the BRIEF operator according to the keypoint's orientation (RUBLEE et al., 2011). For any feature set of n binary tests at location (x_i, y_i), define the 2 x n matrix:

$$S = \begin{bmatrix} x_1 & ,..., & x_n \\ y_1 & ,..., & y_n \end{bmatrix} \tag{2.6}$$

Rotating this matrix using the corresponding rotation matrix R_θ, we can construct S_θ:

$$S_\theta = R_\theta S. \tag{2.7}$$

Thus, we find the BRIEF operator as, considering $f_n(p)$ as Equation 2.5:

$$g_n(p, \theta) := f_n(p) | (x_i, y_i) \in S_\theta \tag{2.8}$$

To recover from the loss of variance in steered BRIEF, and to reduce correlation among the binary tests, it was developed a learning method for choosing a good subset of binary tests (RUBLEE et al., 2011). The learning algorithm consists of searching among all possible binary tests to find ones that both have high variance, as well as being uncorrelated.

2.3.5 RANSAC

Random Sample Consensus (RANSAC) (FISCHLER; BOLLES, 1981) is a mathematical model used for fitting and estimate parameters from a mathematical model. Different from others smoothing techniques, it uses a small initial set of data and enlarge this set when possible (FISCHLER; BOLLES, 1981). If the sample has enough compatible points, RANSAC would employ a smoothing technique, like least square, to improve the estimation of the parameters from the data. On Figure 2.1, we can see the RANSAC estimation compared to other models, like linear fit. We notice that since RANSAC is random, it ignored the data on the lower part and estimated a model based only on the more dense part of the graph.

It works by random sampling the observed data, and estimate a model based on it. If the estimated model error is small, accept as answer. If the error is higher than the tolerance, estimate another model. The algorithm ends when a model with few missing data is found or when no model is found with the predetermined tolerance. If the latter happens, it will use the model with the larger consensus found or terminate in failure.

In this thesis, RANSAC is used to find the homography between a sample image keypoints and descriptors and the camera image keypoints and descriptors. This process returns a set of four points and the geometry of the object, an exemple of this can be seen

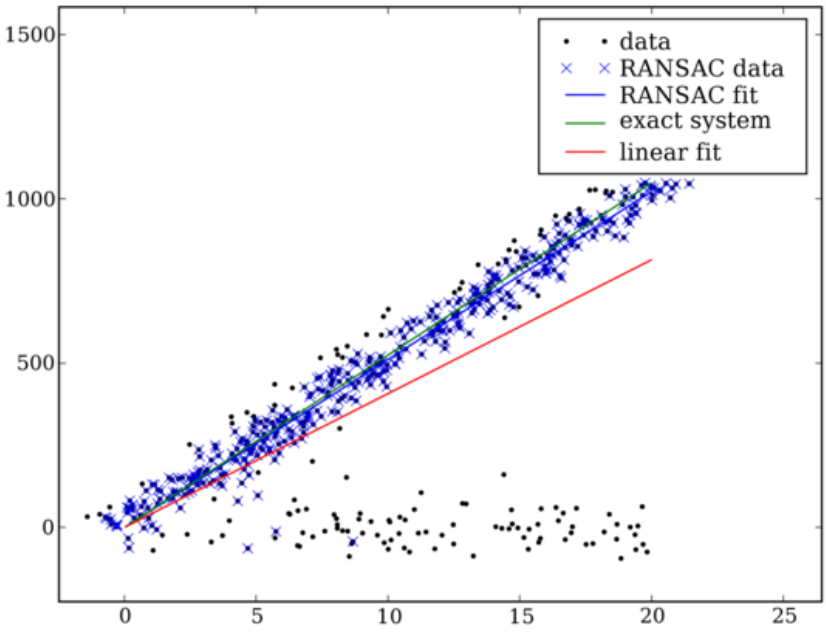


Figure 2.1: RANSAC execution.
Source: SciPy Cookbook

in Figure 1.3, where it compares an image of the object and the image of the camera. This geometry and set of points is necessary for the object localization.

2.4 APRILTAGS

AprilTags are a lexicode-based system for generating tags. It is an improved version of ARTags (FIALA, 2005), where it was empirically shown to reduce the false positive rate compared to ARTag designs of similar bit length (WANG; OLSON, 2016). The tags and its variations, called families, can be seen in Figure 2.2. In this work, AprilTags are used to detect objects with tags on it. Since those objects usually lack features, object detection algorithms that uses features, like ORB, have problems detecting objects. When it comes to grasping, use of an object with a tag gives two advantages when grasping: since we are going to aim at the tag, as seen in Figures 2.3 and 2.4, the object can have any kind of geometry and the tag has an ID, so if there multiple tags in a scene, it is possible to identify an specific object by its ID.

The detection process starts by turning the image into a black-and-white image. After this, an adaptive threshold is applied to find the minimum and maximum values in a region around each pixel. With this minimum and maximum values, each pixel is assigned a black or white color. The advantage of this approach is that the tags does not need to be black-and-white initially, so as long tags have a darker and a lighter color, it can be detected as seen in Figure 2.5. The problem of this approch we found was that

Figure 2.2: AprilTags families.

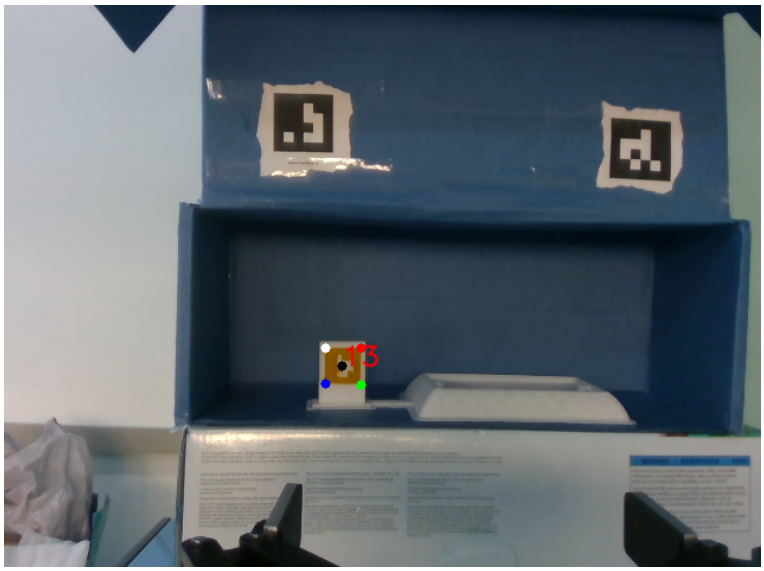


Figure 2.3: AprilTag and an object.

on ambient with lots of light, it may cause trouble detection the tag or even generating false detection. Reducing the camera brightness may help solve this problem.

Given the binarized image, the next step is to find edges which might form the boundary of a tag (WANG; OLSON, 2016). The solution found was to segment the edges based on the identities of the black and white components from which they arise, using the union-find algorithm (CORMEN C. E. LEISERSON; STEIN, 2009). For every pair of adjacent black and white components, the pixels of the boundaries are identified as a distinct cluster.

With those clusters, the next step is to fit a quad to each cluster of unordered boundary points, partitioning the points into four groups corresponding to line segments. First, the points are sorted by angle around their centroid. Corner points are identified by attempting to fit a line to windows of neighboring points, and finding the peaks in the mean squared error function as the window is swept across the points (WANG; OLSON, 2016). The last step is to select four corners which result in the smallest mean squared line fit errors. These quads outputs a set of candidates to be decoded.

To decode the tag, is used a simple XOR operation. The tag is identified as the code with the smallest Hamming distance from the detected code. To improve the detection, and avoid noises, the edges are refined by using the image gradient along the edges of the candidate quads to fit new edges. Since tags are always dark on the inside, points whose

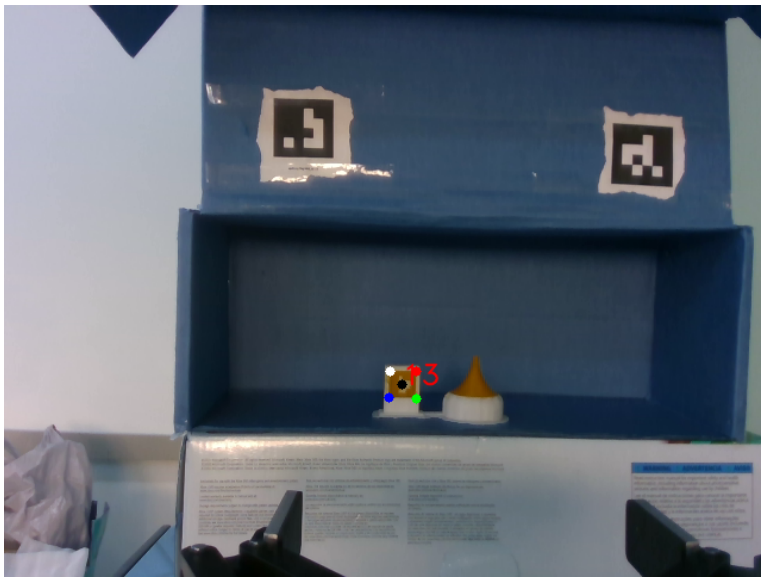


Figure 2.4: AprilTag and another object.

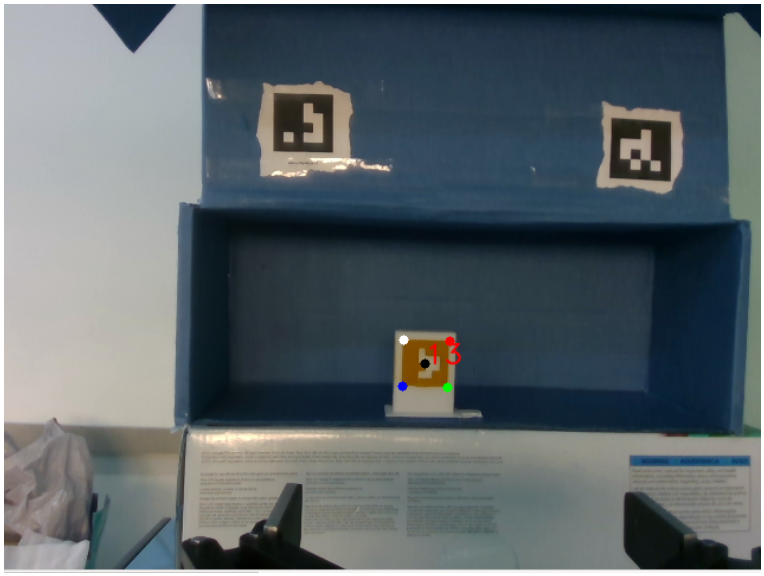


Figure 2.5: Yellow AprilTag from 3D printer.

gradient is not the expected sign are rejected.

As seem in John Wang and Edwin Olson (WANG; OLSON, 2016), it takes around 0.072 microseconds to compute each pixel, so it should take around 22ms for a 640 x 480 image. This computational time should be enough to use in most real time applications, like a navigation or real time localization.

Chapter 3

CONTROL SYSTEM

This chapter will discuss the control system used, the Position Based Visual Servoing. On Section 3.1, will be shown related work that uses similar types of control scheme, Section 3.2 will talk about the robots used, Section 3.3 will talk about Homogeneous Transformation, Section 3.4 will talk about the Forward and Inverse Kinematics, Section 3.5 will show the trajectory planning algorithm, Section 3.6 will show how to convert a pixel position to Cartesian Coordinates using point cloud and 3.7 will discuss the control system used.

3.1 RELATED WORK

Position Based Visual Servoing is a common control architecture in robotic, been used in multiple works through the years. The paper (WILSON; HULLS; BELL, 1996) shows its application on robotic and how this control system works. On the series of papers by François Chaumette and Seth Hutchinson (CHAUMETTE; HUTCHINSON, 2007a) (CHAUMETTE; HUTCHINSON, 2007b) they use computer vision data to control the motion of a robot while teaching the basics, discussing performance, stability and other problems. Position-based visual servoing robotic capture of moving target enhanced by Kalman filter (LAROUCHE; ZHU, 2015) uses Position Based Visual Servoing to capture a moving target using a Kalman Filter. While most works use this control scheme by itself, it is possible to mix it with other common visual control system, Image Based Visual Servoing, to generate better results, as seen in (KIM; OH, 2007).

3.2 ROBOTIC SYSTEM

The robot manipulator used in this work is the UR5, a robotic manipulator developed by Universal Robots, and the JACO by Kinova. According to Universal Robots (ROBOTS, 2019b), the UR5 features:

- Six Degrees of Freedom.

- Each joint has a working range of 360°and a maximum speed of 180°/Sec.
- Can carry up to 5 kg.
- Support to Robotiq Grippers.
- ROS driver.

Since the UR5 doesn't come with a gripper, the Robotiq 2-finger was used since it can be easily installed on the UR5. The gripper, as seen in Figure 3.1, features, as seem in (ROBOTIQ, 2019):

Figure 3.1: UR5 with Robotiq Gripper.
Source: Universal Robots.

- Plug and Play kit for Universal Robots.
- Adjustable stroke, speed and force.
- Detect if an object is picked.
- Two versions: 85 mm and 140mm. In this thesis, the 140mm version was used.
- Form-fit grip payload of 2.5 kg.
- Position resolution of 0.6mm.
- Speed of 30 to 250 mm/s.

Meanwhile the JACO, as seem in figure 3.2, features (KINOVA, 2019):

- Three fingers gripper.

Figure 3.2: JACO robotic arm.
Source: Kinova Robots.

- Can carry up to 5.2kg.
- 90cm reach.
- Maximum linear speed of 20 cm/s.
- Light rain resistant.

3.3 HOMOGENEOUS TRANSFORMATION

The Homogeneous Transformation is used to compute the projections and perspectives transformations of an object (BRIOT S.; KHALIL, 2015). A set of the basic homogeneous transformation for translation and rotation about the (x, y, z) axes respectively, is given by (SPONG; VIDYASAGAR, 2005):

$$Trans(x, a) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$Trans(y, b) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$Trans(z, c) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$Trans(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} = Trans(x,a)Trans(y,b)Trans(z,c) \tag{3.4}$$

$$Rot(x,\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_\alpha & -s_\alpha & 0 \\ 0 & s_\alpha & c_\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$Rot(y,\beta) = \begin{bmatrix} c_\beta & 0 & s_\beta & 0 \\ 0 & 0 & 0 & 0 \\ -s_\beta & 0 & c_\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$Rot(z,\gamma) = \begin{bmatrix} c_\gamma & -s_\gamma & 0 & 0 \\ s_\gamma & c_\gamma & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$Rot(\alpha,\beta,\gamma) = Rot(z,\gamma)Rot(y,\beta)Rot(x,\alpha) \tag{3.8}$$

With equations (3.4) and (3.8), we can define the Homogeneous Transformation Matrix **H** as:

$$H = Rot(\alpha,\beta,\gamma)Trans(a,b,c) \tag{3.9}$$

In this work, homogeneous transformation will be used mostly to solve the problem of converting the camera coordinates of an object to the arm coordinates, as seen in Figure 3.3, where we need to convert the position p, relative to the camera axis, to its position relative to the arm axis. To do this, consider the robotic manipulator origin as *o(x,y,z)*, the camera origin *o'(x',y',z')*, a point *p(x",y",z")* with orientation *θ'(α',β',γ')*, the position of the camera relative to the origin o as *c(xc,yc,zc)* and the camera orientation relative to the o axis as *θ(α,β,γ)*. To convert the position p, relative to the camera origin *o'*, to the robotic manipulator coordinate o, we can use the following homogeneous transformation:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = Rot(\alpha,\beta,\gamma)Trans(xc,yc,zc)Trans(x'',y'',z'')Rot(\alpha',\beta',\gamma') \tag{3.10}$$

3.4 KINEMATICS

Kinematics is used to describe the motion of the robotic manipulator without consideration of the forces and torques causing the motion (SPONG; VIDYASAGAR, 2005). The concept of kinematics can be divided into two: Forward and Inverse Kinematics,

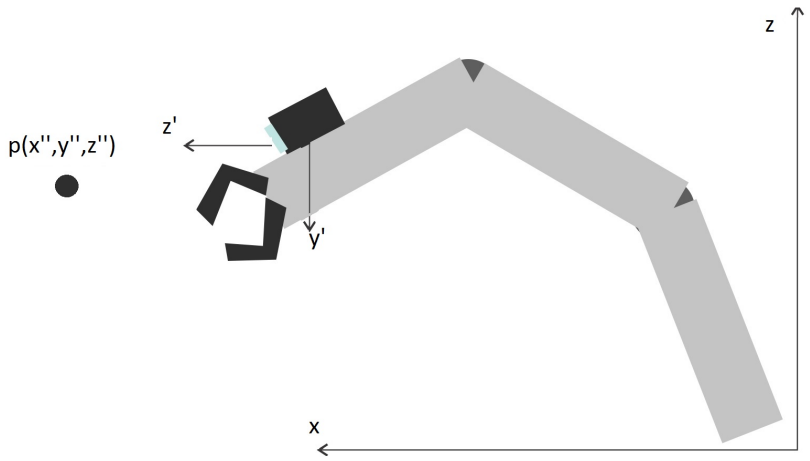


Figure 3.3: Eye on hand set up.

where the former is used to determine the position and orientation of the manipulator end-effector using the joints positions, and the latter is to determine the joints positions given the end-effector pose. While solving the Forward Kinematics can be a trivial task, the Inverse Kinematics can become a complex task on robotic manipulators with four or more degrees of freedom.

To solve Forward and Inverse Kinematics for the UR5 in this thesis, the equations found on Kinematics of a UR5 (ANDERSEN, 2018) were used. The JACO on ROS already came with its Forward and Inverse Kinematics solved by the manufacturer. These equations were deduced based on its Denavit-Hartenberg parameters and geometry.

3.4.1 Forward Kinematics

To get the Forward Kinematics, it is used the Denavit-Hartenberg (DH) convention. In this convention, each homogeneous transformation A_i is represented as a product of four basic transformations (SPONG; VIDYASAGAR, 2005):

$$A_i = Rot(z,\theta_i)Trans(z,d_i)Trans(x,a_i)Rot(x,\alpha_i) = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Where a_i, α_i, d_i, θ_i are the link length, link twist, link offset, and joint angle, respectively. The reference used to define those variables can be seen in Figure 3.4. To be able to define those values, first each joint must be assigned with coordinates frames. The UR5 DH parameters can be seen in the Table 3.1, as seen in the manufacturer's website (ROBOTS, 2019a).

Using the DH parameters, the general transformation between links *i - 1* and *i* are given by the matrix:

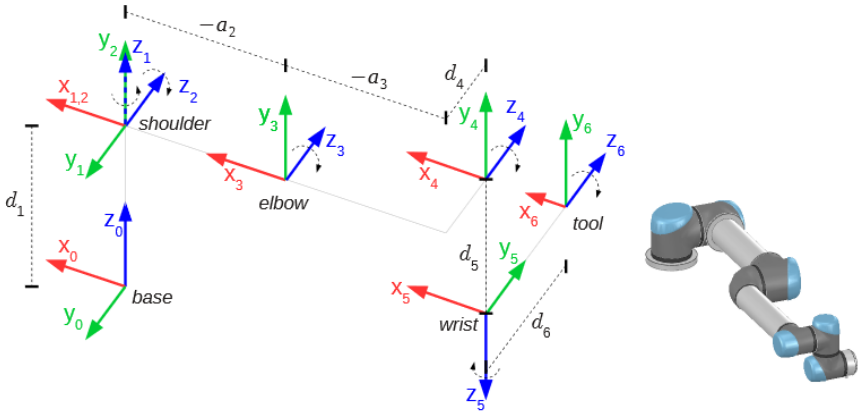


Figure 3.4: Joit Axis Reference.
Source: (ANDERSEN, 2018)

Kinematics	theta [rad]	a[m]	d[m]	alpha [rad]
Joint1	0	0	0.1625	$\pi/2$
Joint2	0	-0.425	0	0
Joint3	0	-0.3922	0	0
Joint4	0	0	0.1333	$\pi/2$
Joint5	0	0	0.0997	$-\pi/2$
Joint6	0	0	0.0996	0

Table 3.1: UR5 DH parameters.

$$T_i^{i-1} = \begin{bmatrix} cos_{\theta_i} & -sin_{\theta_i} & 0 & a_{i-1} \\ sin_{\theta_i}cos_{\alpha_{i-1}} & cos_{\theta_i}cos_{\alpha_{i-1}} & -sin_{\alpha_{i-1}} & -sin_{\alpha_{i-1}}d_i \\ sin_{\theta_i}sin_{\alpha_{i-1}} & cos_{\theta_i}sin_{\alpha_{i-1}} & cos_{\alpha_{i-1}} & cos_{\alpha_{i-1}}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

3.4.2 Inverse Kinematics

The Inverse Kinematics is used to calculate the joint angles θ_{1-6} based on the desired position and orientation of the arm end effector, specified as the transformation T_6^0. The method used in the Kinematics of a UR5 (ANDERSEN, 2018) was the geometric approach, where the joints angle are calculated based on the geometry of the arm and its DH parameters. When using the geometric approach, first we need to consider the type of joints of the robotic manipulator. On the UR5 we have six rotary joints, so, to find the joint angles θ_{1-6}, we need to apply geometry and trigonometry equations on each joint. For example, on Figure 3.5, we have a single rotary joint, so the inverse kinematics can be trivial to find. Using trigonometry, we can find the joint angle to move the arm to the position (x_c, y_c) as:

$$\theta_1 = atan2(x_c, y_c) \tag{3.13}$$

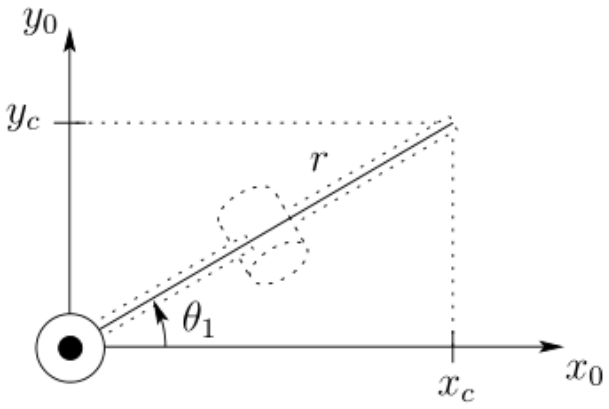


Figure 3.5: Rotary Joint example.
Source: Robot Modeling and Control (SPONG; VIDYASAGAR, 2005).

