Luís Henrique Veloso Mesquita

Computer Vision Solutions for Robots in the Context of Industry 4.0

Luís Henrique Veloso Mesquita

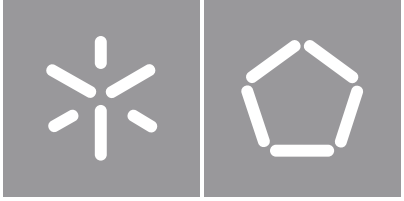
UMinho | 2021

Universidade do Minho
Escola de Engenharia

Luís Henrique Veloso Mesquita

Computer Vision Solutions for Robots in the Context of Industry 4.0

Outubro de 2021



Universidade do Minho
Escola de Engenharia

Luís Henrique Veloso Mesquita

Computer Vision Solutions for Robots in the Context of Industry 4.0

Dissertação de Mestrado
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores - Controlo, Automação e Robótica

Trabalho efetuado sob a orientação do
Doutor Luís Filipe Castro Freitas Louro

Outubro de 2021

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Acknowledgements

I would like to use this opportunity to thank and show my appreciation for everyone that influenced this dissertation and added their contribute to it.

To my advisor, Doctor Luís Louro, for the guidance, advices and support provided, especially during the setbacks and tougher times.

To professors Doctor Estela Bicho and Doctor Sérgio Monteiro for the interest shown in my academic journey, bringing me to the MARLab and providing me with the opportunity to be a better professional.

To Paulo Vicente for the support given with everything required in the MARLab.

To Duarte Teixeira, Toni Machado, Tiago Malheiro and Margarida Trigo for being valuable members of the team and guiding me in the autonomous stacker project.

To my parents and siblings for the support shown throughout this dissertation and my entire academic journey. Much of what I am today is direct result of their work and influence.

To João and Andreia for the experience, help and guidance during my time in this University.

To the friends with whom I was fortunate to share this journey with, during the best and worst times.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

Soluções de Visão por Computador para Robôs no contexto da Indústria 4.0

Esta dissertação teve como objetivo o desenvolvimento de soluções recorrendo a tecnologia e conceitos de visão por computador, com o propósito de serem aplicados tanto em processos de robótica móvel como de manipuladores robóticos.

É apresentado, num primeiro caso, o *software* de deteção e identificação de características de paletes para um veiculo empilhador autónomo com uma câmara Astra da Orbbec a bordo, como parte de um projeto em parceria com uma empresa. Esta funcionalidade permite assistir o sistema de navegação do veículo no processo de recolha de uma palete, onde o seu sistema de posicionamento geral não possuí resolução suficiente para os ajustes finos requeridos ao inserir os garfos na palete. O veículo foi testado em chão de fábrica e passou com sucesso os testes de segurança requeridos para poder operar em ambiente partilhado com operadores.

No segundo caso, o reconhecimento de objetos tem como objetivo que robôs colaborativos consigam realizar tarefas em parceria com humanos. Os robôs necessitam de ser capazes de obter não só a identificação dos objetos como também a sua postura (posição e orientação). Foi utilizada uma *framework* denominada Object Recognition Kitchen, da Williow Garage, como base para o processo de deteção. O cenário empregue utiliza o robô Sawyer da Rethink Robotics e uma câmara Microsoft Kinect.

Palavras-chave: reconhecimento de objetos, reconhecimento de paletes, robótica autónoma, robótica colaborativa, Sawyer Robot

Abstract

Computer Vision Solutions for Robots in the Context of Industry 4.0

The goal of this dissertation was to develop solutions using technology and concepts of Computer Vision, with the purpose of being applied in processes of both mobile and manipulation robotics.

It is presented software for the detection and identification of pallet features for an autonomous stacker with an on-board Orbbec Astra camera, as part of a project in association with an industrial company. This functionality is used for the steering of the vehicle during the process of picking a pallet, when the general positioning system does not have enough resolution for the fine tuning required to the insertion of the forks in the pallet. The vehicle was tested on the factory floor and successfully passed the safety tests required in order to be allowed to operate in a worker shared environment.