While the inverse kinematics can be trivial on simple robots, finding it on robots with more than three joints is no trivial task, so, in Thesis, we used the equations found in Kinematics of a UR5 (ANDERSEN, 2018), where they were able to successfully derive the joint equations through geometry. Considering the notation P_6^0 as he origin of frame 6 seen from frame 0 and Y_6^0 as a unit vector giving the direction of y-axis of frame 6 seen from frame 0, the joint angles necessary to make the arm go to a position (x, y, z) with orientation (ox, oy, oz) can be defined as:

$$\theta_1 = atan2(P_{5y}^0, P_{5x}^0) \pm acos(\frac{d4}{\sqrt{(P_{5x}^0)^2 + (P_{5y}^0)^2}}) \tag{3.14}$$

$$\theta_5 = \pm acos(\frac{P_{6x}^0 sin\theta_1 - P_{6y}^0 cos\theta_1 - d4}{d6}) \tag{3.15}$$

$$\theta_6 = atan2(\frac{-X_{0y}^6 sin\theta_1 + Y_{0y}^6 cos\theta_1}{sin\theta_5}, \frac{X_{0y}^6 sin\theta_1 - Y_{0y}^6 cos\theta_1}{sin\theta_5}) \tag{3.16}$$

$$\theta_3 = \pm acos(\frac{|P_{4xz}^1|^2 - a_2^2 - a_3^2}{2a_2a_3}) \tag{3.17}$$

$$\theta_2 = atan2(-P_{4x}^1 - P_{4x}^1) - asin(\frac{-a_3 sin\theta_3}{|P_{4xz}^1|}) \tag{3.18}$$

$$\theta_4 = atan2(X_{4y}^3, X_{4x}^3) \tag{3.19}$$

Where the two solutions in Equation 3.14 correspond to the shoulder being “left” or “right”, in Equation 3.15 correspond to the wrist being “up” or “down” and in Equation 3.17 correspond to “elbow up” and “elbow down”.

3.5 TRAJECTORY PLANNING

To plan each joint trajectory, Quintic Polynomial Trajectories was used. This trajectory planner is used to ensure a smooth trajectory between two points. It uses variable velocity and acceleration, making sure that the joints will move faster when far away from the destination and slow down when getting closer. Different from the Cubic Polynomial Trajectories, Quintic Polynomial Trajectories does not generate a jerk on the joints, thanks to the variable acceleration (SPONG; VIDYASAGAR, 2005).

We consider a quintic trajectory as:

$$q(t) = a_0 + a_1 t + a_3 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{3.20}$$

Where, q is the position and a is a constant value. By deriving q, we can obtain the velocity v and acceleration α as:

$$q_0 = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 \tag{3.21}$$

$$v_0 = a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 \tag{3.22}$$

$$\alpha_0 = 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 \tag{3.23}$$

$$q_f = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 \tag{3.24}$$

$$v_f = a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 \tag{3.25}$$

$$\alpha_f = 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3 \tag{3.26}$$

Which can be written as:

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ \alpha_0 \\ q_f \\ v_f \\ \alpha_f \end{bmatrix} \tag{3.27}$$

With equation 3.27, we can define the constant values and find the position, velocity and acceleration over the trajectory. Since the UR5 and the JACO has six degrees of freedom, the equation has to be applied to each joint individually.

3.6 3D LOCALIZATION

After detecting the object and finding its 2D position on the image plane, it is necessary to transform its 2D position to XYZ coordinates. To do this, the most common method is using the pinhole camera model, as seen in (FORSYTH; PONCE, 2003), where the camera needs to be calibrated to find the camera intrinsic parameters. Based on this parameters, it is possible to correlate a pixel to a 3D point in space. While this method is used on RGB camera, on a RGB-D camera it is possible to make a 2D to 3D conversion by using the depth sensor. This approach is more efficient, since it removes the

need to calibration and used the precision of the depth sensor. Since ROS is used in this research, its point cloud format was used to avoid the need for external dependencies like Point Cloud Library(PCL). By following the ROS point cloud documentation (POINTCLOUD2. . . , 2019), we can get the Algorithm 1 to convert a pixel to XYZ on ROS point cloud format, where *(u,v)* are the pixel coordinates, *pCloud* the point cloud data array and *(X,Y,Z)* the final 3D position.

Algorithm 1 Pixel to 3D

Require: $u > 0$ and $v > 0$

$arrayPosition \leftarrow v * pCloud.row\_step + u * pCloud.point\_step$
$arrayPosX \leftarrow arrayPosition + pCloud.fields[0].offset$
$arrayPosY \leftarrow arrayPosition + pCloud.fields[1].offset$
$arrayPosZ \leftarrow arrayPosition + pCloud.fields[2].offset$
$X \leftarrow pCloud.data[arrayPosX]$
$Y \leftarrow pCloud.data[arrayPosY]$
$Z \leftarrow pCloud.data[arrayPosZ]$

While this approach returns a precise measurement of the object position, since the point cloud is made by the camera infra-red, it suffers some problems like interference from the environment and the object material, which may result in some wrong measurements. Another weak point when compared to classic pinhole model is the need for more memory from the computer, since the point cloud is a three-dimensional array.

3.7 POSITION-BASED VISUAL SERVOING

Position-Based Visual Servoing (PBVS) is a visual servoing control method, where the system will depend of the position in space of a target. Different from image-based visual servo (IBVS), where the control is based solely on the image plane, on PBVS it is needed the target pose in Cartesian space with respect to the camera. Since in this work we are going to work with a robotic manipulator, that only receives commands in Cartesian space, we are going to need a control system that uses Cartesian positions, making the PBVS ideal for this application. To apply the PBVS, firstly we need to estimate the pose of the of the target and, to do so, we need an algorithm to detect the object. After its detection, we find its position in 2D space and convert its position to 3D. With the 3D pose, we can estimate the required motion to the target pose. In Figure 3.6, we can see a basic PBVS control system. The main weak point of this system is the need of an accurate pose estimation of the target, so a precise calibration of intrinsic and extrinsic parameters is needed. Since we are using point cloud to estimate the 3D pose, the main problem becomes the estimation of the extrinsic parameters. The control system used in this work can be seem in Figure 3.7, where it will receive a desired target, by the ID of the tag or by a reference image, and will send a joint trajectory to the manipulator after the execution of the relevant algorithms.

This system will run on each ROS loop, where each loop has 0.1 seconds, and the end condition will depend of the application.

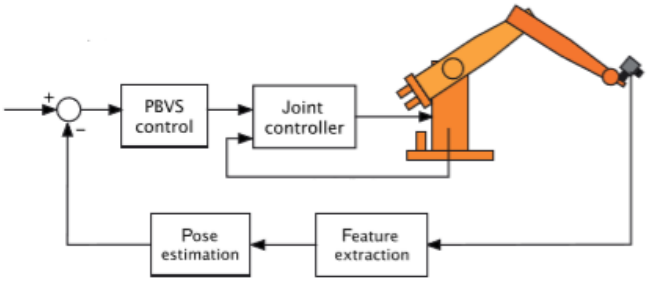


Figure 3.6: PBVS control system example.
Source: Robotics, Vision and Control (CORKE, 2017)

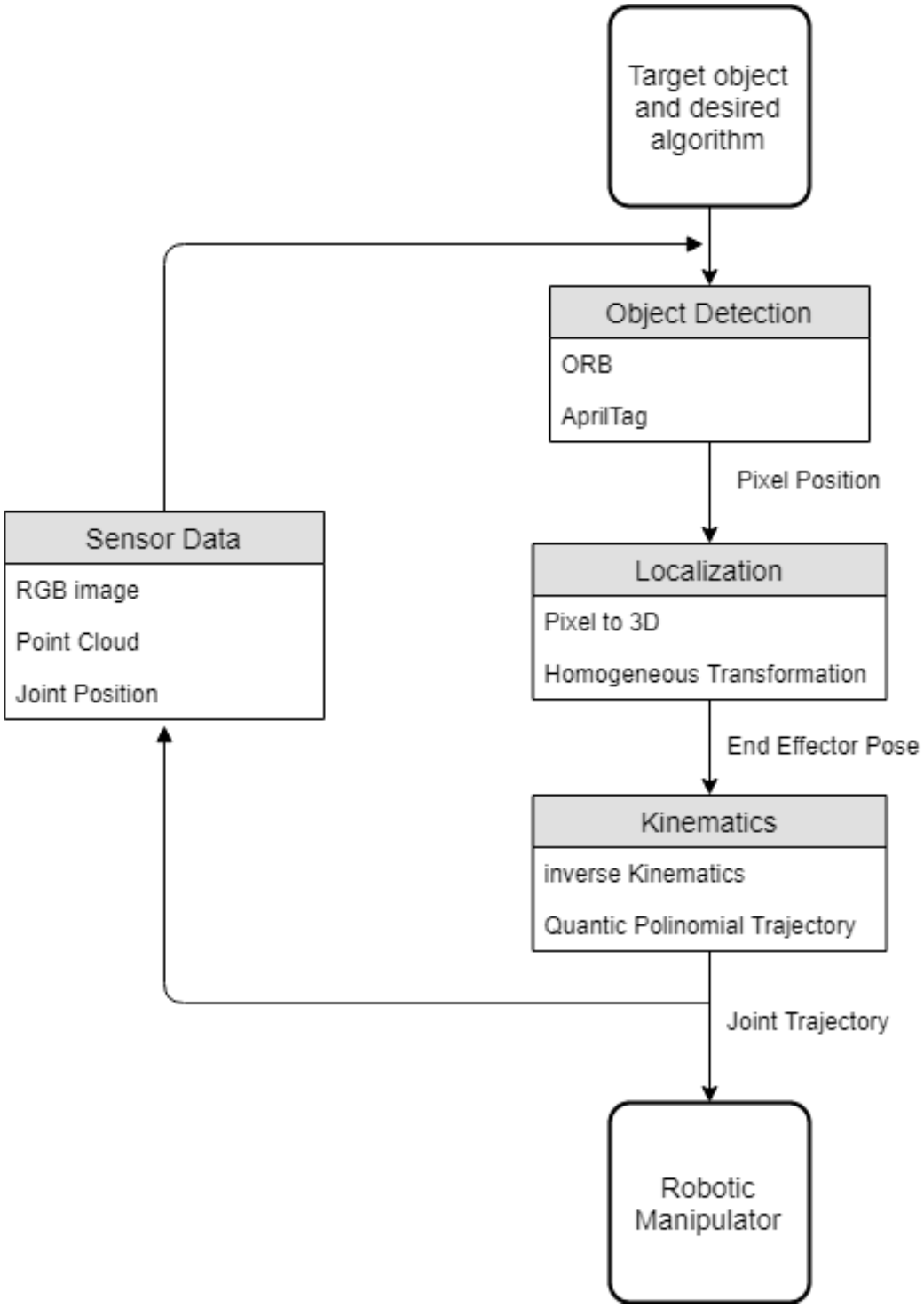


Figure 3.7: PBVS control system.

Chapter 4

RESULTS

In this chapter we are going to discuss the results of this work. The results will be separated in two sections: Detection and Pose Estimation, where visual data from both algorithms are going to be compared, and Pick and Place System, where each algorithm are going to have multiples experiments. All the algorithms discussed were implemented on Ubuntu 16.04 using ROS Kinetic, C++14, OpenCV3 and Eigen3.

4.1 DETECTION AND POSE ESTIMATION

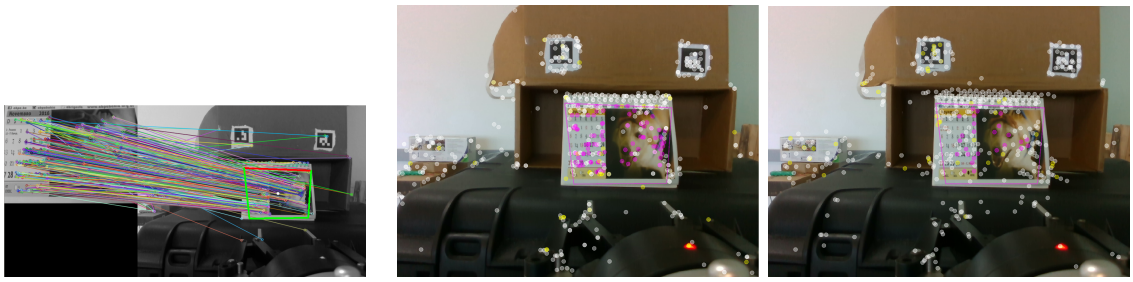
In this section, we are going to compare firstly ORB to SURF and SIFT to detect objects and then AprilTags and ORB, where the object will be placed in two positions from the camera. We are going to compare visual data, like pose estimation, pose variation on each frame and reliability from the algorithms while using the visual sensor Intel D435.

4.1.1 Visual Data

4.1.1.1 ORB vs SIFT vs SURF In this experiment, we are going to compare the detection of an object while using ORB, SIFT and SURF. When it comes to detection, all three algorithms can detect objects rich in features, as seen in Figures 4.1a, 4.1b and 4.1c.

Comparing the estimated position related to the camera, as seen in Figures 4.2, 4.3 and 4.4, we notice that they have similar behaviour when it comes to detection, with a small variation in millimeters scale between each frame. On Table 4.1, we can see the mean and standard deviation of the object position, where, while the mean seems similar, SIFT returns a better standard deviation and SURF the worse deviation.

Doing the same experiment, but with the object rotated around 90 degrees, it is noticeable that some with the algorithms starts to lose some quality of the estimation, as seen from the Figures 4.5, 4.6 and 4.7. On Table 4.2 we notice that while ORB and SIFT were able to maintain some consistence in the detection, SUFT detection deviation got worse comparing to the last experiment.



(a) ORB detection. (b) SIFT detection. (c) SURF detection.

Figure 4.1: Object detection from the algorithms.

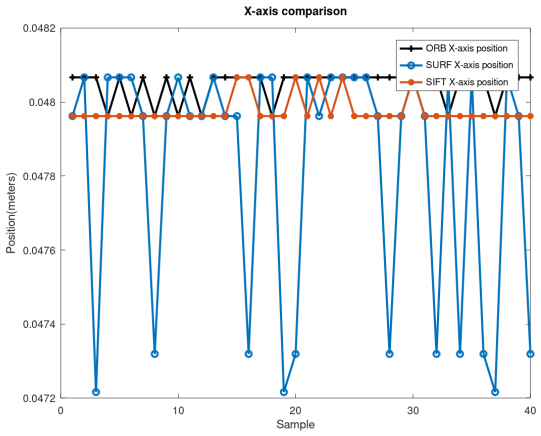


Figure 4.2: Object X position.

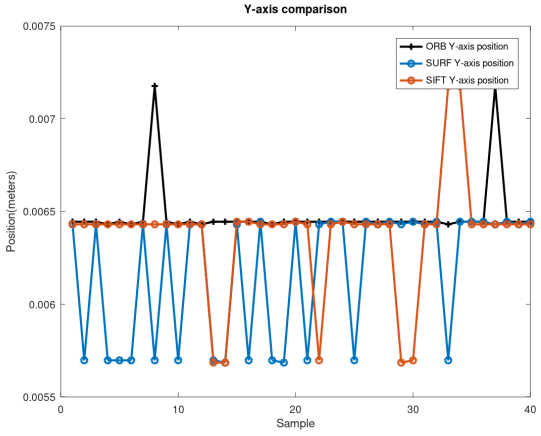


Figure 4.3: Object Y position.

When it comes to execution time, ORB took around 0.032 seconds do detect, while SURF took 0.045 seconds and SIFT 0.062 seconds. While they can detect a object rich on features, SIFT and SURF had trouble detecting objects with a small number of features, like an AprilTag. ORB could detect the tag since it generates a greater amount of features

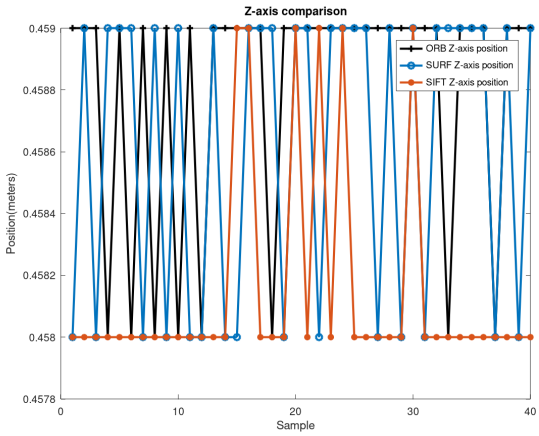


Figure 4.4: Object Z position.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	0.048067	0.0064450	0.45900	0.00013919	0.00022843	0.00044278
SURF	0.047962	0.0064310	0.45900	0.00039818	0.00037014	0.00049694
SIFT	0.047962	0.0064310	0.45800	0.000037466	0.00030601	0.00035784

Table 4.1: Comparison between mean and standard deviation from the algorithms on the object position.

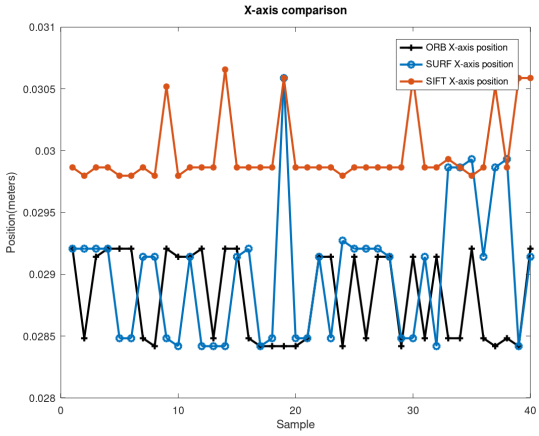


Figure 4.5: Object X position while rotated.

when compared to other feature algorithms (TAREEN; SALEEM, 2018).

4.1.1.2 ORB vs AprilTag: Distance = 0.36 meters For this experiment, the target was at 0.36 meters away from the visual sensor. On Figures 4.8 and 4.9, it is noticeable that in both cases the object is detectable with some differences: while the ORB detects the entire object, giving a better notion of its dimensions, the AprilTag can only detect the tag on it. Also, since we are detect the object center, both are going

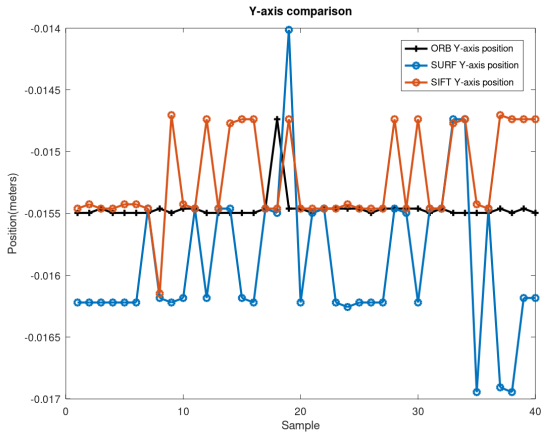


Figure 4.6: Object Y position while rotated.

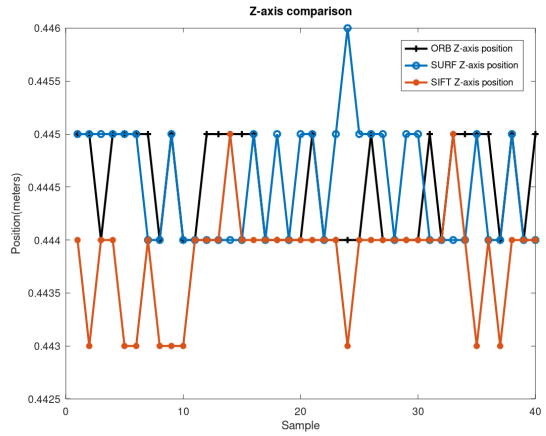


Figure 4.7: Object Z position while rotated.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	0.028483	-0.015460	0.44400	0.00037457	0.00012874	0.00052557
SURF	0.029142	-0.016183	0.44400	0.00057289	0.00060230	0.00056494
SIFT	0.029865	-0.015460	0.44400	0.00026553	0.00036342	0.00048643

Table 4.2: Comparison between mean and standard deviation from the algorithms on the object position.

to return a slightly different position. One of the biggest disadvantage of the ORB is that, while it can detect rotated objects, it doesn't give a 3D pose, but it is possible to get its orientation relative to the *Z-axis* by applying basic geometry on the red top line (CONCEICAO; OLIVEIRA; CARVALHO, 2018). To do this, consider the point *po=(xo,yo)* and the point *pf= (xf,yf)*, defined by the red line on top of the object, we can define the orientation of the object as:

$$\Theta = atan2(yf - yo, xf - xo) \tag{4.1}$$

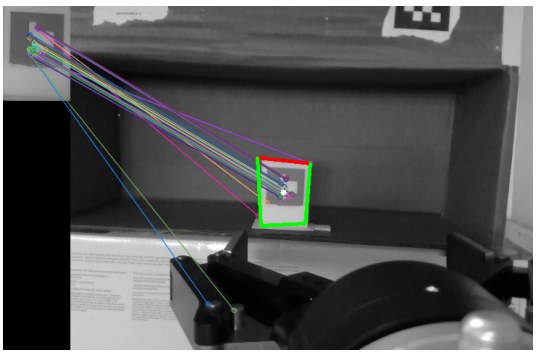


Figure 4.8: Experiment 1.1.1: ORB detection.

Figure 4.9: Experiment 1.1.1: Tag detection.

Comparing the sensor data, seen in Figures 4.10, 4.11 and 4.12, we notice that while both return a similar position, with a small different thanks to the difference in the center position, the ORB has a bigger position variation. Since the tag isn't a rich feature object, any mismatches with features in the ambient may cause position variations on its detection.

Analyzing Table 4.3 we can see that while the mean of the position seems close, the standard deviation (Std Dev) of the ORB algorithm is bigger than the AprilTag, showing the instability of the detection.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	-0.024378	0.015023	0.35900	0.0016830	0.0018604	0.00063669
AprilTag	-0.024243	0.0079201	0.35700	0.00026038	0.000010305	0.00046460

Table 4.3: Experiment 1.1.1: Comparison between mean and standard deviation from both algorithms on its position.

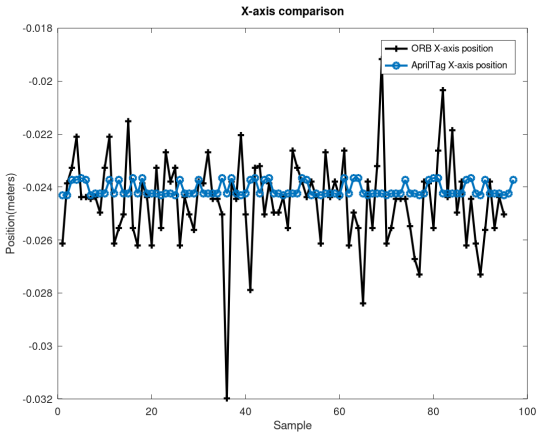


Figure 4.10: Experiment 1.1.1: ORB vs AprilTag x-axis position.

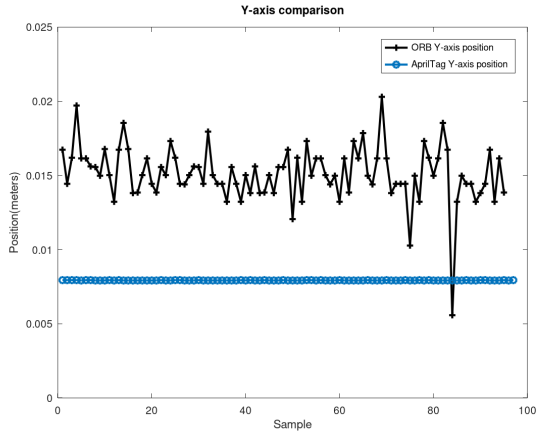


Figure 4.11: Experiment 1.1.1: ORB vs AprilTag y-axis position.