The object recognition has the main goal of endowing collaborative robots with the ability to perform tasks in cooperation with humans. The robots needs to be capable of obtaining not only the identification of the objects but also their posture (positions and orientation). The Object Recognition Kitchen framework, from Willow Garage, was used as basis for the detection process. The adopted scenario uses the Sawyer Robot from Rethink Robotics and a Microsoft Kinect camera.

Key Words: autonomous robotics, collaborative robotics, object detection, pallet recognition, Sawyer Robot

Contents

List of Figures

List of Tables

Part I

Scope of the Dissertation

Chapter 1

Introduction

Industry 4.0, also known as the forth industrial revolution, is the name given to an increased investment on digitalization and connectivity of traditional industrial production processes. Several fields of technology contribute to this revolution, the majority being heavy related to information handling and networking.

The main areas of expertise that can characterize and shape Industry 4.0 include the designated "big data", Internet of things, cloud computing and autonomous robotics. When applied together in an industrial environment where the means of manufacturing are provided with mechanisms capable of making them intelligent (or "smart"), it unfolds an array of new possibilities for the optimization and personalisation of production. (I-SCOOP, 2021)

Most contributors and supporters of the production process can be integrated on an Industry 4.0 setting and the sharing of information between them allows for automatic adaptation to new conditions and requirements. Collecting and analysing big volumes of data obtained from the producers themselves allows for the identification and correction of anomalies that otherwise would go undetected. Machine learning methods can be deployed on individual machines or on the cloud for broader utilization. Autonomous robots can learn to operate on human-shared environments and even adapt their behaviours in function of the individuals that surround them.

Taking into account the growing implementation of Industry 4.0 to real industrial concepts, new solutions are required in order to adapt existent technologies or create new ones for this paradigm. With that in mind, the field of Computer Vision is one that attracts great interest.

When taking into consideration key robotic components of this new industry, more particularly collaborative robots and autonomous vehicles, Computer Vision is one of the areas of development that can be exploited for the acquisition of information regarding surrounding dynamic environments. Robots are expected to have adaptive capacities, learning new procedures and scenarios by themselves. Object de-

tection and identification is an important feature in order to endow robots with the capability of operating in joint tasks with human workers. The same can be said for vehicles, where the growing number of shared work spaces increases the need for adaptive solutions.

Computer Vision is widely adaptable and, when taking into consideration the processing speeds currently available, capable of responding in real time. This attributes make the technology attractive, as it can be used on its own or in association with other methods for greater robustness and redundancy.

1.1 Motivation and Goals

Robotics technology is already widely present both in the industry as in day-to-day situations, but the most advanced solutions, the ones that bring big enhancements to Industry 4.0, are still being developed and matured. With that in mind, this dissertation focuses on the acquisition of external information by robots using cameras and Computer Vision software. Specific circumstances that require tailored solutions are addressed (as is the case of the pallet detection for an autonomous stacker) but not only. Generic methods for object detection and pose acquisition, that can be more easily adapted from case to case, are also explored.

Figure 1: A stacker getting ready to pick a pallet

The first topic of the dissertation, referring to pallet detection, is the result of work developed in an academic project for a prominent international company. In this project, autonomous stackers were required to load and transport raw materials and finished goods between the production and storage areas. The management and monitoring of the stackers in the company was based on a global positioning system, using triangulation methods between anchors and tags, allowing for the vehicles to know their localization in the world. The orientation was obtained by information provided by an IMU. Nevertheless, certain tasks, such as the picking of pallets, require a greater precision than what the positioning system is capable of

providing. One of the main goals of this dissertation was the development of a Computer Vision system, with which it is possible to obtain the positioning of a pallet relative to the vehicle, adding a local positioning system that ensures the good execution of the task.

The second topic derives from work done in a simulated work scenario in the University. Parts of a structure are on a table and it is expected that a human worker and a robotic manipulator work together to assemble it. A Computer Vision system was developed to provide information regarding the objects in the workspace so that the robot can act accordingly. This includes identification and pose calculation (position and orientation) of the structure parts in relation to a world frame.