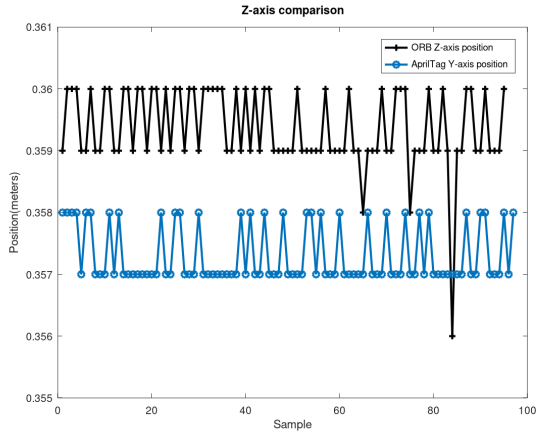


Figure 4.12: Experiment 1.1.1: ORB vs AprilTag z-axis position.

Comparing the orientation, seen in Figures 4.13, 4.14 and 4.15, we see that the ORB doesn't return a orientation for the X and Y axis, while return for the *Z-axis*. We notice a variation of a couple degrees on the AprilTag detection, and the angles on the **X** and **Y** axis aren't zero. That happens since the camera may no be perfectly aligned and the object is on a uneven surface.

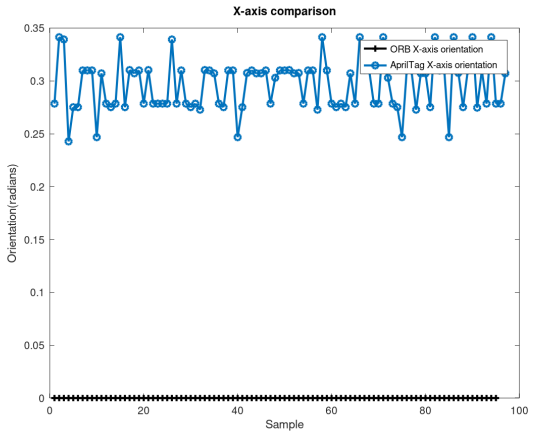


Figure 4.13: Experiment 1.1.1: ORB vs AprilTag x-axis orientation.

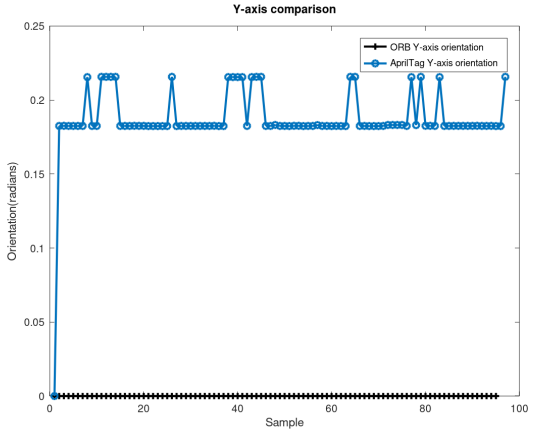


Figure 4.14: Experiment 1.1.1: ORB vs AprilTag y-axis orientation.

On table 4.4 we see a similar result as last table, that while the mean may be similar on the *Z-Axis*, the Std Dev from ORB is higher than the from AprilTag.

The second part of this experiment, the object was rotated around 95 degrees on the *Z-Axis* while keeping a similar distance to the camera, as seen in Figure 4.16, where we can also see that the ORB can keep the detection of the object even with a 90 degree rotation in relation to the reference image.

Comparing the sensor data relative to position, seen in Figures 4.17, 4.18 and 4.19, we see similar results with a sightly offset on the position and a small variation on ORB

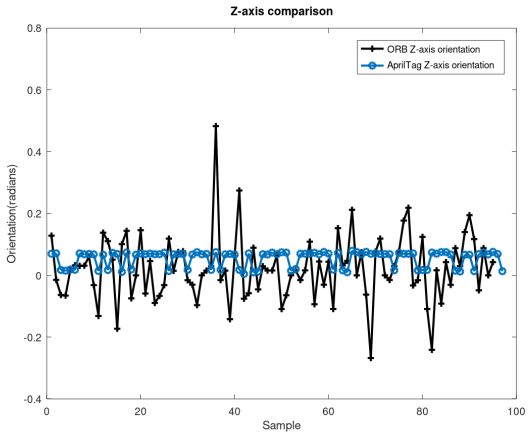


Figure 4.15: Experiment 1.1.1: ORB vs AprilTag z-axis orientation.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	0	0	0.016128	0	0	0.10526
AprilTag	0.30715	0.18251	0.068406	0.024184	0.023283	0.025649

Table 4.4: Experiment 1.1.1: Comparison between mean and standard deviation from both algorithms on its orientation.

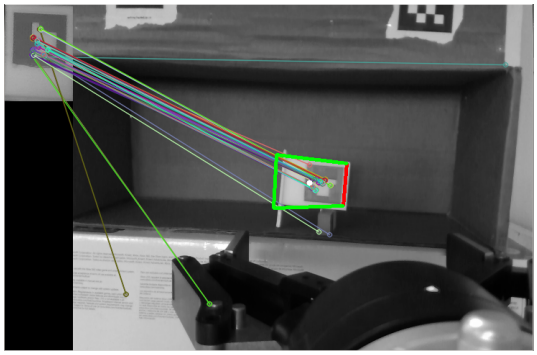


Figure 4.16: Experiment 1.1.1: ORB 90 degree detection.

estimation. On Table 4.5 we notice a similar pattern found on the last experiment, with similar values for the standard deviation, showing that both algorithms can remain with the same quality of detection even when the object is subject to rotation.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	0.0042219	0.0038176	0.35500	0.0017094	0.0019355	0.00053813
AprilTag	0.011738	0.0067193	0.35500	0.000009762	0.000005599	0.0002958

Table 4.5: Experiment 1.1.1: Comparison between mean and standard deviation from both algorithms on its position when rotated 95 degrees.

Comparing the orientation, seen in Figures Figures 4.20, 4.21 and 4.22, we notice a

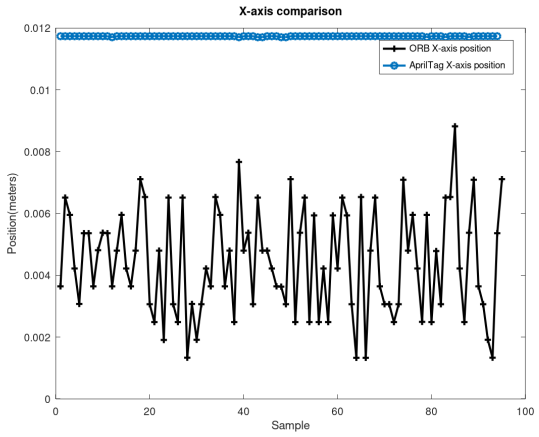


Figure 4.17: Experiment 1.1.1: ORB vs AprilTag x-axis position with 95 degree rotation on Z axis.

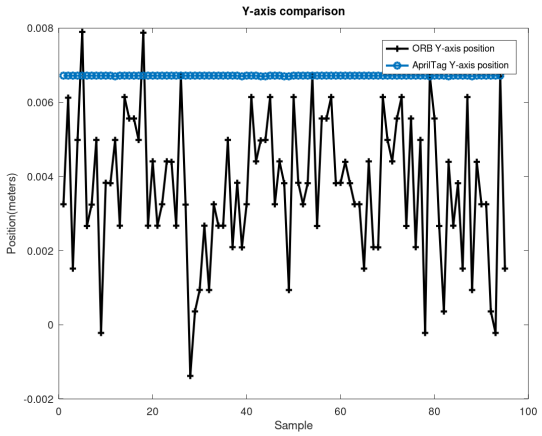


Figure 4.18: Experiment 1.1.1: ORB vs AprilTag y-axis position with 95 degree rotation on Z axis.

huge instability on ORB, which can be a problem when dealing with grasp on objects with any rotation. Table 4.6 confirms the results from the first part of this experiment, but with a even greater Std Dev on the *Z-Axis*.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	0	0	1.8019	0	0	0.12983
AprilTag	0.262	0.18282	1.6671	0.00098001	2.2324e-16	0.0016720

Table 4.6: Experiment 1.1.1: Comparison between mean and standard deviation from both algorithms on its orientation when rotated 95 degrees.

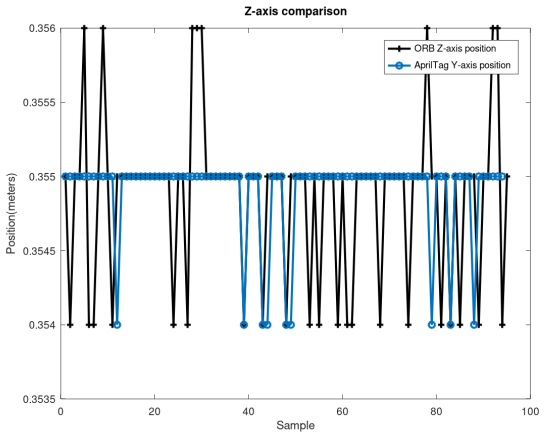


Figure 4.19: Experiment 1.1.1: ORB vs AprilTag z-axis position with 95 degree rotation on Z axis.

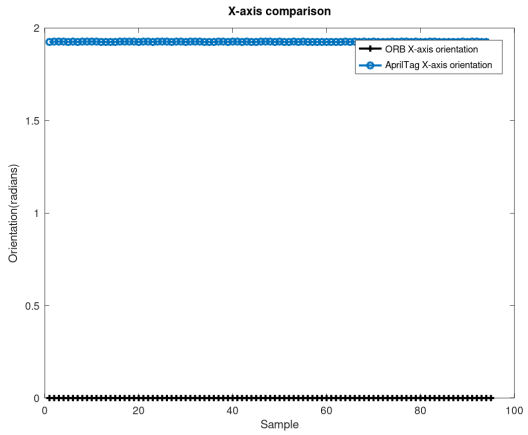


Figure 4.20: Experiment 1.1.1: ORB vs AprilTag x-axis orientation with 95 degree rotation on Z axis.

4.1.1.3 ORB vs AprilTag: Distance = 0.57 meters This experiment will be conducted in a similar way to the previous one, but with a distance of 0.57 meters from the camera to the object. On Figures 4.23 and 4.24, we can see that while the AprilTag can detect the object without issues, ORB is starting to have problems finding features from this far, resulting in a bad estimation of its position and geometry.

On Figures 4.25, 4.26 and 4.27, we see the comparison between the *(x,y,z)* position estimated by the algorithms. It is noticeable that, even with a similar results compared to last experiment, with a standard variation. The "empty" spots on the ORB mean that the object was completely lost for those frames.

On Table 4.7 we can see that while the mean of the position from both algorithms are close, the Std Dev from the ORB is more than ten times higher than we saw in Table 4.3, showing that from this distance the ORB becomes unreliable. Meanwhile, AprilTag is able to maintain the same detection quality from the last experiment, showing how

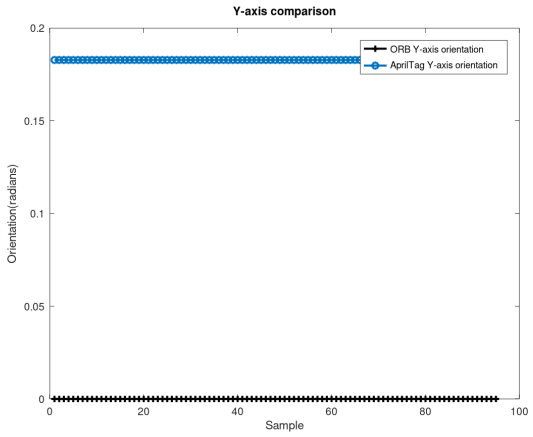


Figure 4.21: Experiment 1.1.1: ORB vs AprilTag y-axis orientation with 95 degree rotation on Z axis.

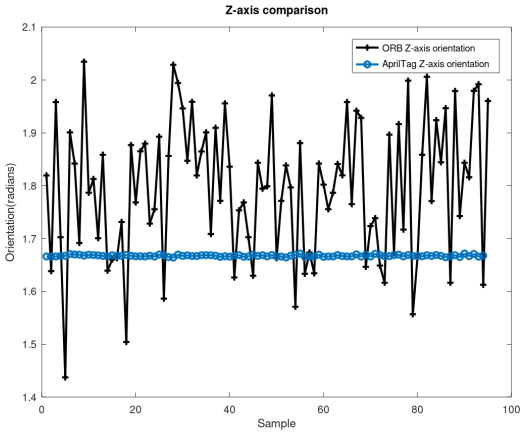


Figure 4.22: Experiment 1.1.1: ORB vs AprilTag z-axis orientation with 95 degree rotation on Z axis.

robust the algorithm is.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	-0.017387	0.0015077	0.57100	0.035116	0.037192	0.10429
AprilTag	-0.0099127	-0.0068382	0.56900	0.00000998	0.00000688	0.000573

Table 4.7: Experiment 1.1.2: Comparison between mean and standard deviation from both algorithms on its position.

Comparing the orientation detection, on Figures 4.28, 4.29 and 4.30, a small offset on the angles, since the camera may no be perfect aligned to the robotic arm axis and the object is a uneven surface.

On Table 4.8, we see a increase in the standard deviation from both algorithms, but

Figure 4.23: Experiment 1.1.2: Tag Detection.

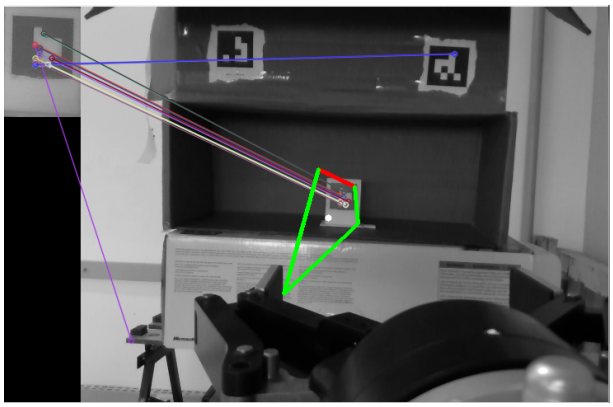


Figure 4.24: Experiment 1.1.2: ORB detection.

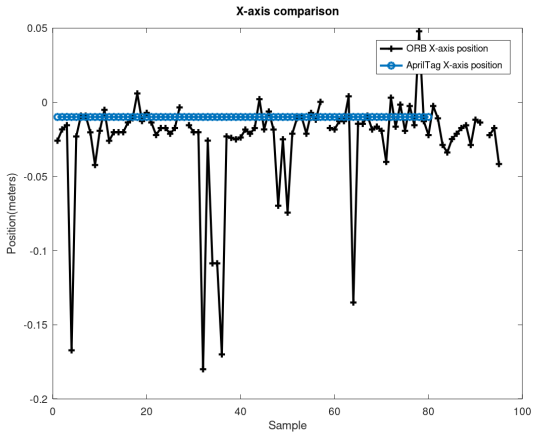


Figure 4.25: Experiment 1.1.2: ORB vs AprilTag x-axis position.

while the AprilTag had a small increase, ORB doubled in relation to Table 4.4 and had **Z-mean** value totally different than the one from AprilTag.

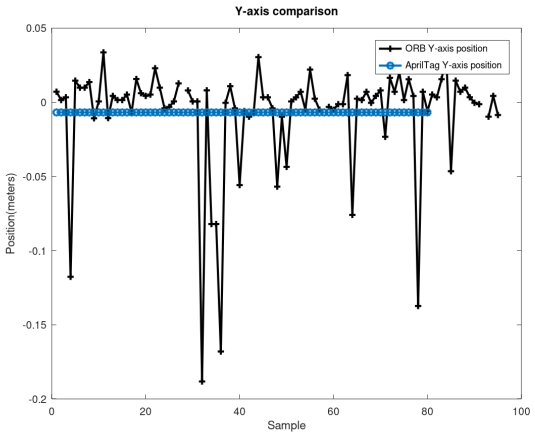


Figure 4.26: Experiment 1.1.2: ORB vs AprilTag y-axis position.

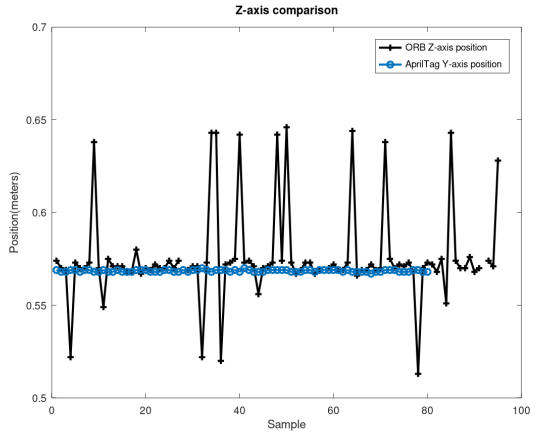


Figure 4.27: Experiment 1.1.2: ORB vs AprilTag z-axis position.

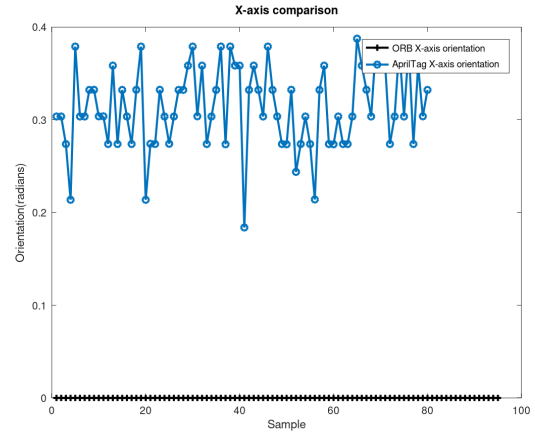


Figure 4.28: Experiment 1.1.2: ORB vs AprilTag x-axis orientation.

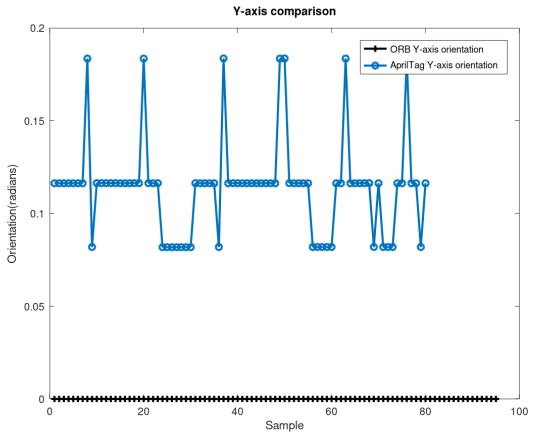


Figure 4.29: Experiment 1.1.2: ORB vs AprilTag y-axis orientation.

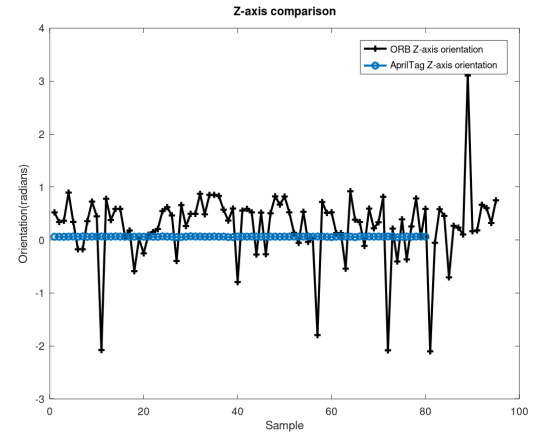


Figure 4.30: Experiment 1.1.2: ORB vs AprilTag z-axis orientation.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	0	0	0.36717	0	0	0.67262
AprilTag	0.30370	0.11632	0.063791	0.044292	0.026089	0.0033716

Table 4.8: Experiment 1.1.2: Comparison between mean and standard deviation from both algorithms on its orientation.

For the second part of this experiment, the target object was rotated **15** degrees on the **Y** axis. Figures 4.31 and 4.32 show the detection of the object. While AprilTag has no problems detecting it, ORB shows the same problem detecting objects from this distance.

The position data can be seen in Figures 4.33, 4.34 and 4.35. It is noticeable that while both algorithms are return a good estimation in this case, ORB returns a value out of the curve for one frame due to wrong estimation.

On Table 4.9 we see similar values for the position of the object using both algorithms

Figure 4.31: Experiment 1.1.2: Tag Detection while rotated 15 degrees on the Y axis.

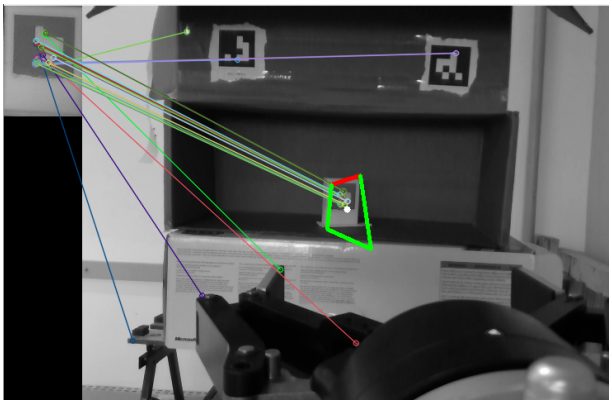


Figure 4.32: Experiment 1.1.2: ORB detection while rotated 15 degrees on the Y axis.

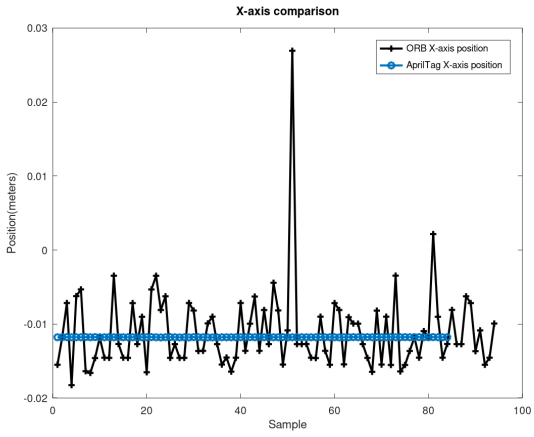


Figure 4.33: Experiment 1.1.2: ORB vs AprilTag x-axis position while rotated 15 degrees on the Y axis.

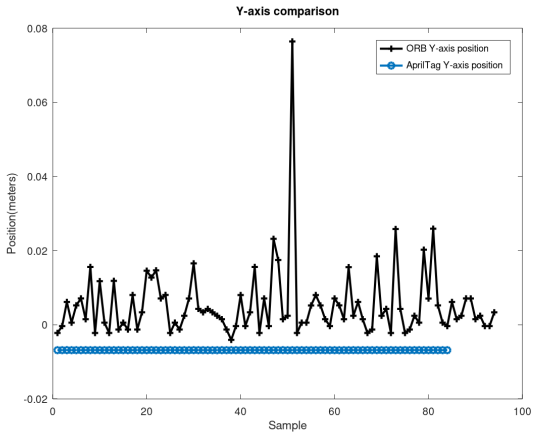


Figure 4.34: Experiment 1.1.2: ORB vs AprilTag y-axis position while rotated 15 degrees on the Y axis.