1.2 Dissertation Organization

Two different topics involving Computer Vision applications are part of the core of this dissertation: detection and position identification of pallets for an autonomous stacker vehicle (referred to as Pallet Detection) and object detection and pose extraction for collaborative robots (referred to as Object Detection and Pose Calculation). Throughout Chapter 2, Part III and Part IV, both cases will be presented in parallel, with Pallet Detection appearing first.

Chapter 2 presents the current technologies and possibilities for the solution of the problems at hand, with examples from other works.

In Part II are explained key concepts of Computer Vision and robotics.

In Chapter 4 are described the different resources (hardware and software) used in the projects that are part of this dissertation.

Part III focuses on the steps taken in the development of the solutions presented. Divided into two Chapters, the first concerns the Pallet Detection and the second Object Detection and Pose Calculation.

Part IV documents the tests made throughout the development and the obtained results. Similar to Part III, this is also divided in two Chapters.

Part V is the last and has an overview of the work and the obtained results. It also explores possibilities for future improvement.

Chapter 2

State of The Art

In this Chapter, modern day technologies and methods are explored. The goal is to analyse which approaches are most suited and offer the best chance at solving the problems at hand. Relevant work done by others in the scientific community on similar topics is also used as example and taken into consideration.

Similarly to the rest of this dissertation, the Chapter is divided in two different areas of interest.

2.1 Pallet Approach Maneuvers

With regards to hardware, there are two different approaches to address pallet recognition that present good results when using a camera and Computer Vision. Cameras can either be RGB (this is the most usual type of camera seen in everyday life, producing three channel images) (Byun and Kim, 2008) (Chen et al., 2012) or can be depth cameras (producing single channel images) (Haanpaa et al., 2017). Some models have both options incorporated, generating the two types of images at the same time. This cameras are called RGB-D. This choice of hardware has direct influence over the procedures later employed.

2.1.1 Pallet Detection

For autonomous vehicles in an industrial context, in particular stackers, it is fundamental that they possess the ability to detect pallets and their characteristics with precision. Computer Vision can be the technology chosen to perform such task.

The RGB camera operates on a colour basis. Each channel is a matrix of the same size as the captured image and corresponds to one of the three primary colours of light (Red, Green and Blue). Any pixel of the image can then be represented by three bytes, one for each channel. With a camera of this type, the

detection of the pallet can be done by either colour or edges search. With colour it is required that the pallet is of a predetermined colour and for the luminosity conditions to be constant, as the image captured by the camera is strictly dependant on the light exposure present on the work environment.

The depth camera (or 3D camera) has a single channel, in which the value of each pixel is the distance to the nearest obstruction located in a straight line from the position of said pixel in the image. Whereas in a colour image a pixel has the colour of the area it represents, in a depth image a pixel has the distance to the area it represents. Two distinct technologies dominate the current landscape: time of flight and structured light. Time of flight consists on emitting IR radiation and measuring the time elapsed in order to assess distance. On the other hand, structured light produces an IR radiation field and calculates the distance by examining the displacement created by the surroundings in said field (Kadambi et al., 2014).

The first step requires the identification, in the image, of areas of pixels that may correspond to a pallet, also know as extracting the pallet features. With a RGB camera this process can be done using colour search, as mentioned earlier, or by placing fiducial markers on the pallet like the ones used in (Chen et al., 2012). When using a depth camera, this procedure is done by searching for spatial characteristics (Haanpaa et al., 2017).

Once the existence of the pallet has been acknowledged, it is required to acquire its position and orientation. This can be done effectively with two different methods: identification of the two openings and the respective central points, such as shown in (Byun and Kim, 2008) or identification of the three columns that make the base of the pallet, such as presented by (Chen et al., 2012). With those points in hand, the position and alignment of the pallet relative to the stacker can be calculated.

2.1.2 Discussion

Taking into account the possibilities above, it was decided that finding and identifying each column individually was the best procedure. Colour search was rapidly dismissed given that this type of solution was not general enough. As mentioned above, luminosity conditions have a high risk of influencing the detection and it would require always using the same type of pallet. For similar reasons, the use of fiducial markers (Chen et al., 2012) was also rejected, as it would be limiting the solution to pallets with said markers. Using depth data as shown by (Haanpaa et al., 2017) produces more reliable results that are only affected by possible obstructions.

As for the choice between identifying the columns or the two openings, both options are capable of providing good results, as demonstrated in (Byun and Kim, 2008) and (Chen et al., 2012). The decision ultimately ended up being in favour of finding the columns since knowing their position was always going

to be a necessity.

2.2 Object Detection

The detection of objects on images is one of the most significant activities in Computer Vision. Commonly, the term *Object Detection* stands for the identification and classification of predetermined objects on stand alone images or video frames (Kanimozhi et al., 2019).

Most of the techniques used today for object detection are based on machine learning types, such as Convolutional Neural Networks (see Subsection 2.2.1.2) (Kanimozhi et al., 2019) (Mane and Mangale, 2019) (Kim et al., 2018).

2.2.1 Methods

2.2.1.1 Template Matching

One popular technique for object detection is template matching. There are different methods in which template matching can be applied, but they all revolve around the same principle: previously defined images of objects are taken as templates and passed through the images captured by the camera by resorting to a sliding window (Xiao et al., 2010). The aim is to find "matches", areas in the captured images that present a close similarity to one or more templates.

The same principle used for RGB images can also be applied to point clouds using the structure of an object as template (Hinterstoisser et al., 2010). Using the 3D model of an object, it is possible to generate templates from a rich variety of angles and points of view that are then compared with the data present in the point cloud. When a match is found, the original angle of the template in question is used to produce an estimation of the orientation of the object.

Template matching is a good option for object detection when dealing with objects that have uncommon textures and/or patterns, such as Shogi pieces with Japanese characters (Yata et al., 2015), full body images in surveillance footage or human faces (Chantara and Ho, 2015). Unlike machine learning methods, it does not need long and computational demanding training procedures, allowing for a more agile and practical implementation (Hinterstoisser et al., 2010).

2.2.1.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is the name given to a neural network that has at least one convolutional layer (Skansi, 2018). In a convolutional layer, a logistic regression is passed throughout the

pixels of an image using a kernel, from left to right and top to bottom. In Figure 2 is shown the example of a 3 x 3 pixels kernel being passed through a 7 x 7 pixels image.

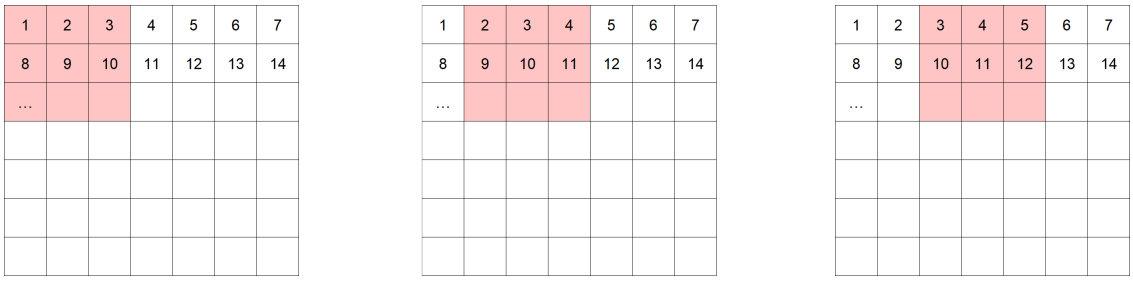


Figure 2: 3 x 3 kernel going through an image

Each new position of the kernel generates the value of one pixel in the output image, making it smaller than the original one in function of the size of the kernel used (see Figure 3). That output image is then passed on to the next layer on the network (Zhang et al., 2015).