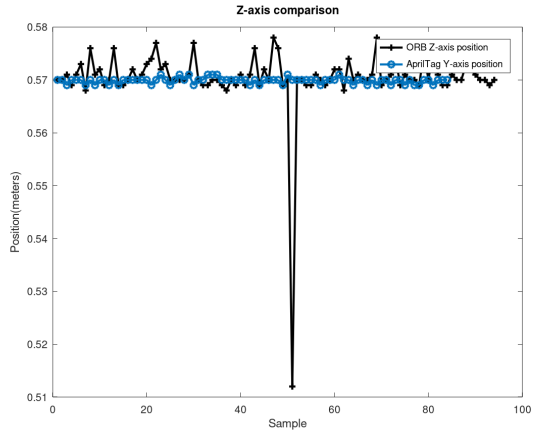


Figure 4.35: Experiment 1.1.2: ORB vs AprilTag z-axis position while rotated 15 degrees on the Y axis.

while ORB has a higher standard deviation.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	-0.012715	0.0024334	0.57000	0.0055426	0.00990	0.00651
AprilTag	-0.011787	-0.0068502	0.57000	0.000011630	0.00000676	0.000562

Table 4.9: Experiment 1.1.2: Comparison between mean and standard deviation from both algorithms on its position while rotated 15 degrees on the Y axis.

Figures 4.36, 4.37 and 4.38 show the orientation over time. While the **X** axis orientation got some lost values, all the orientations values are inside the expetected parameters.

On Table 4.10 we get values inside the expected parameters, with the **Z-Mean** value from both algorithms close, but, the standard deviation from ORB from this distance

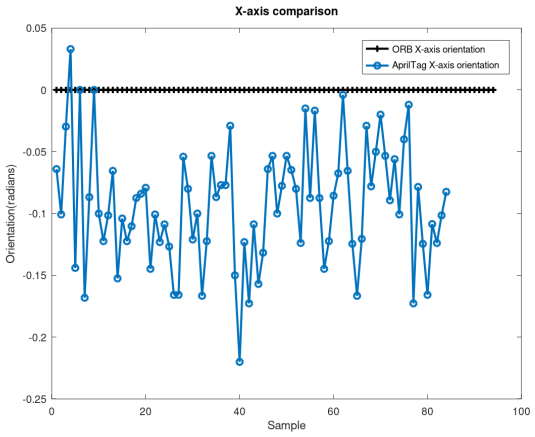


Figure 4.36: Experiment 1.1.2: ORB vs AprilTag x-axis orientation while rotated 15 degrees on the Y axis.

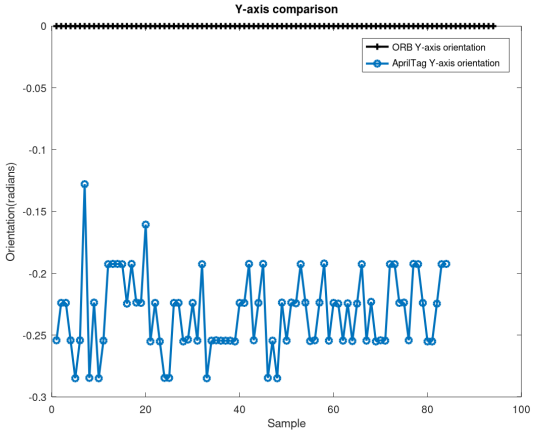


Figure 4.37: Experiment 1.1.2: ORB vs AprilTag y-axis orientation while rotated 15 degrees on the Y axis.

makes the orientation estimation not trustworthy.

	X-Mean	Y-Mean	Z-Mean	X-Std Dev	Y-Std Dev	Z-Std Dev
ORB	0	0	-0.050598	0	0	0.19742
AprilTag	-0.064378	-0.22437	0.015351	0.060445	0.031385	0.0096150

Table 4.10: Experiment 1.1.2: Comparison between mean and standard deviation from both algorithms on its orientation while rotated 15 degrees on the Y axis.

4.2 PICK AND PLACE SYSTEM

For these experiments, AprilTags and ORB will be used for object detection, where the first experiment will feature ORB picking a common day to day object, a calendar,

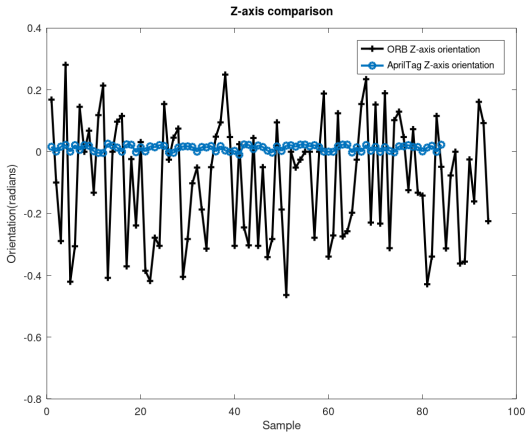


Figure 4.38: Experiment 1.1.2: ORB vs AprilTag z-axis orientation while rotated 15 degrees on the Y axis.

the second will compare ORB and AprilTag to grasp objects and the rest will feature AprilTag to grasp objects. Having an extra object with a tag attached to the target will avoid problems related to the object geometry, since we are using a single gripper with two fingers, in the UR5 case, and three fingers, in the JACO case. The experiments will features to types of systems: eye in hand and eye to hand, where the eye in hand will use the Intel Realsense camera and the eye to hand system will use the Microsoft Kinect image sensor.

4.2.1 Object on a table

In this experiment, the robotic manipulator JACO will be used with the visual sensor Microsoft Kinect. The system will use an eye to hand set up, as seen in figure 4.39, due to the Kinect limitations.

The arm will move from its initial position, seen in Figure 4.39, to the desired object position, in this experiment case, a calendar seen in Figure 4.40. After grasping the object, it will lift it and move to another position.

Figures 4.41 and 4.42 show the trajectory and velocity over time for cubic polynomial trajectory, according to the robot sensors, where it accelerates when it is far and slows down when approaching the objective.

When comparing the estimated object position by the Kinect with the end effector position, as seen in Figure 4.43, the object can be seen following the arm after the grasp. It is also possible to see that when the grasp happens, it covers some features from the object, resulting in the RANSAC estimating the object center in different positions, causing the "noise", and, some times, total loss. If the object had a low amount of features, even if it was detectable, after the grasp the object would be completely lost. On Table 4.11 we can see the Initial Position of the arm, the position where the manipulator picks the object and the final position acording to the sensors.

Figure 4.39: Initial position of the arm.

Figure 4.40: Object to be grasped.

4.2.2 Object inside a box

In this experiment, we are going to compare ORB and AprilTag to grasp objects using tags. On our previously published article, (CONCEICAO; OLIVEIRA; CARVALHO,

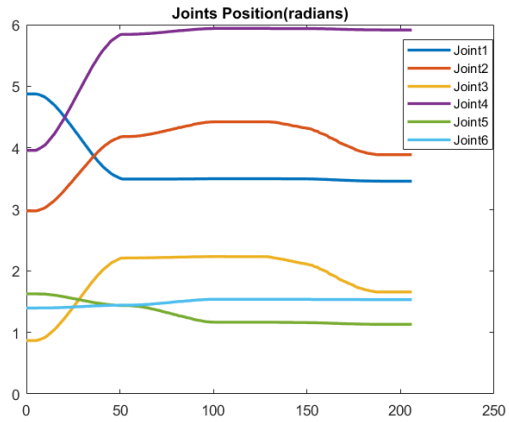


Figure 4.41: Joint positions over time - experiment 1.1.

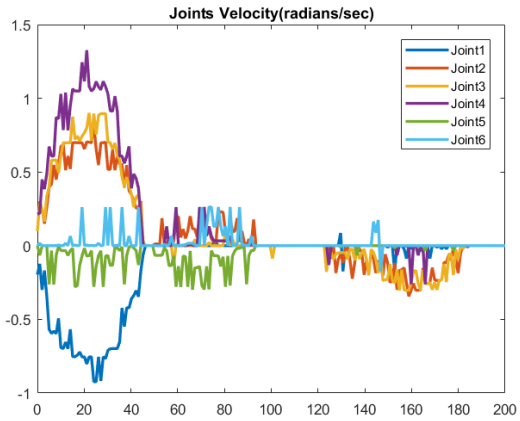


Figure 4.42: Joint velocities over time - experiment 1.1.

2018), ORB was applied to detect and grasp a calendar, as seen in Figure 4.44. Since the calendar was an object rich with features, its detection was accurate. But, in this experiment we are going to use an AprilTag features to be detect by ORB and compare it to the algorithm to detect AprilTags. While in the previously published work we used the Kinect in a eye to hand environment, in these experiment the Intel D435 will be used, since it is going to be used an eye in hand set up and the Kinect minimal distance on the depth sensor will make this set up impracticable.

When it comes to detection, both algorithms can do the job as seen in Figures 4.45 and 4.46. We can see some mismatches using ORB but most of the time the detection is accurate when comparing the boundbox generated to the reference image. When it comes to execution time, ORB can detect one object in 0.013 seconds while AprilTag algorithm takes 0.046 seconds to find all tags in the image. So, in a system where execution time is crucial or lack processing power, the ORB will be better to detect a single object, but if

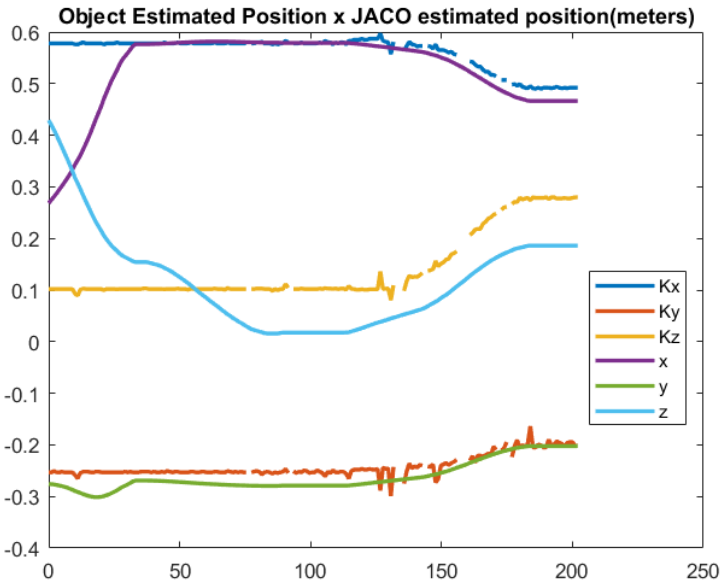


Figure 4.43: Estimated object position by the Kinect versus end effector position over time - experiment 1.1.

	x	y	z
Initial Position	0.211925	-0.27501	0.4961
Pick Position	0.578906	-0.278942	0.017514
End Position	0.466422	-0.202357	0.186079

Table 4.11: Initial, Pick and End position for this experiment.

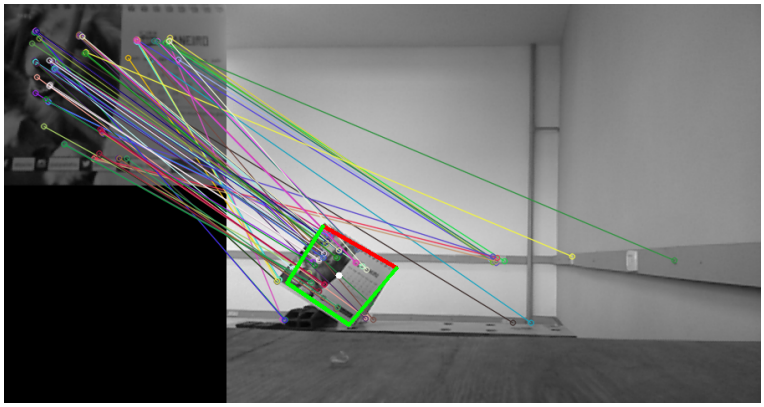


Figure 4.44: Experiment 1.2: ORB Detection.

we are going to detect more than one tag, the AprilTag may take the edge since it find all tags in the scene in a single execution. To compare these algorithms, a experiment where the arm moves to the object position, with a single movement, and grasp it will

be executed.

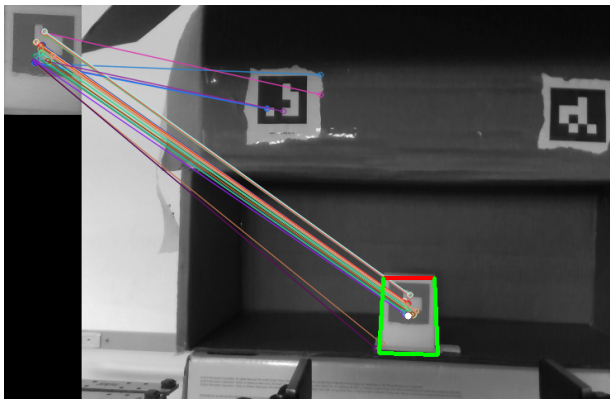


Figure 4.45: Experiment 1.2: Tag detection using ORB.

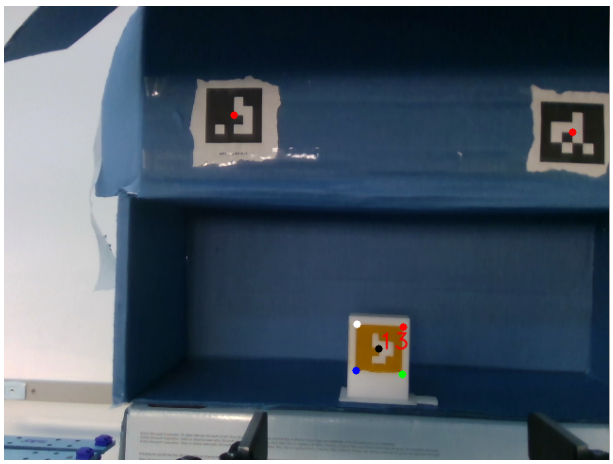


Figure 4.46: Experiment 1.2: AprilTag detection.

Analyzing the Figure 4.47, we see the arm position *(x,y,z)* converging to the tag position *(tx,ty,tz)* using the AprilTag algorithm, where the position *zero* means the object was lost. The object was lost when the gripper covered the tag while moving and after the grasp. The same result can be seen on Figure 4.48, where the tag position tends to *zero* when the arm comes close to its position.

Repeating the same experiment using ORB, we get the plots seen in Figures 4.49 and 4.50. In general, the results looks similar to the ones found on AprilTag but the plot has some noise, that happens because the ORB sometimes does not found the same features in two different frames, which may result in a position estimation slightly different. When the gripper closes, it covers some features, resulting in a wrong estimation of the object position, as seen on the plots after sample 100.

Since the tags are low on features, the features found on the environment are non-deterministic and some mismatches will happen with features in the environment, it may result in missing the grasp in some cases, thanks to wrong object position estimation.

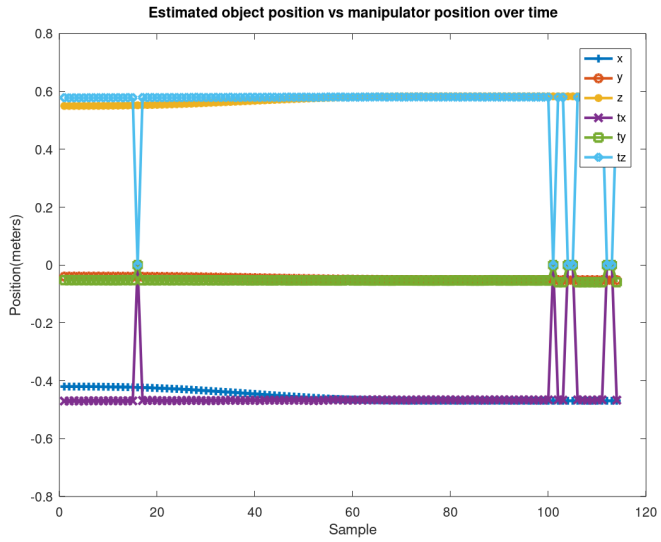


Figure 4.47: Experiment 1.2: Estimated tag position vs arm position.

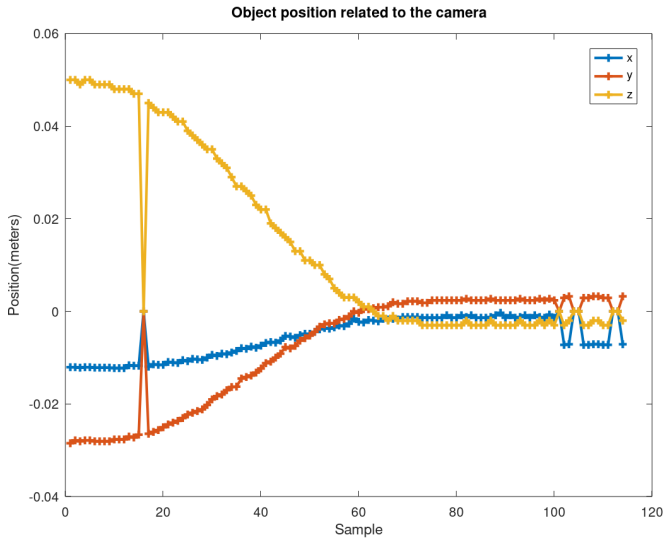


Figure 4.48: Experiment 1.2: Tag position relative to the camera.

On Figures 4.51, 4.52 and 4.53, we can see multiples frame after a detection, and all of them return a slightly different object center, with exception of Figure 4.53, where the geometry and object position does not match with reality and may cause problems when estimating its position.

On Table 4.12, we see the initial position, where it is the same for both experiments, and the position it grasps the object. We notice that the pick position from both algorithms is really similar, with the position in difference being related to the way the detect the object, mentioned before.

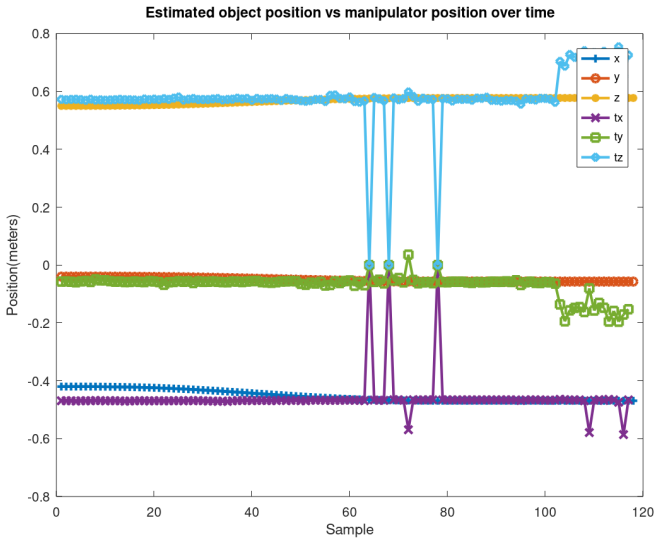


Figure 4.49: Experiment 1.2: Estimated tag position using ORB vs arm position.

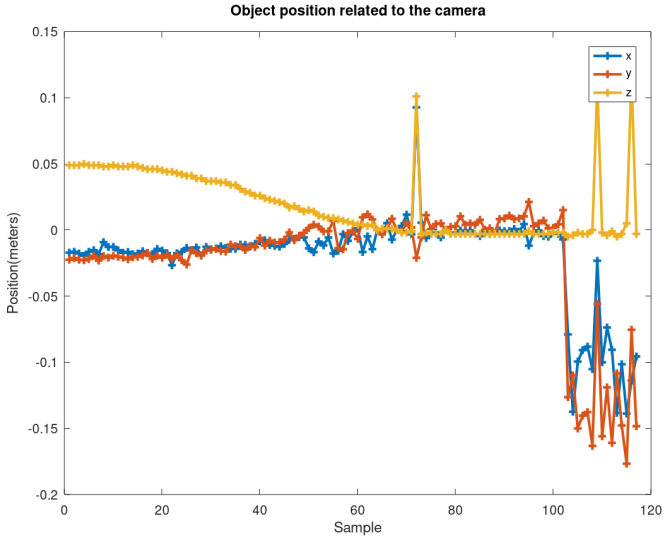


Figure 4.50: Experiment 1.2: Tag position relative to the camera using ORB.

	x	y	z
Initial Position	-0.42	0.4	0.55
ORB Pick Position	-0.469	-0.0570832	0.5775
Tag Pick Position	-0.468961	-0.0521626	0.583116

Table 4.12: Initial and Pick position from both experiments.

4.2.3 Pick and Place: Object inside a 3D printer

This experiment is set up as seen in Figure 4.54, where the printer has two tags that are used to find it and move the arm closer, making easier to find the object inside it. After

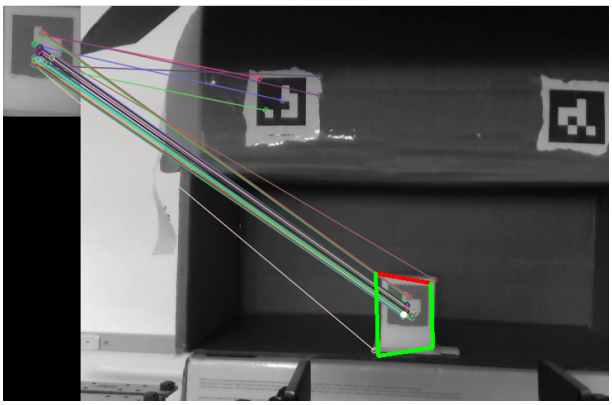


Figure 4.51: Experiment 1.2: Tag detection using ORB 2.

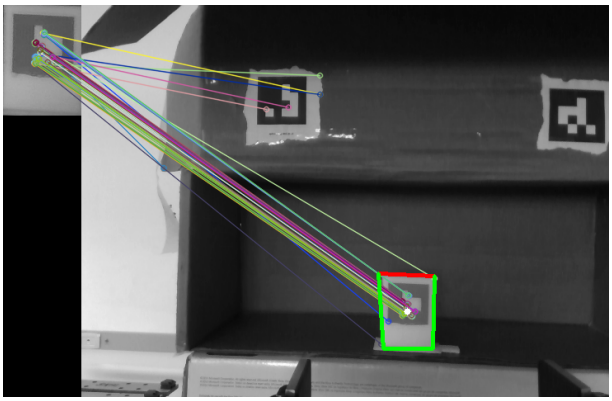


Figure 4.52: Experiment 1.2: Tag detection using ORB 3.

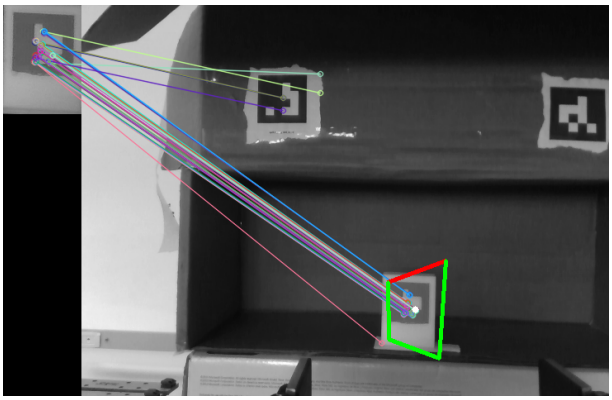


Figure 4.53: Experiment 1.2: Tag detection using ORB 4.

detecting the AprilTags in the printer, the arm will search the object tag and move to grasp it. After the grasping, it will remove the object from the printer and place it on the table.

On Figure 4.55 can be seen the joint position over time, where the transition between positions is smooth and non linear thanks to the Quintic Polynomial Trajectories. The

Figure 4.54: Experiment 1.3: Set up.

joint velocities on Figure 4.56 shows the speed changes over time, where it accelerates when far away from the objective and slow down when getting close to the destination.

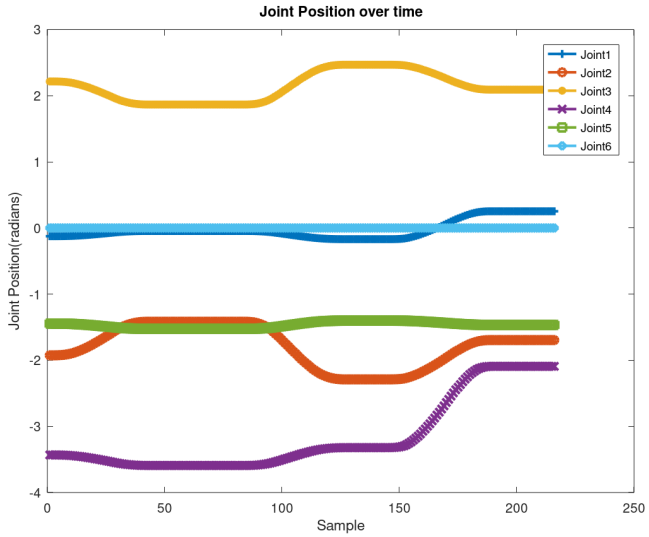


Figure 4.55: Experiment 1.3: Joint position over time.

On Figure 4.57 we can see the estimated object position by the homogeneous transform vs the arm position, considering *(tx,ty,tz)* the estimated object position and *(x,y,z)* the arm position. It is noted that the arm moves to the object position and after the grasp the object is lost by the tag detection algorithm (tx,ty and tz equals to zero when the object is lost). On Figure 4.58 we can confirm the arm approach to the object, since the object position relative to the camera tends to zero until the object is released on the table. On Table 4.13 we see the initial position, where it picks the object and the arm position where it releases the object on the table.

On Figure 4.59 we can see the gripper position over time, where 0 is open and 255 is closed. Around seven seconds, the gripper position is closed, meaning the object is grasped. Figure 4.60 it is shown the gripper status, where:

- 0: Waiting or moving.

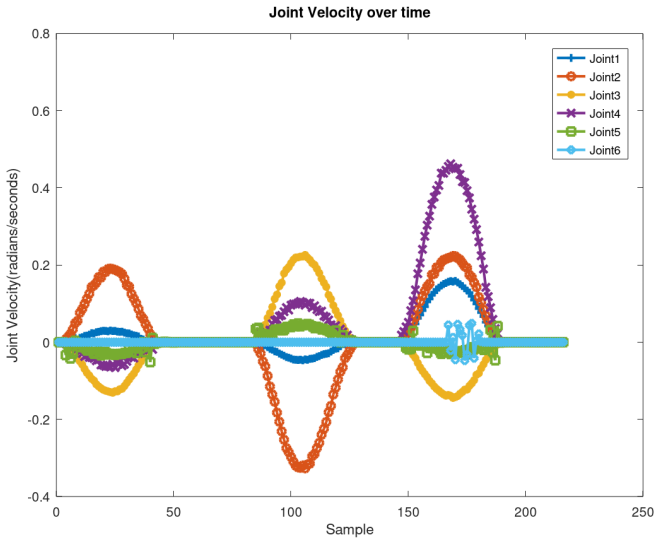


Figure 4.56: Experiment 1.3: Joint velocity over time.

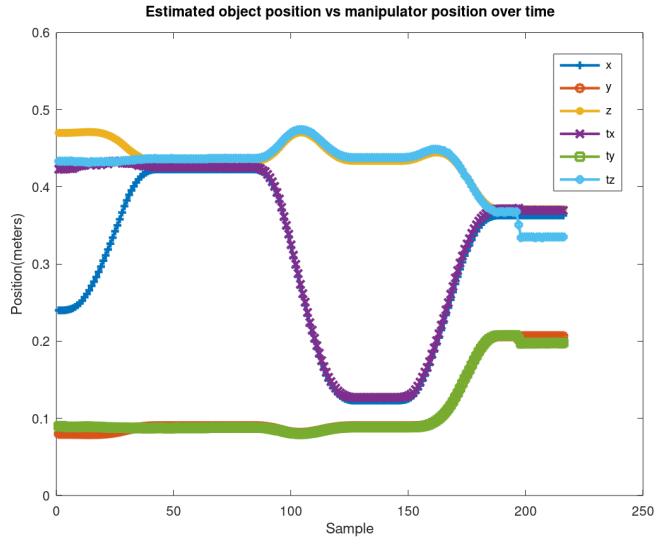


Figure 4.57: Experiment 1.3: Estimated object position vs arm position.

	x	y	z
Initial Position	0.24	0.08	0.47
Pick Position	0.424711	0.0905327	0.441354
Place Position	0.363021	0.206499	0.37097

Table 4.13: Initial, Pick position and Place Position.

- 1: Object detected while opening the gripper.

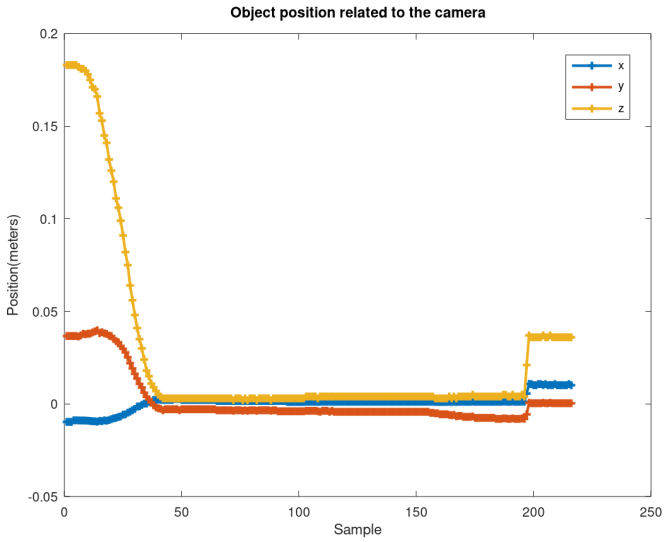


Figure 4.58: Experiment 1.3: Object position relative to the camera.

- 2: Object detected while closing the gripper.
- 3: Movement completed without object detection.

Comparing Figures 4.59 and 4.60, we can see that when the gripper is closed, it grasps the object, since the gripper status changes to two. The gripper status and position changes by the end of the arm movement, where it is close to the table and release the object on it.

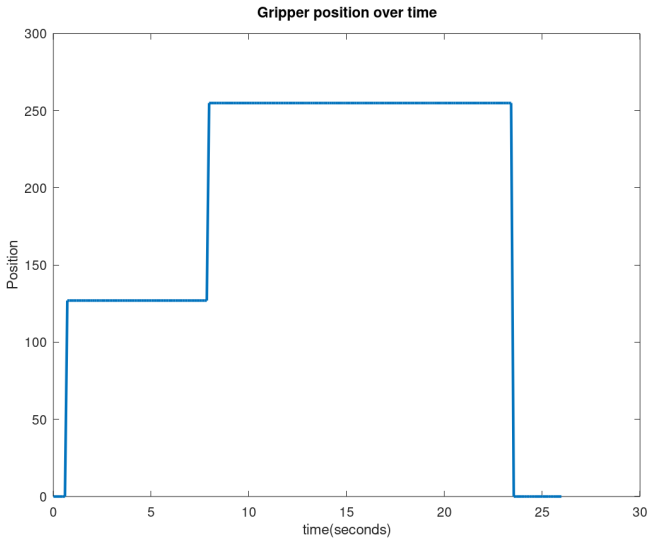


Figure 4.59: Experiment 1.3: Gripper position over time.

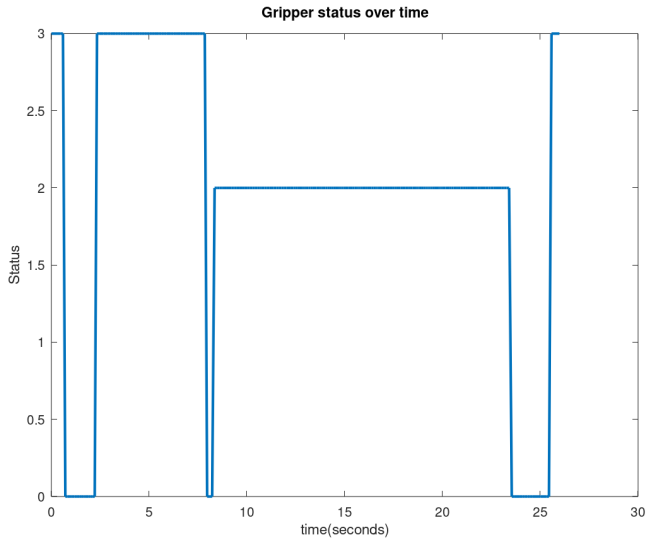


Figure 4.60: Experiment 1.3: Gripper status over time.

4.2.4 Error Control and Repeatability

In this experiment, the robotic manipulator will move to the tag position over forty times and grasp it, as seen in the video UR5 grasp (OLIVIERA, 2019). It will be discussed error treatment and repeatability based on those executions. The arm will move using a state machine, seen in Figure 4.61, where each state is a ROS action. This state machine will ensure that when each action ends in success, it will continue with the grasping and when an action fail, it will return to the initial position while alerting of the occurred error.

Each action will work as follow:

- Inicio: Will move the arm to the initial position and initialize the gripper. Will fail if can not reach the position, collides or can't initialize the gripper.
- Detect: Will detect the tag. If the tag is not found, the action will fail.
- Pick: Will move to the object position. Will fail if the tag is lost or collides.
- Grasp: Closes the gripper. Fail if the gripper does not hold the object.

After executing the state machine forty three times, we had forty cases of success and three errors, where the errors were: could not find the tag after thirty executions, that happened since each time the gripper close it would drag the object a couple of millimeters to the left, until it was covered by the gripper on the initial position, making the tag not detectable; could not initialize the gripper due to ROS driver problem; could not move due to collision with an object that was put in the environment. The Figure 4.62 shows the arm movement over the experiment, where we can see the arm moving slightly on the *Y axis* and *X axis* due to the object being dragged when the gripper closes

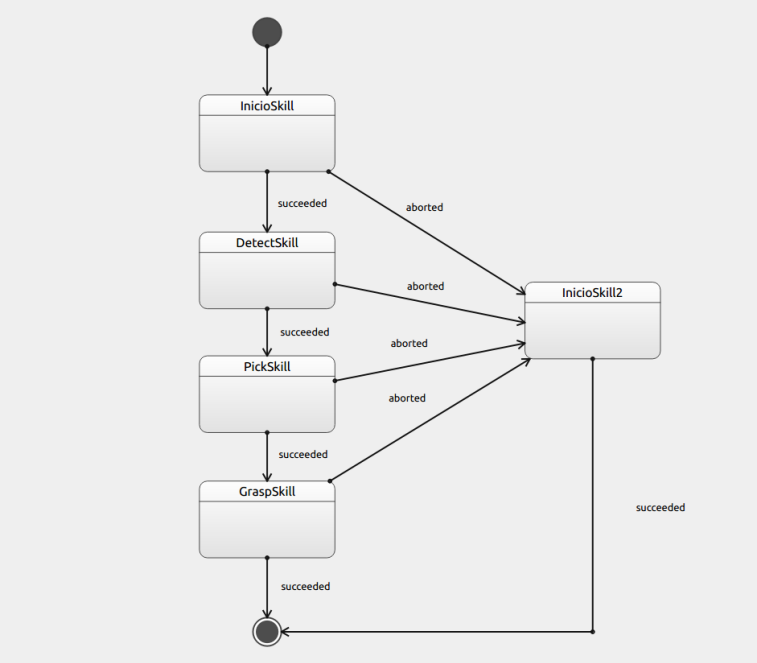


Figure 4.61: Experiment 1.4: State Machine.

on the previous experiment. The object being dragged happened due to small errors in the process of calibrating the extrinsic camera parameters, resulting the arm moving a couple of millimeters to the left.

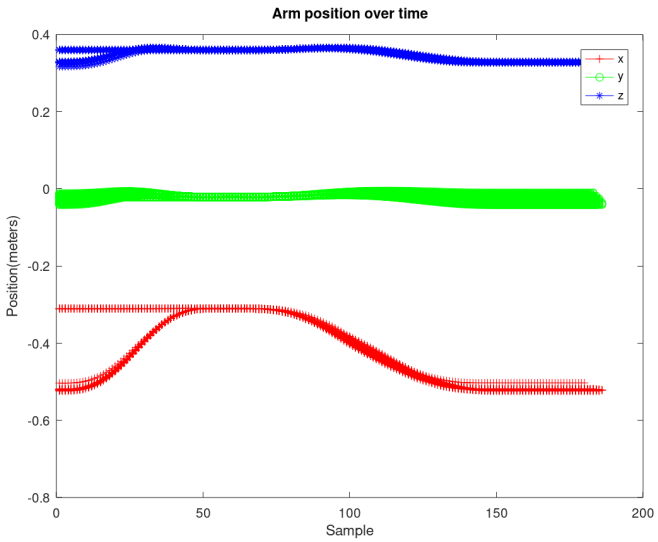


Figure 4.62: Experiment 1.4: Arm position over the experiments.

When analyzing the position of the arm vs the estimated position of the object, we having the error over the experiments seen in Figure 4.63, where the error is calculated as *arm position when grasping - object estimated position*. The Table 4.14 we can see that

	x	y	z
Mean Error	0.00034340	0.00011117	0.00015905
Minimum Error	0.00032100	0.000043800	0.00012500
Maximum Error	0.00036100	0.00015890	0.00019800

Table 4.14: Mean, maximum and minimum position error.

	x	y	z
Mean Error	0.000022000	-0.000050590	-0.00012425
Minimum Error	-0.00016000	-0.00018574	-0.00028000
Maximum Error	0.00015000	0.00017212	0.00014000

Table 4.15: Mean, maximum and minimum orientation error.

the *X axis* had the highest error while the *Y axis* had the smallest error. Considering that the maximum error found was around 0.361 millimeters, we can see that the arm is precise with the commands given to it, so the only way for it to not grasp the object depends on others factors, like vision algorithms precision, obstacles or human intervention.

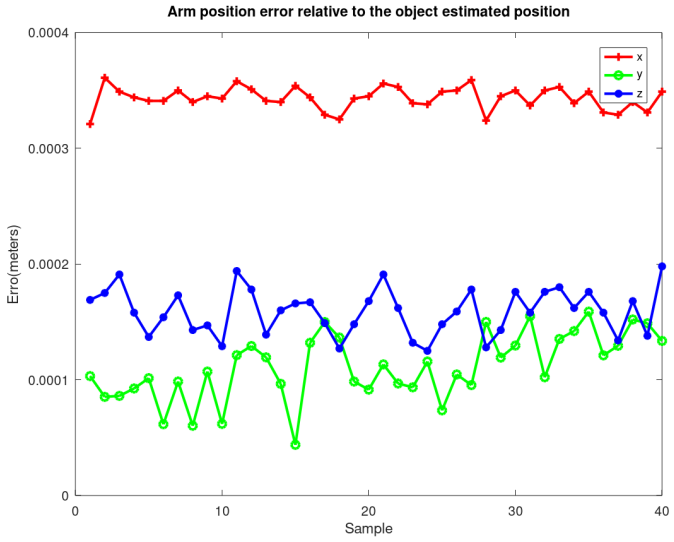


Figure 4.63: Experiment 1.4: Arm position error compared to the object estimated position.

The arm orientation error can be seen on Figure 4.64, where the error is calculated as *arm final orientation - arm desired orientation*. On Table 4.15 we see that like the position, the orientation has a small error comparing to the desired orientation where the maximum error is around 0.00017212 radians, showing the precision of the manipulator.

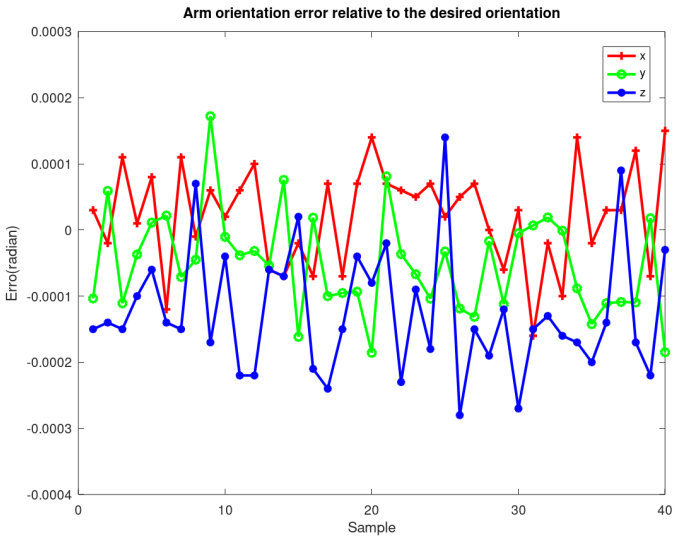


Figure 4.64: Experiment 1.4: Arm orientation error.

4.2.5 Mobile Manipulator

For this experiment, the robotic manipulator will be mounted on the mobile robot Husky (CLEARPATH, 2019), manufactured by Clearpath, as seem in Figure 4.65. In this experiment, the mobile robot will move to the printer position and , after getting there, the robotic manipulator will pick the object inside it, as seen in the video (LAR, 2019).

Figure 4.65: UR5 mounted on Husky.
Source: Clearpath.

To perform the task, the arm will execute the following movements:

- Home Position: Moves the arm to a retracted position for navigation. This is made to avoid the arm colliding with the environment while navigating.

- Initialize Arm: Initialize the gripper and moves the arm to a predetermined position pointing to the direction of the printer.
- Pick Object: Detect the printer position based on tags, pick the object from the printer and remove it from the printer. The tags from the printer and from the object are predetermined in the state machine, so each execution can have different tags.
- Place Object: Place the object on a tag over the Husky.

On Figure 4.66, we can see the arm movement, where it starts with the *Initialize Arm* and ends in *Home Position*. Figure 4.67 shows the arm *(x,y,z)* versus the object estimated position *(tx,ty,tz)*, where the object is detected when starting the *Pick Printer Skill* and after the grasp, the object is lost. Analyzing the graph we see the arm moving to the object estimated position and after the grasp it is possible to see the object following the arm position for a couple of frames until it is no longer detectable. On Table 4.16 we see the positions where each movement happens and its events, like pick and place position, considering that the *Start Position* of the movement is the same as the *End Position* of the last movement.

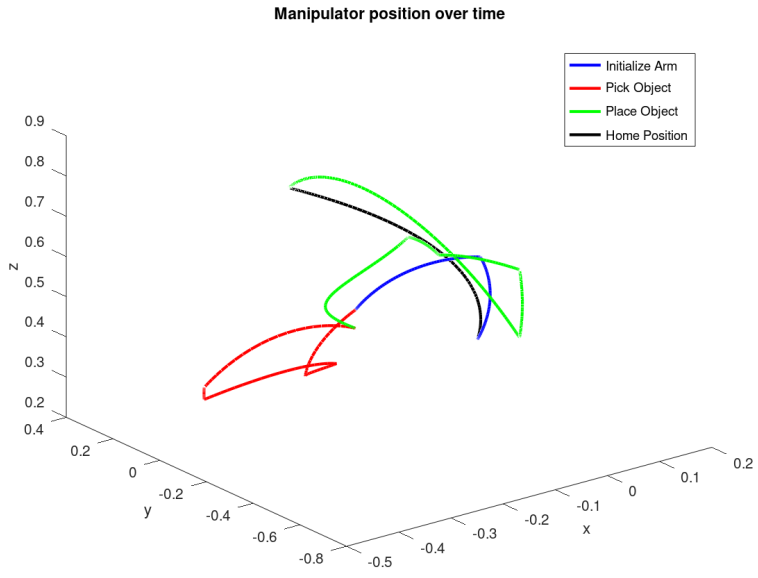


Figure 4.66: Experiment 2: 3D movement of the arm.

On Figures 4.68 and 4.69, we can seem the gripper position and gripper status respectively, where the status is *2* when the object is grasped, around sample 390, and the gripper opens at the end of the *Place Object*.

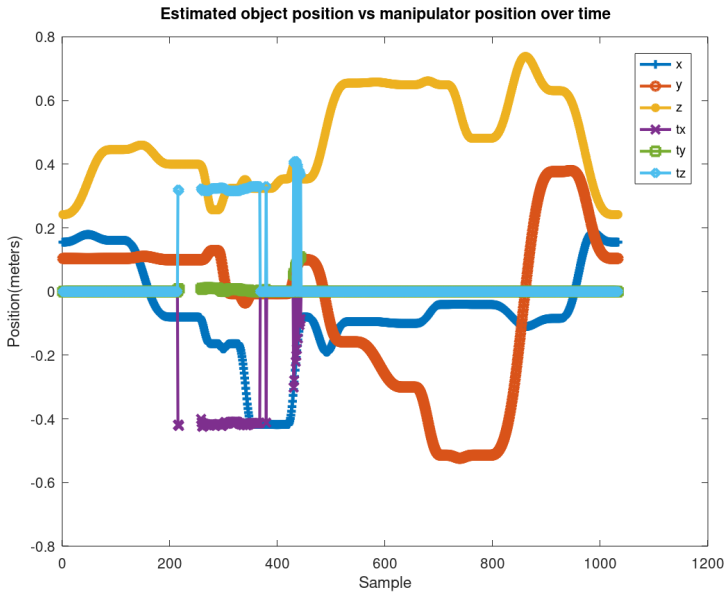


Figure 4.67: Experiment 2: Estimated object position vs arm position.

	Initialize Arm		
	x	y	z
Start Position	0.155315	0.103986	0.241986
End Position	-0.0800062	0.0999956	0.399968
	Pick Object		
	x	y	z
Pick Position	-0.417647	-0.00624312	0.32385
End Position	-0.0799801	0.0999973	0.353852
	Place Object		
	x	y	z
Place Position	-0.040636	-0.514095	0.481204
End Position	-0.0843788	0.376869	0.630957
	Home Position		
	x	y	z
Home Position	0.155311	0.103988	0.241997

Table 4.16: Start, Pick, Place and Home position

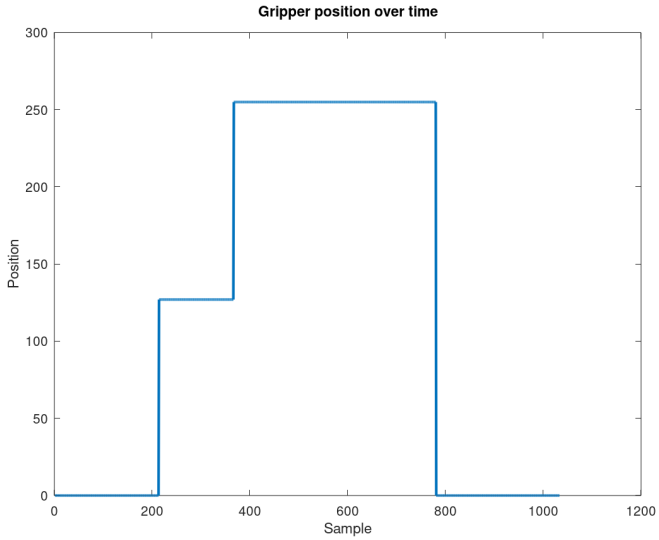


Figure 4.68: Experiment 2: Gripper position over time.