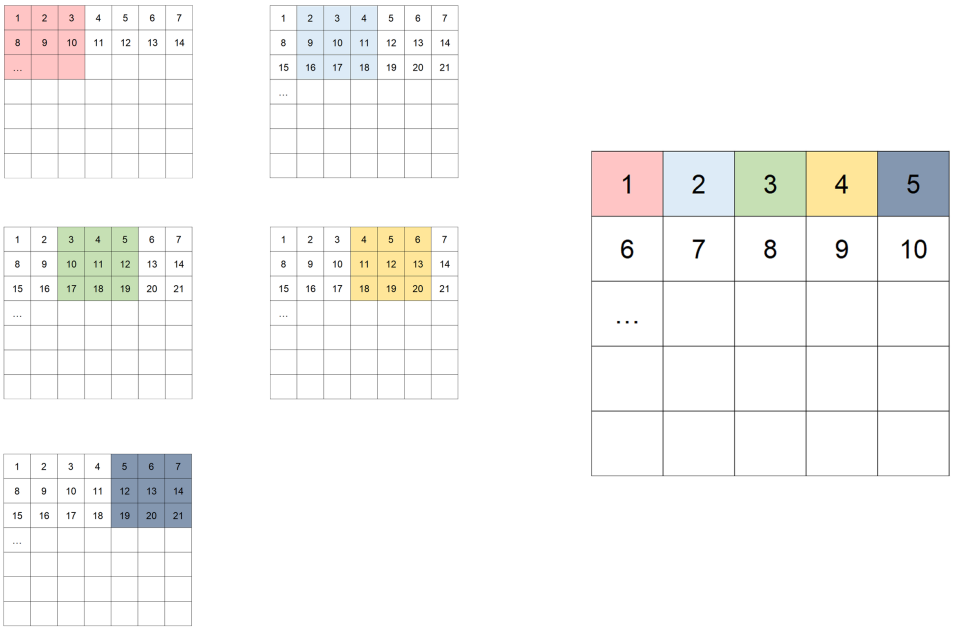


Figure 3: Resulting image being produced from the successive convolutions

Typically, CNNs have multiple convolutional layers and a fully-connected layer at the end making the connection to the output layer (Skansi, 2018). CNNs take an image as input and produce a probability based output in which each neuron from the output layer gets assigned to one object from the list of objects that that specific CNN is capable to predict. The neuron with the highest activation value signalizes the CNN prediction.

Given the limiting nature of the single outputs of CNNs, a more complete method was developed,

the Region-based Convolutional Neural Network (R-CNN). When using R-CNNs, the image is divided in approximately two thousand different regions, with each of those regions being treated as an image and fed to a CNN (Girshick et al., 2014). This procedure allows the identification of various objects from an image as opposed to using only one CNN.

2.2.1.3 YOLOv3

The YOLO (abbreviation for 'You Only Look Once') is a fast and accurate object detection algorithm for RGB images. Contrasting with R-CNNs, that use a network for each region of an image, YOLO employs one network for the entire image, being the network itself responsible for the division into regions and making the process significantly faster (Redmon and Farhadi, 2018). As is also the case with R-CNNs and the TensorFlow Object Detection API (see Subsection 2.2.2.4), YOLO predicts the objects, creating bounding boxes around them and generating confidence probabilities.

Furthermore, a modified version of the YOLOv3 called Expandable YOLO has also been documented. It uses RGB-D images that have a fourth channel with depth information (Takahashi et al., 2020). The depth image is used as extra data that helps the detecting process but does not provide pose.

2.2.1.4 SSD

The Single Shot MultiBox Detector (SSD) is a method for object detection in images. It is very fast and returns viable results.

SSD works by running a convolutional neural network a single time over the image and generating a feature map. For each position in the feature map, standard bounding boxes are tested with different scales and aspect ratios and a score is produced for each pair of hypothetical bounding box/type of object (Liu et al., 2016).

2.2.2 Tools

The above mentioned methods, and others, require in most cases the use of complementary tools that expedite and simplify the procedures, making them easier to implement and adjust to concrete situations.

2.2.2.1 Object Recognition Kitchen

The Object Recognition Kitchen (ORK) is a framework created by Willow Garage to provide support for object recognition operations using template matching. It takes care of database management, handles

inputs and outputs and offers ROS integration (ORK, 2020). This framework was ultimately chosen for the work in hand (see Subsection 4.2.2.1).

2.2.2.2 TensorFlow

Tensorflow is a framework developed by Google for creation and training of machine learning models. Tensorflow combines algorithms of machine learning with deep learning neural networks. It is open source, versatile and can be used for a great variety of purposes, from industrial production environments to mobile applications for smartphones and Internet of things (IoT) (TensorFlow, 2019).