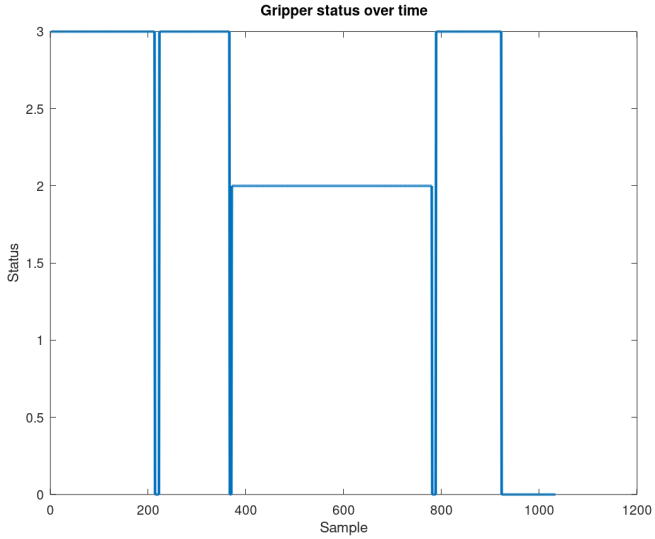


Figure 4.69: Experiment 2: Gripper status over time.

Chapter 5

CONCLUSION

This work presented practical results obtained by developing a pick and place system based on robotic vision, where two algorithms were used to identify the desired object to grasp: ORB and AprilTags. Based on the object position in 2D space, a RGB-D visual sensor was used to translate the 2D point to 3D coordinates and, with those coordinates, use Homogeneous Transformation and Inverse Kinematics to pick it.

To control the robotic manipulator, a study to find equations that describe its kinematics was necessary, since each robotic manipulator has its unique equations that describe its actual and desired joint position. With the kinematics done, it was needed an equation to convert the object position related to the camera to the arm coordinates so Homogeneous Transformation was used. To make the arm movement smooth, Quintic Polynomial Trajectory was used.

With the algorithms to move and control the arm done, computer vision algorithms were needed to find and localize the object to be grasped. The ORB is a feature algorithm that combined with RANSAC can be used to detect objects based on a reference image. While it has good performance, its weak point is the need for features, so featureless objects or objects with low amount of features are troublesome to be detected. Another way used to detect objects was based on tags, where AprilTag were used. While fast and reliable, it suffers the problem of the need for a tag, which in some real life applications may be unfeasible.

Since we are working in 3D space, after the pixel position localization of the object is detected, we need to convert its position to *(x,y,z)* coordinates, since we are using a RGB-D sensor, this can be done by converting pixel to point cloud coordinates. With theses coordinates, using a PBVS becomes intuitive and simple to use in the application.

To compare the efficiency of the algorithms, an visual data study was performed. Both algorithms were compared based on distance to camera and orientation relative to the camera. When it comes to pick and place, diverse experiments were made, using a eye on hand and eye to hand system. In these experiments, pick and place tasks were made using ORB and AprilTag in different environments. To show that the task was

performed, the object position is compared to the arm position, in order to show that the arm moved to the object position correctly and grasped it.

When comparing the visual data, it was shown that while the ORB can be used to detect objects, it has problems when it has a low numbers of features, generating wrong estimations, not making it reliable to grasping when the object is far away or partially covered. Meanwhile the AprilTag showed trustworthy in the experiments, showing its reliability.

On pick and place tasks, both algorithms can be used if within both limitations. When using an object rich with features, ORB becomes reliable even when there is some distance from the visual sensor. Grasping objects with an AprilTag attached to it made the task simple, since we don't need to know the object geometry to grasp it in the correct position. In both cases, the pick and place tasks were executed without any problem or setback.

The plots in the experiments shows the smooth joint trajectories thanks to the trajectory planner used. When analyzing the arm position, object estimated position and gripper sensor, we showed that the object was correctly grasped, since not only the arm was on the object position with a small estimation error, but the gripper sensor also confirmed the grasp.

To conclude, the proposed system is proved to complete pick and place tasks using two computer vision algorithms to detect objects, even with some limitations. Future works will aim to improve the object detection and grasp algorithms using Deep Learning and the point cloud.

BIBLIOGRAPHY

ANDERSEN, R. S. *Kinematics of a UR5*. 2018. ⟨http://rasmusan.blog.aau.dk/files/ur5_kinematics.pdf⟩. Accessed: 06-06-2019.

BAY, T. T. H.; GOOL, L. V. Surf: Speeded up robust features. *European Conference on Computer Vision*, 2006.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

BRIOT S.; KHALIL, W. *Dynamics of Parallel Robots: From Rigid Bodies to Flexible Elements*. 1ª. ed. : Springer International Publishing, 2015.

CALONDER V. LEPETIT, C. S. M.; FUA, P. Brief: Binary robust independent elementary features. *European Conference on Computer Vision*, 2010.

CHAUMETTE, F.; HUTCHINSON, S. Visual servo control. i. basic approaches. *Robotics Automation Magazine, IEEE*, v. 13, p. 82 – 90, 01 2007.

CHAUMETTE, F.; HUTCHINSON, S. Visual servo control, part ii: Advanced approaches. *IEEE Robot. Autom. Mag.*, v. 14, 01 2007.

CLEARPATH. *HuskyUGV*. 2019. ⟨https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/⟩. Accessed: 11-10-2019.

CONCEICAO, A. G.; OLIVEIRA, D. M.; CARVALHO, M. P. Orb algorithm applied to detection, location and grasping objects. In: . 2018. p. 176–181.

COOK, D. et al. Mavhome: An agent-based smart home. In: . 2003. p. 521 – 524. ISBN 0-7695-1893-1.

CORKE, P. *Robotics, Vision and Control: Fundamental Algorithms In MATLAB, Second Edition*. 2nd. ed. : Springer Publishing Company, Incorporated, 2017. ISBN 3319544128, 9783319544120.

CORMEN C. E. LEISERSON, R. L. R. T. H.; STEIN, C. *Introduction to Algorithms*. 3rd. ed. : MIT Press, 2009.

FIALA, M. Artag, a fiducial marker system using digital techniques. In: . 2005. v. 2, p. 590 – 596 vol. 2. ISBN 0-7695-2372-2.

FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, v. 24, p. 381–395, 06 1981.

FORSYTH, D. A.; PONCE, J. *Computer Vision: A Modern Approach*. 2003.

GUENNEBAUD, G.; JACOB, B. et al. *Eigen v3*. 2010. Http://eigen.tuxfamily.org.

HAN, D.-M.; LIM, J. Design and implementation of smart home energy management systems based on zigbee. *Consumer Electronics, IEEE Transactions on*, v. 56, p. 1417 – 1425, 09 2010.

HARRIS, C. G.; STEPHENS, M. A combined corner and edge detector. In: *Alvey Vision Conference*. 1988.

INTEL. *Intel Realsense Depth Camera D400*. 2019. ⟨https://software.intel.com/en-us/realsense/d400⟩. Accessed: 03-04-2019.

KIM, S.; OH, S.-Y. Hybrid position and image based visual servoing for mobile robots. *Journal of Intelligent and Fuzzy Systems*, v. 18, p. 73–82, 2007.

KINOVA. *KINOVA JACO Assistive robotic arm*. 2019. ⟨https://www.kinovarobotics.com/en/products/assistive-technologies/kinova-jaco-assistive-robotic-arm⟩. Accessed: 08-08-2019.

KLEIN, G.; MURRAY, D. W. Improving the agility of keyframe-based slam. In: . 2008. v. 5303, p. 802–815.

KULKARNI, A. V.; JAGTAP, J.; HARPALE, V. K. Object recognition with orb and its implementation on fpga. In: . 2013.

LAR. *Pick and Place task using vision-based control*. 2019. Https://www.youtube.com/watch?v=cjeQNrC0Rn0.

LAROUCHE, B.; ZHU, Z. H. Position-based visual servoingin robotic capture of moving target enhanced by kalman filter. *International Journal of Robotics and Automation*, v. 30, p. 267–277, 06 2015.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

MERCHANT, B. *The Biggest Sign Yet That Automation Is Taking Over at Amazon*. 2019. ⟨https://gizmodo.com/the-biggest-sign-yet-that-automation-is-taking-over-at-1831460994⟩. Accessed: 06-03-2019.

MOBINI, S. B. A.; FOUMANI, M. S. Accuracy of kinect's skeleton tracking for upper body rehabilitation applications. *Disability and rehabilitation. Assistive technology*, 2013.

MUR-ARTAL, R.; MONTIEL, J.; TARDOS, J. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, v. 31, p. 1147 – 1163, 10 2015.

OLIVIERA, D. M. de. *UR5 Grasp*. 2019. Https://drive.google.com/file/d/1-7qiFW3WiIGkV6V9USLy7DYvpQEdJo1S/view?usp=sharing.

PENG, C.-Y.; CHEN, R.-C. Voice recognition by google home and raspberry pi for smart socket control. *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*, p. 324–329, 2018.

POINTCLOUD2 Message. 2019. ⟨http://docs.ros.org/melodic/api/sensor_msgs/html/msg/PointCloud2.html⟩. Accessed: 02-05-2019.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: *ICRA Workshop on Open Source Software*. 2009.

ROBOTIQ. *Product Sheet: Adaptive Gripper*. 2019. ⟨https://blog.robotiq.com/hubfs/Product-sheets/Product-sheet-Adaptive-Grippers-EN.pdf?_ga=2.235506949.580940770.1553174797-282070942.1553174797⟩. Accessed: 21-03-2019.

ROBOTS, U. *Parameters for calculations of kinematics and dynamics*. 2019. ⟨https://www.universal-robots.com/how-tos-and-faqs/faq/ur-faq/parameters-for-calculations-of-kinematics-and-dynamics-45257/⟩. Accessed: 27-03-2019.

ROBOTS, U. *UR5: Technical details*. 2019. ⟨https://www.universal-robots.com/media/1801303/eng_199901_ur5_tech_spec_web_a4.pdf⟩. Accessed: 21-03-2019.

ROSIN, P. Measuring corner properties. *Computer Vision and Image Understanding*, v. 73, p. 291–307, 02 1999.

ROSTEN, E.; DRUMMOND, T. Machine learning for highspeed corner detection. *European Conference on Computer Vision*, 2006.

RUBLEE, E. et al. Orb: an efficient alternative to sift or surf. In: . 2011. p. 2564–2571.

SPONG, S. H. M. W.; VIDYASAGAR, M. *Robot Modeling and Control*. 1ª. ed. : John Wiley Sons, 2005.

TAREEN, S. A. K.; SALEEM, Z. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In: . 2018.

TEKE, B. *Real–time Target Tracking and Following with UR5 Collaborative Robot Arm*. Dissertação (Mestrado) — Tampere University of Technology, Tampere, Finland, 2018.

WANG, J.; OLSON, E. AprilTag 2: Efficient and robust fiducial detection. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016.

WANG, J. et al. Optimizing ground vehicle tracking using unmanned aerial vehicle and embedded apriltag design. In: *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2016. p. 739–744.

WESTMAN, E.; KAESS, M. Underwater apriltag slam and calibration for high precision robot localization. In: . 2018.

WILSON, W. J.; HULLS, C. C. W.; BELL, G. S. Relative end-effector control using cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, v. 12, n. 5, p. 684–696, Oct 1996. ISSN 1042-296X.

XIE, S. et al. Fast detecting moving objects in moving background using orb feature matching. In: *2013 Fourth International Conference on Intelligent Control and Information Processing (ICICIP)*. 2013. p. 304–309.

APPENDIX

A.1 SOURCE CODE

In this appendix we are going to find the codes related to the UR5 arm, computer vision algorithms, gripper control and the ROS actions used in some of the experiments.

UR5 Class

```
#include "ur5_arm.hpp"


Matrix4f UR5::AH(int n,VectorXf th){


    VectorXf d(6),a(6),alph(6);
    d << 0.089159, 0, 0, 0.10915, 0.09465, 0.0823;
    a << 0 ,-0.425 ,-0.39225 ,0 ,0 ,0;
    alph << M_PI/2, 0, 0, M_PI/2, -M_PI/2, 0;


    Matrix4f t_a,t_d,rzt,rxa;
    t_a << 1, 0 , 0 ,0,
           0 ,1 ,0 ,0,
           0, 0 , 1, 0,
           0, 0 , 0 ,1;
    t_d = t_a;

    t_a(0,3) = a(n-1);
    t_d(2,3) = d(n-1);


    rzt << cos(th(n-1)), -sin(th(n-1)), 0 ,0,
                   sin(th(n-1)),  cos(th(n-1)), 0, 0,
                   0,               0,              1, 0,
                   0,               0,              0, 1;

    rxa << 1, 0,  0,     0,
              0, cos(alph(n-1)), -sin(alph(n-1)),   0,
              0, sin(alph(n-1)),  cos(alph(n-1)),   0,
               0, 0, 0, 1;

    auto A_i = t_d * rzt * t_a * rxa;
    return A_i;
}

Matrix4f UR5::radianToRotation(float x,float y,float z,float ox,
   float oy,float oz){
    auto s_1 = sin(oz);
    auto c_1 = cos(oz);
    auto s_2 = sin(oy);
    auto c_2 = cos(oy);
    auto s_3 = sin(ox);
    auto c_3 = cos(ox);
```

```
    Matrix4f final_mat;
    final_mat << c_1*c_2   ,  c_1*s_2*s_3 - s_1*c_3   ,  c_1*s_2*c_3 +
        s_1*s_3 , x,
            s_1*c_2   ,  s_1*s_2*s_3 + c_1*c_3   ,  s_1*s_2*c_3 - c_1*
               s_3 ,y,
            -s_2          ,          c_2*s_3      ,              c_2*c_3
               ,z,
             0,0,0,1;
    return final_mat;

}

Vector3f UR5::rotationToEuler(Matrix4f matriz){
    Vector3f eul;
    if (matriz(2,0) < 1){
                    if (matriz(2,0) > -1){
                        eul(2) = atan2(matriz(1,0), matriz(0,0));
                        eul(1) = asin(-matriz(2,0));
                        eul(0) = atan2(matriz(2,1), matriz(2,2));
                    }else{
                        eul(2) = -atan2(-matriz(1,2), matriz(1,1));
                        eul(1) = M_PI/2;
                        eul(0) = 0;
                    }
                }else{
                    eul(2) = atan2(-matriz(1,2), matriz(1,1));
                    eul(1) = -M_PI/2;
                    eul(0) = 0;
                }
    return eul;
}


UR5::UR5(ros::NodeHandle n_,int number_of_movements){
    n=n_;
    arm=n.advertise<control_msgs::FollowJointTrajectoryActionGoal
       >("/follow_joint_trajectory/goal", 1);
    msg.header.stamp = ros::Time::now();

    msg.goal.trajectory.joint_names.push_back("shoulder_pan_joint");
    msg.goal.trajectory.joint_names.push_back("shoulder_lift_joint")
       ;
    msg.goal.trajectory.joint_names.push_back("elbow_joint");
    msg.goal.trajectory.joint_names.push_back("wrist_1_joint");
    msg.goal.trajectory.joint_names.push_back("wrist_2_joint");
    msg.goal.trajectory.joint_names.push_back("wrist_3_joint");
```

```
    msg.goal.trajectory.points.resize(number_of_movements);
    for (int i=0;i < number_of_movements;i++){
        msg.goal.trajectory.points[i].positions.resize(6);
        msg.goal.trajectory.points[i].velocities.resize(6);
        msg.goal.trajectory.points[i].accelerations.resize(6);
    }
    state.name.resize(6);
    state.velocity.resize(6);
    state.position.resize(6);
    state.effort.resize(6);
}

void UR5::set_state(const sensor_msgs::JointState::ConstPtr& now){
    for (int i=0; i < 6; i++){
        state.name[i]= now->name[i];
        state.velocity[i]=now->velocity[i];
        state.position[i]=now->position[i];

    }
}

void UR5::set_status(const actionlib_msgs::GoalStatusArrayConstPtr&
   now){
    int size=now->status_list.size();
    status=now->status_list[size - 1].status;
}

void UR5::publish(){
    arm.publish(msg);
}

void UR5::move_arm(float* pos,float duration,int movement_number){
    for (int i=0; i < 6; i++){
        msg.goal.trajectory.points[movement_number].positions[i]=pos
            [i];
    }
    msg.goal.trajectory.points[movement_number].time_from_start =
       ros::Duration(duration);
}

void UR5::move_arm_speed(float* speed,float duration,int
   movement_number){
    for (int i=0; i < 6; i++){
        msg.goal.trajectory.points[movement_number].velocities[i]=
            speed[i];
    }
    msg.goal.trajectory.points[movement_number].time_from_start =
```

```
        ros::Duration(duration);
}

void UR5::move_arm_acc(float* acc,float duration,int movement_number
    ){
     for (int i=0; i < 6; i++){
         msg.goal.trajectory.points[movement_number].accelerations[i
            ]=acc[i];
     }
     msg.goal.trajectory.points[movement_number].time_from_start =
        ros::Duration(duration);
}

void UR5::move_arm_planner(float* pos,int movement_number,float
    startTime,float endTime){
     float qq[10],vq[10],aq[10];
     float t0=startTime;
     //float tf=endTime-startTime;
     //float t=tf/10;
     //const float ta=tf/10;
     float tf=endTime;
     float t=(tf - t0)/10;
     const float ta=(tf - t0)/10;
     float q0=0;
     float qf=0;
     float v0=0;
     float vf=0;
     float ai=0;
     float af=0;
     float a0,a1,a2,a3,a4,a5;

     Matrix<float, 6, 1> b[6];

     Matrix<float, 6, 1> a[6];
     Matrix<float, 6, 6> m;

     for (int i=0;i < 6; i++){
         q0=get_state().position[i];

         qf=pos[i];
         b[i] << q0,v0,ai,qf,vf,af;
         m << 1, t0, t0*t0, t0*t0*t0,t0*t0*t0*t0,t0*t0*t0*t0,
              0, 1, 2*t0, 3*t0*t0,4*t0*t0*t0,5*t0*t0*t0*t0,
              0,0,2,6*t0,12*t0*t0,20*t0*t0*t0,
              1, tf, tf*tf, tf*tf*tf,tf*tf*tf*tf,tf*tf*tf*tf*tf,
              0, 1, 2*tf, 3*tf*tf,4*tf*tf*tf,5*tf*tf*tf*tf,
              0,0,2,6*tf,12*tf*tf,20*tf*tf*tf;
```

```
        a[i]=m.inverse()*b[i];
    }
    for (int i=movement_number; i < movement_number+10; i++){
            for(int j=0;j < 6; j++){
                a0=a[j].coeff(0,0);
                a1=a[j].coeff(1,0);
                a2=a[j].coeff(2,0);
                a3=a[j].coeff(3,0);
                a4=a[j].coeff(4,0);
                a5=a[j].coeff(5,0);
                qq[j]= a0+a1*t+a2*t*t+a3*t*t*t+a4*t*t*t*t+a5*t*t*t*t*t
                    *t;
                vq[j]= a1+2*a2*t+3*a3*t*t+4*a4*t*t*t+5*a5*t*t*t*t;
                aq[j]= 2*a2+6*a3*t+12*a4*t*t+20*a5*t*t*t;

            }
            move_arm(qq,startTime+t,i);
            move_arm_speed(vq,startTime+t,i);
            move_arm_acc(aq,startTime+t,i);
            t=t+ta;
     }

}


bool UR5::ik_move(float x,float y,float z,float thetax,float thetay,
   float thetaz,int movement_number,float startTime,float endTime){
    //float d1 =  0.1273;
    float a2 = -0.425;
    float a3 = -0.39225;
    //float a7 = 0.075;
    float d4 =  0.10915;
    //float d5 =  0.1157;
    float d6 = 0.0823;

    float pos[6];
    Matrix4f desired_pos;
    desired_pos= radianToRotation(x,y,z,thetax,thetay,thetaz);

    VectorXf th(6);

    VectorXf d(6),a(6),alph(6);
    d << 0.089159, 0, 0, 0.10915, 0.09465, 0.0823;
    a << 0 ,0,-0.425 ,-0.39225,0 ,0;
    alph << 0,  M_PI/2, 0,0, M_PI/2, - M_PI/2;

    MatrixXf p1(4,1),p2(4,1),p3(4,1);
```

```
p1 << 0,0, -d6, 1;
p2 << 0,0,0,1;
p3 << 0, -d4, 0, 1;

MatrixXf P_05(4,1);
P_05=desired_pos * p1 - p2;

//**** theta1 ****

auto psi = atan2(P_05(1,0), P_05(0,0));
auto phi = acos(d4 /sqrt(P_05(1,0)*P_05(1,0) + P_05(0,0)*P_05
    (0,0)));

th(0) = M_PI/2 + psi + phi;


//******theta 5******
auto T_10 = AH(1,th).inverse();
auto T_16 = T_10 * desired_pos;
th(4)=-acos((T_16(2,3)-d4)/d6);


//**** theta 6 ******
auto T_17=T_16.inverse();
th(5)=atan2((-T_17(1,2)/sin(th(4))),(T_17(0,2)/sin(th(4))));


//***** theta 3 ******
//T_10 = linalg.inv(AH(1,th,c));
auto T_65 = AH( 6,th);
auto T_54 = AH( 5,th);
auto T_14 = ( T_10 * desired_pos) * (T_54 * T_65).inverse();
auto P_13 = T_14 * p3 - p2;
auto P_14=P_13.norm();
th(2) = acos((P_14*P_14 - a2*a2 - a3*a3)/(2 * a2 * a3));


//*** theta 2 e 4 ******
auto F_10 = AH(1,th).inverse();
auto F_65 = AH( 6,th).inverse();
auto F_54 = AH( 5,th).inverse();
auto F_14 = (F_10 * desired_pos) * F_65 * F_54;
auto F_13 = F_14 * p3 - p2;
th(1) = -atan2(F_13(1), -F_13(0)) + asin(a3* sin(th(2))/F_13.
    norm());
```

```
    auto F_32 = AH( 3,th).inverse();
    auto F_21 = AH( 2,th).inverse();
    auto F_34 = F_32 * F_21 * F_14;
    th(3) = atan2(F_34(1,0), F_34(0,0));
    if (th(3) > 0)
        th(3)= th(3) - 2*M_PI;

    for (int i=0;i<6;i++){
        pos[i]=th(i);
    }
    //pos[5]=pos[5]+1.5708;
    move_arm_planner(pos,movement_number,startTime,endTime);
    return true;

}


geometry_msgs::PoseStamped UR5::get_end_position(){
    VectorXf th(6);
    for (int i=0; i < 6; i++){
        th(i)=get_state().position[i];
    }

    auto A_1=AH( 1,th);
    auto A_2=AH( 2,th);
    auto A_3=AH( 3,th);
    auto A_4=AH( 4,th);
    auto A_5=AH( 5,th);
    auto A_6=AH( 6,th);

    Matrix4f T_06=A_1*A_2*A_3*A_4*A_5*A_6;

    geometry_msgs::PoseStamped p;

    auto angles=rotationToEuler(T_06);
    p.pose.position.x= T_06(0,3);
    p.pose.position.y= T_06(1,3);
    p.pose.position.z= T_06(2,3);
    p.pose.orientation.x=angles(0);
    p.pose.orientation.y=angles(1);
    p.pose.orientation.z=angles(2);

    return p;

}

geometry_msgs::Point UR5::homotransform(geometry_msgs::Point point){
```

```
    auto pos=get_end_position();

    float thetax=pos.pose.orientation.x;
    float thetay=pos.pose.orientation.y;
    float thetaz=pos.pose.orientation.z;
    float x=pos.pose.position.x;
    float y=pos.pose.position.y;
    float z=pos.pose.position.z;

    Matrix4f trans;
    trans=radianToRotation(x,y,z,thetax,thetay,thetaz);

    Vector4f campos;
    campos << point.x ,point.y , point.z, 1;

    Vector4f final_pos;
    final_pos= trans * campos;
    geometry_msgs::Point solution;
    solution.x= final_pos(0);
    solution.y= final_pos(1);
    solution.z= final_pos(2);
    return solution;

}

geometry_msgs::PoseStamped UR5::homotransform(geometry_msgs::Point
   point,float ax,float ay,float az){
    auto pos=get_end_position();

    float thetax=pos.pose.orientation.x;
    float thetay=pos.pose.orientation.y;
    float thetaz=pos.pose.orientation.z;
    float x=pos.pose.position.x;
    float y=pos.pose.position.y;
    float z=pos.pose.position.z;

    Matrix4f trans;
    trans=radianToRotation(x,y,z,thetax,thetay,thetaz);

    Matrix4f campos;
    campos=radianToRotation(point.x,point.y,point.z,ax,ay,az);

    Matrix4f final_pos;
    final_pos= trans * campos;

    auto angles=rotationToEuler(final_pos);
```

```
    geometry_msgs::PoseStamped solution;
    solution.pose.position.x= final_pos(0,3);
    solution.pose.position.y= final_pos(1,3);
    solution.pose.position.z= final_pos(2,3);

    solution.pose.orientation.x=angles(0);
    solution.pose.orientation.y=angles(1);
    solution.pose.orientation.z=angles(2);

    return solution;
}
```

Kinect Class

```
#include "Kinect.hpp"


std::string Kinect::type2str() {
  int type= img.type();
  std::string r;

  uchar depth = type & CV_MAT_DEPTH_MASK;
  uchar chans = 1 + (type >> CV_CN_SHIFT);

  switch ( depth ) {
    case CV_8U:  r = "8U"; break;
    case CV_8S:  r = "8S"; break;
    case CV_16U: r = "16U"; break;
    case CV_16S: r = "16S"; break;
    case CV_32S: r = "32S"; break;
    case CV_32F: r = "32F"; break;
    case CV_64F: r = "64F"; break;
    default:     r = "User"; break;
  }

  r += "C";
  r += (chans+'0');

  return r;
}


void Kinect::set_image(const sensor_msgs::ImageConstPtr& msg){
        cv_bridge::CvImageConstPtr cv_ptr;
        try
        {
                cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                   image_encodings::TYPE_8UC3);
                img = cv_ptr->image;
        }
        catch (cv_bridge::Exception& e)
        {
                ROS_ERROR("cv_bridge exception: %s", e.what());
                return;
        }
}

void Kinect::set_image_depth(const sensor_msgs::ImageConstPtr& msg){
        cv_bridge::CvImagePtr cv_ptr;
```

```
        try
        {
                cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                   image_encodings::TYPE_16UC1);//now cv_ptr is the
                   matrix, do not forget "TYPE_" before "16UC1"
                img_depth=cv_ptr->image;
        }
        catch (cv_bridge::Exception& e)
        {
                ROS_ERROR("cv_bridge exception: %s", e.what());
                return;
        }

}


void Kinect::set_point(const sensor_msgs::PointCloud2Ptr& msg){
        cloud.header=msg->header;
        cloud.height=msg->height;
        cloud.width=msg->width;
        cloud.is_bigendian=msg->is_bigendian;
        cloud.point_step=msg->point_step;
        cloud.row_step=msg->row_step;
        cloud.is_dense=msg->is_dense;
        int sizef=msg->fields.size();
        int sized=msg->data.size();

        cloud.fields.resize(sizef);
        cloud.data.resize(sized);

        for (int i=0; i < sizef; i++){
                cloud.fields[i].name = msg->fields[i].name;
                cloud.fields[i].offset= msg->fields[i].offset;
                cloud.fields[i].datatype= msg->fields[i].datatype;
                cloud.fields[i].count= msg->fields[i].count;
        }

        for (int i=0; i < sized; i++)
                cloud.data[i] = msg->data[i];
    cloud_filled=true;
}


void Kinect::show_image(){

        if(img.data)                                    // Check for
           invalid input
```

```
                {
                        imshow( "Kinect RGB Image",img );
                        waitKey(5);
                }

}

void Kinect::show_image_depth(){

        if(img.data)                                    // Check for
           invalid input
                {
                        imshow( "Kinect Depth Image",img_depth );
                        waitKey(5);
                }

}
```

Gripper Class

```
#include "rfinger.hpp"

RFinger::RFinger(ros::NodeHandle n_){
  n=n_;
  f=n.advertise<robotiq_2f_gripper_control::
      Robotiq2FGripper_robot_output>("/Robotiq2FGripperRobotOutput",
      1);
}

void RFinger::set_state(const robotiq_2f_gripper_control::
   Robotiq2FGripper_robot_input::ConstPtr& now){
  is_subbed=true;
  status.gACT=now->gACT;
  status.gGTO=now->gGTO;
  status.gSTA=now->gSTA;
  status.gOBJ=now->gOBJ;
  status.gFLT=now->gFLT;
  status.gPR=now->gPR;
  status.gPO=now->gPO;
  status.gCU=now->gCU;

}

void RFinger::init(){
  if (status.gACT == 0){
    data.rACT=1;
    data.rGTO=1;
    data.rATR=0;
    data.rPR=0;
    data.rSP=255;
    data.rFR=150;
    speed=255;
    force=150;
    f.publish(data);
    ROS_INFO("Gripper initialized");
  }else if(status.gACT == 1){
    ROS_WARN("Gripper already on");
  }
}

void RFinger::reset(){
  if(status.gACT == 1){
    data.rACT=0;
    data.rGTO=0;
    data.rATR=0;
```

```
    data.rPR=0;
    data.rSP=0;
    data.rFR=0;
    f.publish(data);
    ROS_INFO("Gripper Restarted");
  }else{
    ROS_WARN("Gripper already off");
  }
}

void RFinger::close(){
  if(status.gACT == 1){
    data.rACT=1;
    data.rGTO=1;
    data.rATR=0;
    data.rPR=255;
    data.rSP=speed;
    data.rFR=force;
    f.publish(data);
    ROS_INFO("Gripper Closed");
  }else{
    ROS_ERROR("Init the Gripper");
  }
}

void RFinger::open(){
  if(status.gACT == 1){
    data.rACT=1;
    data.rGTO=1;
    data.rATR=0;
    data.rPR=0;
    data.rSP=speed;
    data.rFR=force;
    f.publish(data);
    ROS_INFO("Gripper Open");
  }else{
    ROS_ERROR("Init the Gripper");
  }
}

void RFinger::set_speed(int speedo){
  speed=speedo;
}

void RFinger::set_force(int forceo){
  force=forceo;
}
```

```
void RFinger::set_position(int position){
  if(status.gACT == 1){
    data.rACT=1;
    data.rGTO=1;
    data.rATR=0;
    data.rPR=position;
    data.rSP=speed;
    data.rFR=force;
    f.publish(data);
    ROS_INFO("Gripper In Position");
  }else{
    ROS_ERROR("Init the Gripper");
  }
}

robotiq_2f_gripper_control::Robotiq2FGripper_robot_input  RFinger::
   get_status(){
  return status;
}

int RFinger::get_position(){
  return status.gPR;
}

int RFinger::get_speed(){
  return speed;
}

int RFinger::get_force(){
  return force;
}

int RFinger::has_obj(){
  return status.gOBJ;
}
```

Object Detection Class

```
#include "ObjDetection.hpp"

float ObjDetec::ox=0;
float ObjDetec::oy=0;
float ObjDetec::oz=0;
float ObjDetec::angle=0;

geometry_msgs::Point ObjDetec::pointsToPose(
geometry_msgs::Point p1,geometry_msgs::Point p2){
    geometry_msgs::Point ponto;
    float xp=p1.x;
    float yp=p1.y;
    float zp=p1.z;
    float xc=p2.x;
    float yc=p2.y;
    float zc=p2.z;
    //const float pid= 180/M_PI;
    ponto.x = atan(sqrt(pow((xp-xc),2)+pow((yp-yc),2) )/(zp-zc));
    ponto.y = atan(sqrt( pow((zp-zc),2)+pow((yp-yc),2))/(xp-xc));
    ponto.z = atan(sqrt(pow((xp-xc),2)+pow((zp-zc),2))/(yp-yc));
    /*
    cout << "Angulo x:" << ponto.x*pid << endl;
    cout << "Angulo y:" << ponto.y*pid << endl;
    cout << "Angulo z:" << ponto.z*pid << endl;
    */
    return ponto;
}


geometry_msgs::Point ObjDetec::pixelTo3DPoint(
const sensor_msgs::PointCloud2 pCloud, const int u, const int v)
{
    geometry_msgs::Point p;
    int arrayPosition = v*pCloud.row_step + u*pCloud.point_step;

    int arrayPosX = arrayPosition + pCloud.fields[0].offset;
    int arrayPosY = arrayPosition + pCloud.fields[1].offset;
    int arrayPosZ = arrayPosition + pCloud.fields[2].offset;

    float X = 0.0;
    float Y = 0.0;
    float Z = 0.0;

    memcpy(&X, &pCloud.data[arrayPosX], sizeof(float));
    memcpy(&Y, &pCloud.data[arrayPosY], sizeof(float));
```

```
    memcpy(&Z, &pCloud.data[arrayPosZ], sizeof(float));

    p.x = X;
    p.y = Y;
    p.z = Z;

    return p;
}

Point ObjDetec::objDetec(Kinect cam,string image_path){

    if (cam.get_image().data){
    Mat img=cam.get_image();

    //Aplica Orb na imagem
    std::vector<cv::KeyPoint> keypoints_object, keypoints_scene;
    cv::Mat descriptors_object, descriptors_scene;
    double ti=(double)getTickCount();
    Ptr<Feature2D> detector= ORB::create(2000, 1.2f, 8, 10 , 0, 2,
       ORB::HARRIS_SCORE, 10);
    Ptr<Feature2D> extractor= ORB::create() ;
    Mat out,img_scene;
    cvtColor(img, img_scene , cv::COLOR_BGR2GRAY);
    double t=(double)getTickCount();
    detector->detect(img_scene, keypoints_scene);
    extractor->compute(img_scene, keypoints_scene, descriptors_scene
       );
    t = ((double)getTickCount() - t)/getTickFrequency();
    cout << "Tempo de Processamento: " << t << endl;
    drawKeypoints(img_scene, keypoints_scene, out, Scalar::all(255))
       ;
    imshow("Cena",out);

    Mat img_object=imread(image_path, CV_LOAD_IMAGE_COLOR);
    cvtColor(img_object, img_object , cv::COLOR_BGR2GRAY);
    detector->detect(img_object, keypoints_object);
    extractor->compute(img_object, keypoints_object,
       descriptors_object);
    Mat out2;
    drawKeypoints(img_object, keypoints_object, out2, Scalar::all
       (255));
    imshow("Objeto",out2);



    //Compara as features
    Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("
```

```
    BruteForce-Hamming");
vector<cv::DMatch> matches;
Mat img_matches;
if(!descriptors_object.empty() && !descriptors_scene.empty()) {
    double t1=(double)getTickCount();
    matcher->match (descriptors_object, descriptors_scene,
        matches);
    t1 = ((double)getTickCount() - t1)/getTickFrequency();
    cout << "Tempo de Processamento de Matches: " << t1 << endl;

    double max_dist = 0; double min_dist = 100;

    //-- Quick calculation of max and min idstance between
        keypoints
    for( int i = 0; i < descriptors_object.rows; i++)
    { double dist = matches[i].distance;
      if( dist < min_dist ) min_dist = dist;
      if( dist > max_dist ) max_dist = dist;
    }

    //-- Draw only good matches (i.e. whose distance is less
        than 3*min_dist)
    std::vector< cv::DMatch >good_matches;

    for( int i = 0; i < descriptors_object.rows; i++ )

    { if( matches[i].distance < (max_dist/1.6) )
      { good_matches.push_back( matches[i]); }
    }

    cv::drawMatches(img_object, keypoints_object, img_scene,
        keypoints_scene, \
          good_matches, img_matches, Scalar::all(-1), Scalar::
              all(-1),
          std::vector<char>(), cv::DrawMatchesFlags::
              NOT_DRAW_SINGLE_POINTS);

    //-- localize the object
    std::vector<Point2f> obj;
    std::vector<Point2f> scene;

    for( size_t i = 0; i < good_matches.size(); i++) {
      //-- get the keypoints from the good matches
      obj.push_back( keypoints_object[ good_matches[i].
          queryIdx ].pt );
      scene.push_back( keypoints_scene[ good_matches[i].
          trainIdx ].pt );
```

```
}
if( !obj.empty() && !scene.empty() && good_matches.size() >=
    4) {
    double t2=(double)getTickCount();
    Mat H = findHomography( obj, scene, RANSAC );
    t2 = ((double)getTickCount() - t2)/getTickFrequency();
    cout << "Tempo de Processamento do RANSAC: " << t2 <<
       endl;

    //-- get the corners from the object to be detected
    std::vector<Point2f> obj_corners(4);
    obj_corners[0] = Point(0,0);
    obj_corners[1] = Point(img_object.cols,0);
    obj_corners[2] = Point(img_object.cols,img_object.rows);
    obj_corners[3] = Point(0,img_object.rows);

    std::vector<cv::Point2f> scene_corners(4);

    cv::perspectiveTransform( obj_corners, scene_corners, H)
       ;

    //-- Draw lines between the corners (the mapped object
       in the scene - image_2 )

    Point p1= scene_corners[0] + Point2f(img_object.cols, 0)
       ;
    Point p2= scene_corners[1] + Point2f(img_object.cols, 0)
       ;
    Point p3= scene_corners[2] + Point2f(img_object.cols, 0)
       ;
    Point p4= scene_corners[3] + Point2f(img_object.cols, 0)
       ;

    line( img_matches, p1, p2, cv::Scalar(0,0,255), 4 );
    line( img_matches, p2, p3, cv::Scalar(0,255,0), 4 );
    line( img_matches, p3, p4, cv::Scalar(0,255,0), 4 );
    line( img_matches, p4, p1, cv::Scalar(0,255,0), 4 );

    ObjDetec::angle=atan2(p2.y - p1.y, p2.x - p1.x);
    cout << "Rotacao: " << ObjDetec::angle << endl;
    /*
    Point t,t2;
    t.x= p1.x - 5;
    t.y= p1.y;
    t2.x= p2.x + 5;
    t2.y= p2.y;
```

```
            auto c1=ObjDetec::pixelTo3DPoint(cam.get_cloud(),p1.x,p1
                .y);
            auto c2=ObjDetec::pixelTo3DPoint(cam.get_cloud(),p2.x,p2
                .y);

            cout << "C1: " << c1 << endl;
            cout << "C2: " << c2 << endl;

            pointsToPose(c2,c1);
            */

            auto x=(p1.x+p2.x+p3.x+p4.x)/4 ;
            auto y=(p1.y+p2.y+p3.y+p4.y)/4;

            Point center(x,y);

            circle( img_matches, center, 4, cv::Scalar(255,255,255),
                -1, 8, 0 );
            imshow("match result", img_matches );
            x= x - img_object.cols;
            center.x=x;
            ti = ((double)getTickCount() - ti)/getTickFrequency();
            cout << "Tempo de Processamento Total: " << ti << endl;
            return center;
        }

        }

    }
    Point center_void(0,0);
    return center_void;
}



TagTestOptions ObjDetec::parse_options(int argc, char** argv) {
  TagTestOptions opts;
  const char* options_str = "hdtRvxoDS:s:a:m:V:N:brnf:e:";
  int c;
  while ((c = getopt(argc, argv, options_str)) != -1) {
    switch (c) {
      // Reminder: add new options to 'options_str' above and
         print_usage()!
      case 'd': opts.show_debug_info = true; break;
      case 't': opts.show_timing = true; break;
      case 'R': opts.show_results = true; break;
      case 'v': opts.be_verbose = true; break;
```

```
      case 'x': opts.no_images = true; break;
      case 'o': opts.generate_output_files = true; break;
      case 'D': opts.params.segDecimate = true; break;
      case 'S': opts.params.sigma = atof(optarg); break;
      case 's': opts.params.segSigma = atof(optarg); break;
      case 'a': opts.params.thetaThresh = atof(optarg); break;
      case 'm': opts.params.magThresh = atof(optarg); break;
      case 'V': opts.params.adaptiveThresholdValue = atof(optarg);
         break;
      case 'N': opts.params.adaptiveThresholdRadius = atoi(optarg);
         break;
      case 'b': opts.params.refineBad = true; break;
      case 'r': opts.params.refineQuads = true; break;
      case 'n': opts.params.newQuadAlgorithm = true; break;
      case 'f': opts.family_str = optarg; break;
      case 'e': opts.error_fraction = atof(optarg); break;
      default:
        fprintf(stderr, "\n");
        exit(1);
    }
  }
  opts.params.adaptiveThresholdRadius += (opts.params.
     adaptiveThresholdRadius+1) % 2;
  if (opts.be_verbose) {
    opts.show_debug_info = opts.show_timing = opts.show_results =
       true;
  }
  return opts;
}

Point ObjDetec::aprilDetect(Kinect cam, unsigned int id, bool show){
  const std::string win = "Tag test";
  Point zero(0,0);

  TagTestOptions opts = parse_options(0, 0);

  TagFamily family(opts.family_str);

  if (opts.error_fraction >= 0 && opts.error_fraction < 1) {
    family.setErrorRecoveryFraction(opts.error_fraction);
  }

  TagDetector detector(family, opts.params);
  detector.debug = opts.show_debug_info;
  detector.debugWindowName = opts.generate_output_files ? "" : win;
  if (opts.params.segDecimate && opts.be_verbose) {
    std::cout << "Will decimate for segmentation!\n";
```

```
  }

  TagDetectionArray detections;
  Mat src = cam.get_image();


  if (src.empty()) { return zero; }

  while (std::max(src.rows, src.cols) > 800) {
    cv::Mat tmp;
    cv::resize(src, tmp, cv::Size(0,0), 0.5, 0.5);
    src = tmp;
  }
  cv::Point2d opticalCenter(0.5*src.rows, 0.5*src.cols);

  clock_t start = clock();
  detector.process(src, opticalCenter, detections);
  clock_t end = clock();

  if (opts.show_results) {
    if (opts.show_debug_info) std::cout << "\n";
    std::cout << "Got " << detections.size() << " detections in "
              << double(end-start)/CLOCKS_PER_SEC << " seconds.\n";
    for (size_t i=0; i<detections.size(); ++i) {
      const TagDetection& d = detections[i];
      std::cout << " - Detection: id = " << d.id << ", "
                << "code = " << d.code << ", "
                << "position = " << d.cxy << ", "
                << "rotation = " << d.rotation << "\n";
      cv::Point center(d.cxy.x,d.cxy.y);
      cv::circle( src, center, 4, cv::Scalar(0,0,255), -1, 8, 0 );
    }
  }
  if (!opts.no_images) {
    cv::Mat img = family.superimposeDetections(src, detections);
    labelAndWaitForKey(win, "Detected", img, ScaleNone, true);
  }

   for (size_t i=0; i<detections.size(); ++i) {
      const TagDetection& d = detections[i];
      std::cout << " - Detection: id = " << d.id << ", "
                << "code = " << d.code << ", "
                << "position = " << d.cxy << ", "
                << "rotation = " << d.rotation << "\n";
      if (d.id == id){
        cv::Point center(d.cxy.x,d.cxy.y);
        cv::circle( src, d.p[0], 4, cv::Scalar(255,0,0), -1, 8, 0 );
```

```
        cv::circle( src, d.p[1], 4, cv::Scalar(0,255,0), -1, 8, 0 );
        cv::circle( src, d.p[2], 4, cv::Scalar(0,0,255), -1, 8, 0 );
        cv::circle( src, d.p[3], 4, cv::Scalar(255,255,255), -1, 8,
           0 );
        cv::circle( src, center, 4, cv::Scalar(0,0,0), -1, 8, 0 );
        ObjDetec::angle=atan2(d.p[1].y - d.p[0].y, d.p[1].x - d.p
           [0].x);

        if (cam.cloud_filled){
            auto c1=ObjDetec::pixelTo3DPoint(cam.get_cloud(),d.p[0].
               x,d.p[0].y);
            auto c2=ObjDetec::pixelTo3DPoint(cam.get_cloud(),d.p[1].
               x,d.p[1].y);
            auto c3=ObjDetec::pixelTo3DPoint(cam.get_cloud(),d.p[2].
               x,d.p[2].y);

            //cout << "C1: " << c1 << endl;
            //cout << "C2: " << c2 << endl;
            //cout << "C3: " << c3 << endl;


            auto pose1=pointsToPose(c2,c1);

            auto pose2=pointsToPose(c2,c3);
            if (pose1.x > 0 && pose1.x <= 1.57){
                ObjDetec::ox= 1.57 - pose1.x;
            }else if (pose1.x < 0 && pose1.x >= -1.57){
                ObjDetec::ox= -1.57 - pose1.x;
            }

             ObjDetec::oy=ObjDetec::angle;
             ObjDetec::oz=pose2.z;
        }


        cout << "Rotacao: " << ObjDetec::angle*180/M_PI << endl;
        putText(src, to_string(d.id) , cvPoint(center.x,center.y),
           FONT_HERSHEY_DUPLEX, 0.8, cvScalar(0,0,250), 1, CV_AA);
        return center;
      }

    }

  if (opts.show_timing) detector.reportTimers();
  return zero;
}
```

```
void ObjDetec::multAprilDetect(Kinect cam, unsigned int id[], Point
   output[], int size, bool show){

    const std::string win = "Tag test";
    Point zero(0,0);

    for (int i=0;i < size;i++){
        output[i]=zero;
    }

    TagTestOptions opts = parse_options(0, 0);

    TagFamily family(opts.family_str);

    if (opts.error_fraction >= 0 && opts.error_fraction < 1) {
      family.setErrorRecoveryFraction(opts.error_fraction);
    }

    TagDetector detector(family, opts.params);
    detector.debug = opts.show_debug_info;
    detector.debugWindowName = opts.generate_output_files ? "" : win
       ;
    if (opts.params.segDecimate && opts.be_verbose) {
      std::cout << "Will decimate for segmentation!\n";
    }

    TagDetectionArray detections;
    Mat src = cam.get_image();

    if (!src.empty()) {

        while (std::max(src.rows, src.cols) > 800) {
          cv::Mat tmp;
          cv::resize(src, tmp, cv::Size(0,0), 0.5, 0.5);
          src = tmp;
        }
        cv::Point2d opticalCenter(0.5*src.rows, 0.5*src.cols);

        clock_t start = clock();
        detector.process(src, opticalCenter, detections);
        clock_t end = clock();

        if (opts.show_results) {
          if (opts.show_debug_info) std::cout << "\n";
          std::cout << "Got " << detections.size() << " detections
             in "
                    << double(end-start)/CLOCKS_PER_SEC << " seconds
```

```
                .\n";
  for (size_t i=0; i<detections.size(); ++i) {
    const TagDetection& d = detections[i];
    std::cout << " - Detection: id = " << d.id << ", "
              << "code = " << d.code << ", "
              << "position = " << d.cxy << ", "
              << "rotation = " << d.rotation << "\n";
    cv::Point center(d.cxy.x,d.cxy.y);
    cv::circle( src, center, 4, cv::Scalar(0,0,255), -1, 8,
       0 );
  }
}
if (!opts.no_images) {
  cv::Mat img = family.superimposeDetections(src, detections
     );
  labelAndWaitForKey(win, "Detected", img, ScaleNone, true);
}

 for (size_t i=0; i<detections.size(); ++i) {
    const TagDetection& d = detections[i];
    std::cout << " - Detection: id = " << d.id << ", "
              << "code = " << d.code << ", "
              << "position = " << d.cxy << ", "
              << "rotation = " << d.rotation << "\n";
    for (int j=0;j < size;j++){
        if (d.id == id[j]){
          cv::Point center(d.cxy.x,d.cxy.y);
          cv::circle( src, d.p[0], 4, cv::Scalar(255,0,0),
             -1, 8, 0 );
          cv::circle( src, d.p[1], 4, cv::Scalar(255,0,0),
             -1, 8, 0 );
          cv::circle( src, d.p[2], 4, cv::Scalar(255,0,0),
             -1, 8, 0 );
          cv::circle( src, d.p[3], 4, cv::Scalar(255,0,0),
             -1, 8, 0 );
          cv::circle( src, center, 4, cv::Scalar(0,0,255),
             -1, 8, 0 );
          putText(src, to_string(d.id) , cvPoint(center.x,
             center.y),FONT_HERSHEY_DUPLEX, 0.8, cvScalar
             (0,0,250), 1, CV_AA);
          output[j]=center;
          //return center;
        }
    }


  }
```

```
        }
}
```