Training can be long and require high computational power. For that reason, Google offers the possibility of carrying out the training process on their servers for a fee.

2.2.2.3 Keras

Keras is a high level interface compatible with the TensorFlow and Microsoft Cognitive Toolkit machine learning frameworks. It simplifies user understanding and utilization of said frameworks. Furthermore, it allows a high level of modularity. The created models can be easily edited or combined in order to generate new models (Keras, 2019).

2.2.2.4 Object Detection API

To facilitate the use of machine learning in Computer Vision object detection applications, TensorFlow has an interface called *TensorFlow Object Detection API*. An object detection model is capable of identifying an object, its position in the image and the percentage of certainty (confidence) that the detection is correct (TensorFlowObjectDetection, 2019), creating a bounding box around the object.

Training a model

In order to train a model, the first step is to collect a high number of images of each one of the objects to identify. 100 images is the minimum recommended (Python Programming, 2019). Each individual image is labelled. The labelling process consists on the creation of a XML file which describes what is the object present on the image and in what coordinates that object is (Phadnis et al., 2018). After labelling, the dataset is divided in two sets: the training set and the test set. It is recommended that 90% of the dataset is placed on the training set, while the remaining 10% are placed on the test set. This relation can be altered if the user deems so. Finally, the images and the respective XML files are all converted into a

single TFRecord file (a TensorFlow specific format). This file serves as the input for the machine learning training algorithm provided by TensorFlow (Python Programming, 2019).

2.2.3 Discussion

Taking into account the different methods mentioned above, a template matching based solution was chosen using the Object Recognition Kitchen framework (see Subsubsection 4.2.2.1). The high adaptability and portability of this method, in comparison to others, was the main incentive behind the choice. As one of the goals of this dissertation was to have general and adjustable solutions, the costly training required by the other options was a clear disadvantage. Table 1 offers a small comparison between methods.

Method	Advantages	Disadvantages
Template Matching (ORK)	- Fast adaptability and training - Returns objects pose	- Fast quality degradation with increased distance - Some inconsistency calculating objects' orientation
R-CNNs	- Accurate	- Long and computational expensive training process - Slower performance than other neural networks
YOLOv3	- Accurate - Faster performance compared to other methods	- Long and computational expensive training process, hindering adaptability
SSD	- Accurate when detecting large objects	- Long and computational expensive training process - Not very accurate when detecting small objects

Table 1: Table summary of the indicated methods

Part II

Theoretical Foundations and Robotic System

Chapter 3

Theoretical Foundations

In this Chapter are explained some important concepts of Computer Vision that were used or investigated for the purpose of this dissertation, such as basic notions and specific techniques.

3.1 Colour spaces

For the purposes of Computer Vision, images are interpreted as matrices, with each pixel being an element of the matrix.

In the case of greyscale images, one variable is enough to characterise each pixel, 0 being black and the maximum value of the data type of said variable being white. This type of image is also called single channel.

As for coloured images, multiple variables are required for each pixel, each one representing a different feature of the colour. This type of images are known as multichannel images. Each channel is, in fact, a matrix and the combination of all the matrices forms the image. There are numerous systems in place, with RGB being the most commonly used (Burger and Burge, 2016).

3.1.1 RGB

RGB stands for Red Green Blue and aims to emulate the three primary additive colours, where colours are created by the addition of these three primaries. RGB images are comprised of three channels, one for each primary colour, meaning that each pixel is characterised by three variables, each being the intensity of the respective primary colour for that pixel.

3.1.2 CMY

CMY stands for Cyan Magenta Yellow and is based on the subtractive primary colours. It works similarly to RGB, where each channel portrays one of the three primaries.

For printing purposes, the derived model CMYK was created, where K stands for Key. Key is the black coloured ink.

3.1.3 HSV

HSV stands for Hue Saturation Value. Also a three channel colour space, HSV can be more practical when manipulating images, as each channel represents a different trait of colour, unlike RGB where all channels are colour intensity.

Hue is the tone of the colour. It can be represented by a circular geometry where 0°, 120° and 240° are the three primary additive colours, respectively (see Figure 4). It is common for the angles of the Hue scale to be divided by two, making the maximum possible value 180°, so that the Hue can be specified by a 8-bit variable.