ROS Action Inicio

```
#include <ros/ros.h>
#include <actionlib/server/simple_action_server.h>
#include <inicio_skill_server/inicio_skill_server.h>
#include <fstream>
#include "ur5_arm.hpp"
#include "rfinger.hpp"

void saveData(sensor_msgs::JointState state,geometry_msgs::
    PoseStamped p, int gripper_pos,int obj,double time){


        ofstream csv;
        csv.open("/home/fastenufba/logs UR/dados.csv", std::ios_base
            ::app);

        csv << time << "," <<  0 <<  "," << state.position[0] << ","
           <<  state.position[1] << "," << state.position[2] << ","
           <<  state.position[3] << "," << state.position[4] << ","
           <<  state.position[5] << ","
               << state.velocity[0] << "," <<  state.velocity[1] <<
                  "," << state.velocity[2] << "," <<  state.velocity
                  [3] << "," << state.velocity[4] << "," <<  state.
                  velocity[5] << ","
               << p.pose.position.x << "," << p.pose.position.y <<
                  "," << p.pose.position.z << "," << p.pose.
                  orientation.x << "," << p.pose.orientation.y << ","
                    << p.pose.orientation.z << ","
               << 0 << "," << 0 << "," << 0 << "," << 0 << ","
                  << 0 << "," << 0 << ","
               << 0 << "," << 0 << "," << 0 << "," << gripper_pos
                  << "," << obj << "," << endl;

        cout << "Dados Salvos" << endl;

}

InicioSkill::InicioSkill(std::string name) :
  as_(nh_, name, boost::bind(&InicioSkill::executeCB, this, _1),
     false),
  action_name_(name)
  {
    as_.start();
  }

InicioSkill::~InicioSkill()
```

```
{
}

void InicioSkill::executeCB(const inicio_skill_msgs::
    InicioSkillGoalConstPtr &goal)
{
     ros::NodeHandle n;
     UR5 ur(n,11);
     RFinger g(n);
     float actual_pos[6];
     float x_init=goal->x;
     float y_init=goal->y;
     float z_init=goal->z;
     float ox=goal->ox;
     float oy=goal->oy;
     float oz=goal->oz;


     bool is_pub=false,is_init=false,is_saved=false;
     int tries=0;

     ros::Subscriber gripper = n.subscribe("/
        Robotiq2FGripperRobotInput", 1, &RFinger::set_state,&g);
     ros::Subscriber sub = n.subscribe("/joint_states", 1000, &UR5::
        set_state,&ur);

     ros::Time start_time;

     ros::Duration delta_t;
     double delta_t_sec=0;

     ros::Rate loop_rate(10);
     while(n.ok()){
          if (g.get_status().gACT == 0 && g.is_subbed){
               g.init();
               is_init=true;
          }else if (g.get_status().gACT == 0 && is_init){
               tries++;
          }

          if (tries == 20){

               ROS_ERROR("Could not initiate the gripper");
               set_aborted();
               break;
```

```
}


if (g.get_status().gACT ==1 && !is_pub){
    g.set_position(127);

    for(int i=0; i < 6;i++){
        actual_pos[i]= ur.get_state().position[i];
    }
    start_time = ros::Time::now();

    ur.move_arm(actual_pos,0,0);
    ur.ik_move(x_init,y_init,z_init,ox,oy,oz,1,0,5);
    ur.publish();
    is_pub=true;
}

delta_t= ros::Time::now() - start_time;
delta_t_sec = delta_t.toSec();

if(delta_t_sec > 6 && delta_t_sec < 10){
    ROS_INFO("Movement Complet");
    set_succeeded();
    break;
}

if (!is_saved){
    ofstream csv;
    csv.open("/home/fastenufba/logs UR/dados.csv", std::
       ios_base::app);
    csv << "Time" << "," << "Skill" << "," << "Joint1" <<
       "," << "Joint2" << "," << "Joint3" << "," << "Joint4"
        << "," << "Joint5" << "," << "Joint6" << ","
           << "Velocity1" << "," << "Velocity2" << ","<< "
               Velocity3" << ","<< "Velocity4" << ","<< "
               Velocity5" << ","<< "Velocity6" << ","
           << "x" << "," << "y" << "," << "z" << "," << "
               ox" << "," << "oy" << "," << "oz" << ","
              << "tx" << "," << "ty" << "," << "tz" << ","
                   << "cx" << "," << "cy" << "," << "cz" <<
                  ","
                  << "ax" << "," << "ay" << "," << "az" <<
                      "," << "Gripper Pos" << "," << "Gripper
                       Status" << endl;
    is_saved=true;
    csv.close();
}
```

```
        if (delta_t_sec > 0 && delta_t_sec < 6){
            saveData(ur.get_state(),ur.get_end_position(),g.
                get_position(),g.has_obj(),delta_t_sec);
        }

        ros::spinOnce();
        loop_rate.sleep();
    }
}

void InicioSkill::set_succeeded(std::string outcome)
{
  result_.percentage = 100;
  result_.skillStatus = action_name_.c_str();
  result_.skillStatus += ": Succeeded";
  result_.outcome = outcome;
  ROS_INFO("%s: Succeeded", action_name_.c_str());
  as_.setSucceeded(result_);
}

void InicioSkill::set_aborted(std::string outcome)
{
  result_.percentage = 0;
  result_.skillStatus = action_name_.c_str();
  result_.skillStatus += ": Aborted";
  result_.outcome = outcome;
  ROS_INFO("%s: Aborted", action_name_.c_str());
  as_.setAborted(result_);
}
void InicioSkill::feedback(float percentage)
{
  feedback_.percentage = percentage;
  feedback_.skillStatus = action_name_.c_str();
  feedback_.skillStatus += " Executing";
  ROS_INFO("%s: Executing. Percentage: %f%%.", action_name_.c_str(),
      percentage);
  as_.publishFeedback(feedback_);
}

bool InicioSkill::check_preemption()
{
  if (as_.isPreemptRequested() || !ros::ok()){
    result_.percentage = 0;
    result_.skillStatus = action_name_.c_str();
    result_.skillStatus += ": Preempted";
    result_.outcome = "preempted";
```

```
    ROS_INFO("%s: Preempted", action_name_.c_str());
    as_.setPreempted(result_);
    return true;
  }
  else{
    return false;
  }
}
```

Detect ROS Action

```
#include <ros/ros.h>
#include <actionlib/server/simple_action_server.h>
#include <detect_skill_server/detect_skill_server.h>
#include "Kinect.hpp"
#include "ObjDetection.hpp"
#include "ur5_arm.hpp"



DetectSkill::DetectSkill(std::string name) :
  as_(nh_, name, boost::bind(&DetectSkill::executeCB, this, _1),
     false),
  action_name_(name)
  {
    as_.start();
  }

DetectSkill::~DetectSkill()
{

}

void DetectSkill::executeCB(const detect_skill_msgs::
   DetectSkillGoalConstPtr &goal)
{
    ros::NodeHandle n;
    Kinect cam;
    UR5 ur(n,11);
    int tries=0;

    ros::Subscriber camera = n.subscribe("/camera/color/image_raw",
       1, &Kinect::set_image,&cam);
    ros::Subscriber depth = n.subscribe("/camera/depth/image_raw",
       1, &Kinect::set_image_depth,&cam);
    ros::Subscriber cloud= n.subscribe("/camera/depth_registered/
       points", 1000, &Kinect::set_point,&cam);

    ros::Rate loop_rate(10);
    while(n.ok()){

        auto peca= ObjDetec::aprilDetect(cam,goal->id,false);

        if(peca.x > 0 && peca.y > 0){
            ROS_INFO("Found the tag");
            set_succeeded();
```

```
            break;
        }else{
            tries++;
        }

        if(tries > 15){
            ROS_ERROR("Tag not found");
            set_aborted();
            break;
        }


        ros::spinOnce();
        loop_rate.sleep();
    }
}

void DetectSkill::set_succeeded(std::string outcome)
{
  result_.percentage = 100;
  result_.skillStatus = action_name_.c_str();
  result_.skillStatus += ": Succeeded";
  result_.outcome = outcome;
  ROS_INFO("%s: Succeeded", action_name_.c_str());
  as_.setSucceeded(result_);
}

void DetectSkill::set_aborted(std::string outcome)
{
  result_.percentage = 0;
  result_.skillStatus = action_name_.c_str();
  result_.skillStatus += ": Aborted";
  result_.outcome = outcome;
  ROS_INFO("%s: Aborted", action_name_.c_str());
  as_.setAborted(result_);
}
void DetectSkill::feedback(float percentage)
{
  feedback_.percentage = percentage;
  feedback_.skillStatus = action_name_.c_str();
  feedback_.skillStatus += " Executing";
  ROS_INFO("%s: Executing. Percentage: %f%%.", action_name_.c_str(),
      percentage);
  as_.publishFeedback(feedback_);
}

bool DetectSkill::check_preemption()
```

```
{
  if (as_.isPreemptRequested() || !ros::ok()){
    result_.percentage = 0;
    result_.skillStatus = action_name_.c_str();
    result_.skillStatus += ": Preempted";
    result_.outcome = "preempted";
    ROS_INFO("%s: Preempted", action_name_.c_str());
    as_.setPreempted(result_);
    return true;
  }
  else{
    return false;
  }
}
```

ROS Action Pick

```
#include <ros/ros.h>
#include <actionlib/server/simple_action_server.h>
#include <pick_skill_server/pick_skill_server.h>
#include <fstream>
#include "rfinger.hpp"
#include "Kinect.hpp"
#include "ObjDetection.hpp"
#include "ur5_arm.hpp"

void saveData(sensor_msgs::JointState state,geometry_msgs::
    PoseStamped p, int gripper_pos,int obj,double time,geometry_msgs
    ::Point pos_obj,geometry_msgs::Point pos_cam,float ax,float ay,
    float az){
     ofstream csv;
     csv.open("/home/fastenufba/logs UR/dados.csv", std::ios_base::
        app);

     csv << time << "," << 1 << "," << state.position[0] << "," <<
        state.position[1] << "," << state.position[2] << "," <<
        state.position[3] << "," << state.position[4] << "," <<
        state.position[5] << ","
            << state.velocity[0] << "," << state.velocity[1] << ","
                << state.velocity[2] << "," << state.velocity[3] <<
                "," << state.velocity[4] << "," << state.velocity[5]
                << ","
            << p.pose.position.x << "," << p.pose.position.y << ","
               << p.pose.position.z << "," << p.pose.orientation.x <<
                "," << p.pose.orientation.y << "," << p.pose.
               orientation.z << ","
            << pos_obj.x << "," << pos_obj.y << "," << pos_obj.z
               << "," << pos_cam.x << "," << pos_cam.y << "," <<
               pos_cam.z << ","
            << ax << "," << ay << "," << az << "," << gripper_pos
               << "," << obj << "," << endl;

     cout << "Dados Salvos" << endl;
}

PickSkill::PickSkill(std::string name) :
  as_(nh_, name, boost::bind(&PickSkill::executeCB, this, _1), false
     ),
  action_name_(name)
  {
    as_.start();
  }
```

```
PickSkill::~PickSkill()
{
}

void PickSkill::executeCB(const pick_skill_msgs::
    PickSkillGoalConstPtr &goal)
{
    ros::NodeHandle n;
    Kinect cam;
    UR5 ur(n,11);
    bool is_pub=false;
    float actual_pos[6];
    RFinger g(n);
    geometry_msgs::Point po,pf;

    ros::Subscriber gripper = n.subscribe("/
        Robotiq2FGripperRobotInput", 1, &RFinger::set_state,&g);
    ros::Subscriber sub = n.subscribe("/joint_states", 1000, &UR5::
        set_state,&ur);
    ros::Subscriber camera = n.subscribe("/camera/color/image_raw",
        1, &Kinect::set_image,&cam);
    ros::Subscriber depth = n.subscribe("/camera/depth/image_raw",
        1, &Kinect::set_image_depth,&cam);
    ros::Subscriber cloud= n.subscribe("/camera/depth_registered/
        points", 1000, &Kinect::set_point,&cam);
    ros::Subscriber status= n.subscribe("/follow_joint_trajectory/
        status", 1000, &UR5::set_status,&ur);

    ros::Time start_time;

    ros::Duration delta_t;
    double delta_t_sec=0;

    ros::Rate loop_rate(10);
    while(n.ok()){
        auto peca= ObjDetec::aprilDetect(cam,goal->id,false);
        auto xo=peca.x;
        auto yo=peca.y;

        if(xo > 0 && yo > 0 && !is_pub && cam.cloud_filled){
            po=ObjDetec::pixelTo3DPoint(cam.get_cloud(),xo,yo);
            auto testenan=po.z;

            if(ur.get_state().position[1] != 0 && !isnan(testenan)){
                auto ox=ur.get_end_position().pose.orientation.x;
                auto oy=ur.get_end_position().pose.orientation.y;
```

```
                auto oz=ur.get_end_position().pose.orientation.z;

                po.x=po.x - 0.03263;
                po.y=po.y - 0.044;
                po.z=po.z - 0.327004;
                pf=ur.homotransform(po);

                for(int i=0; i < 6;i++){
                    actual_pos[i]= ur.get_state().position[i];
                }

                ur.move_arm(actual_pos,0,0);

                ur.ik_move(pf.x,pf.y,pf.z,ox,oy,oz,1,0,5);
                ur.publish();
                is_pub=true;
                start_time = ros::Time::now();

            }
        }
        delta_t= ros::Time::now() - start_time;
        delta_t_sec = delta_t.toSec();

        if (delta_t_sec > 0 && delta_t_sec < 10){
            saveData(ur.get_state(),ur.get_end_position(),g.
                get_position(),g.has_obj(),delta_t_sec,pf,po,ObjDetec
                ::ox,ObjDetec::oy,ObjDetec::oz);
        }

        if(delta_t_sec > 6 && delta_t_sec < 10 && ur.get_status() <=
            3){
            ROS_INFO("Movement Completed");
            set_succeeded();
            break;
        }else if(ur.get_status() >= 4 && ur.get_status() < 10 &&
            delta_t_sec > 6 && delta_t_sec < 10){
            ROS_ERROR("Arm Collided");
            set_aborted();
            break;
        }
    }
}

void PickSkill::set_succeeded(std::string outcome)
{
  result_.percentage = 100;
  result_.skillStatus = action_name_.c_str();
```

```
  result_.skillStatus += ": Succeeded";
  result_.outcome = outcome;
  ROS_INFO("%s: Succeeded", action_name_.c_str());
  as_.setSucceeded(result_);
}

void PickSkill::set_aborted(std::string outcome)
{
  result_.percentage = 0;
  result_.skillStatus = action_name_.c_str();
  result_.skillStatus += ": Aborted";
  result_.outcome = outcome;
  ROS_INFO("%s: Aborted", action_name_.c_str());
  as_.setAborted(result_);
}
void PickSkill::feedback(float percentage)
{
  feedback_.percentage = percentage;
  feedback_.skillStatus = action_name_.c_str();
  feedback_.skillStatus += " Executing";
  ROS_INFO("%s: Executing. Percentage: %f%%.", action_name_.c_str(),
      percentage);
  as_.publishFeedback(feedback_);
}

bool PickSkill::check_preemption()
{
  if (as_.isPreemptRequested() || !ros::ok()){
    result_.percentage = 0;
    result_.skillStatus = action_name_.c_str();
    result_.skillStatus += ": Preempted";
    result_.outcome = "preempted";
    ROS_INFO("%s: Preempted", action_name_.c_str());
    as_.setPreempted(result_);
    return true;
  }
  else{
    return false;
  }
}
```

ROS Action Grasp

```
#include <ros/ros.h>
#include <actionlib/server/simple_action_server.h>
#include <grasp_skill_server/grasp_skill_server.h>
#include "rfinger.hpp"
#include <iostream>
#include "ur5_arm.hpp"
#include <fstream>

using namespace std;

void saveData(sensor_msgs::JointState state,geometry_msgs::
   PoseStamped p, int gripper_pos,int obj,double time){


        ofstream csv;
        csv.open("/home/fastenufba/logs UR/dados.csv", std::ios_base
           ::app);

        csv << time << "," <<  2 <<  "," << state.position[0] << ","
           <<  state.position[1] << "," << state.position[2] << ","
           <<  state.position[3] << "," << state.position[4] << ","
           <<  state.position[5] << ","
               << state.velocity[0] << "," <<  state.velocity[1] <<
                  "," << state.velocity[2] << "," <<  state.velocity
                  [3] << "," << state.velocity[4] << "," <<  state.
                  velocity[5] << ","
               << p.pose.position.x << "," << p.pose.position.y <<
                  "," << p.pose.position.z << "," << p.pose.
                  orientation.x << "," << p.pose.orientation.y << ","
                    << p.pose.orientation.z << ","
               << 0 << "," << 0 << "," << 0 << "," << 0 << ","
                  << 0 << "," << 0 << ","
               << 0 << "," << 0 << "," << 0 << "," << gripper_pos
                  << "," << obj << "," << endl;

        cout << "Dados Salvos" << endl;

}

GraspSkill::GraspSkill(std::string name) :
  as_(nh_, name, boost::bind(&GraspSkill::executeCB, this, _1),
     false),
  action_name_(name)
  {
    as_.start();
```

```
  }

GraspSkill::~GraspSkill()
{
}

void GraspSkill::executeCB(const grasp_skill_msgs::
   GraspSkillGoalConstPtr &goal)
{
    ros::NodeHandle n;
    RFinger g(n);
    g.init();
    UR5 ur(n,11);
    bool is_init=false;
    int tries=0;

    ros::Subscriber gripper = n.subscribe("/
       Robotiq2FGripperRobotInput", 1, &RFinger::set_state,&g);


    ros::Time start_time;

    ros::Duration delta_t;
    double delta_t_sec=0;

    ros::Rate loop_rate(10);

    while(n.ok()){

        if(g.get_status().gACT ==0){
            tries++;
        }

        if(tries == 10){
            ROS_ERROR("Gripper not initialized");
            set_aborted();
            break;
        }

        if(g.get_status().gACT ==1 && !is_init){
            g.set_speed(25);
            g.set_force(25);
            g.close();
            is_init=true;
            start_time = ros::Time::now();
        }
```

```
        delta_t= ros::Time::now() - start_time;
        delta_t_sec = delta_t.toSec();
        cout << "Time passed: " << delta_t_sec <<endl;

        if (delta_t_sec > 2 && delta_t_sec < 10 && is_init){
            if (g.has_obj() == 2){
                cout << "Got the obj" << endl;
                set_succeeded();
                break;
            }else{
                ROS_ERROR("OBJ LOST");
                set_aborted();
                break;
            }
        }

        if (delta_t_sec > 0 && delta_t_sec < 6){
            saveData(ur.get_state(),ur.get_end_position(),g.
                get_position(),g.has_obj(),delta_t_sec);
        }

        ros::spinOnce();
        loop_rate.sleep();
    }
}

void GraspSkill::set_succeeded(std::string outcome)
{
  result_.percentage = 100;
  result_.skillStatus = action_name_.c_str();
  result_.skillStatus += ": Succeeded";
  result_.outcome = outcome;
  ROS_INFO("%s: Succeeded", action_name_.c_str());
  as_.setSucceeded(result_);
}

void GraspSkill::set_aborted(std::string outcome)
{
  result_.percentage = 0;
  result_.skillStatus = action_name_.c_str();
  result_.skillStatus += ": Aborted";
  result_.outcome = outcome;
  ROS_INFO("%s: Aborted", action_name_.c_str());
  as_.setAborted(result_);
}
void GraspSkill::feedback(float percentage)
{
```

```
  feedback_.percentage = percentage;
  feedback_.skillStatus = action_name_.c_str();
  feedback_.skillStatus += " Executing";
  ROS_INFO("%s: Executing. Percentage: %f%%.", action_name_.c_str(),
      percentage);
  as_.publishFeedback(feedback_);
}

bool GraspSkill::check_preemption()
{
  if (as_.isPreemptRequested() || !ros::ok()){
    result_.percentage = 0;
    result_.skillStatus = action_name_.c_str();
    result_.skillStatus += ": Preempted";
    result_.outcome = "preempted";
    ROS_INFO("%s: Preempted", action_name_.c_str());
    as_.setPreempted(result_);
    return true;
  }
  else{
    return false;
  }
}
```