Saturation is the presence of colour in the pixel. The lower the Saturation, the less the colour defined by Hue is represented. When Saturation is 0, the pixel can be regarded as a greyscale one.

Value is the feature of the colour space that characterises the intensity of light. When Value is 0, the pixel is completely black, the equivalent to (0,0,0) in RGB.

Figure 4: Hue scale

3.1.4 HSL

HSL stands for Hue Saturation Lightness and can be seen as a variation of the HSV colour space. Unlike HSV, where maximum Value indicates maximum light intensity, in HSL maximum Lightness denotes white. The maximum colour intensity in HSL is placed on the middle of the Lightness scale.

3.2 Histogram

A histogram is an alternative system to represent the pixels of an image. It is a two dimensional frequency chart of all the possible pixel values for one channel. Mathematically, a histogram is a graphical representation of a set of data divided into classes. When applied to image processing, to a greyscale image for example, it depicts the distribution of grey values by the number of pixels of that level. Figure 6 shows the histogram of the grey channel of Figure 5.

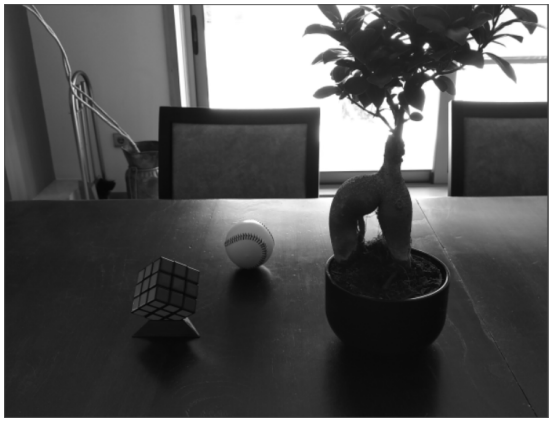


Figure 5: Greyscale Image

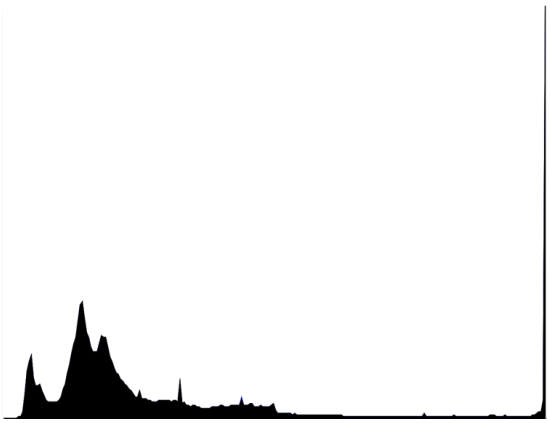


Figure 6: Grey channel histogram

Histogram analyses is a powerful tool to evaluate colour prevalence and contrast in images. For coloured images the histogram is dependent on the colour space used. Figure 8 shows the histogram of the Hue channel of Figure 7.

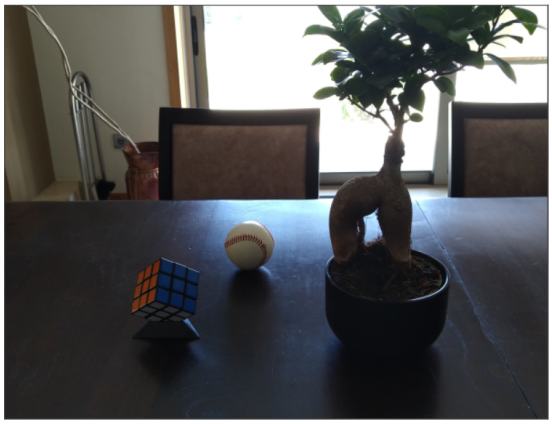


Figure 7: Original Image

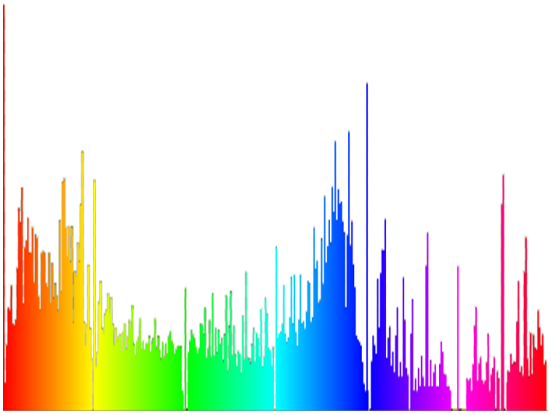


Figure 8: Hue channel histogram

3.3 Dilation and Erosion

Dilation and erosion are two complementary operations used to process images, in which a kernel scans through an image. In the case of the dilation, for each step of the kernel the center pixel is replaced by the maximum value of all pixels in the kernel. This operation increases the brighter areas of an image. The erosion works in the opposite way, the center pixel is replaced by the minimum value of all pixels in the kernel, originating a reduction of the brighter areas of an image (Bradski and Kaehler, 2008).

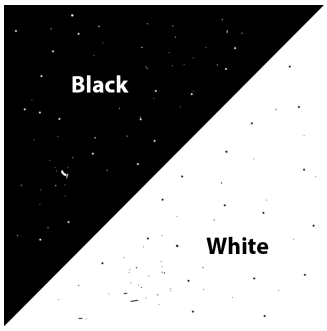


Figure 9: Original Image

Figure 10: Dilation

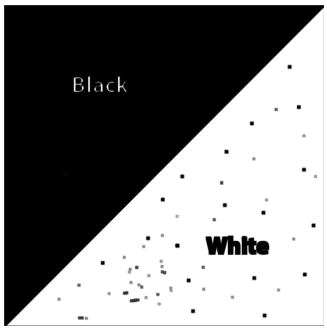


Figure 11: Erosion

When used together, these two operations can be valuable. A dilation followed by an erosion is also known as a ”closing” and it is particularly good at eliminating (closing) dark noise on bright areas (see Figure 12). On the other hand, an erosion followed by a dilation is known as an ”opening” and offers good results at eliminating bright noise on dark areas (see Figure 13).

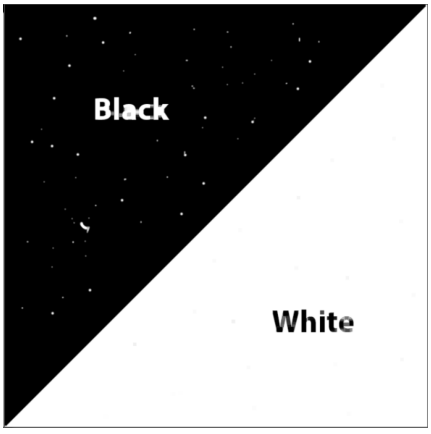


Figure 12: Closing

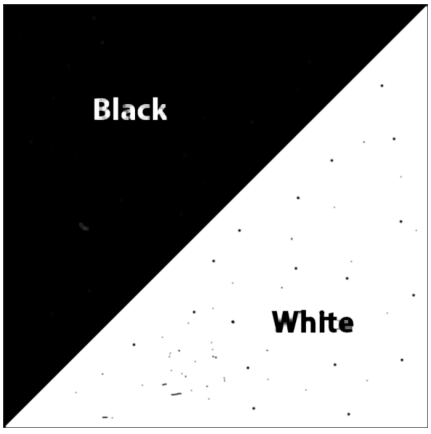


Figure 13: Opening

3.4 Edge Detection

Edge detection is a technique used frequently in Computer Vision applications. Its main purpose is to obtain the contours of the objects present in an image, simplifying the overall information present but maintaining the forms and contours (Muthukrishnan and Radha, 2011).

Numerous edge detection methods have been developed throughout the years, of which the Canny Edge Detector is, to this day, still very popular.

3.4.1 Canny Edge Detector

The Canny Edge Detector is an operator for edge detection in greyscale images. On a first step, the image is passed through a Gaussian filter in order to smooth it and reduce noise. The local gradient magnitude and direction are calculated. After that, the algorithm selects the pixels which are local maxima in the direction of the gradient. These are known as edge points. Lastly, edge points near to each other are connected, forming edge lines (Burger and Burge, 2016).

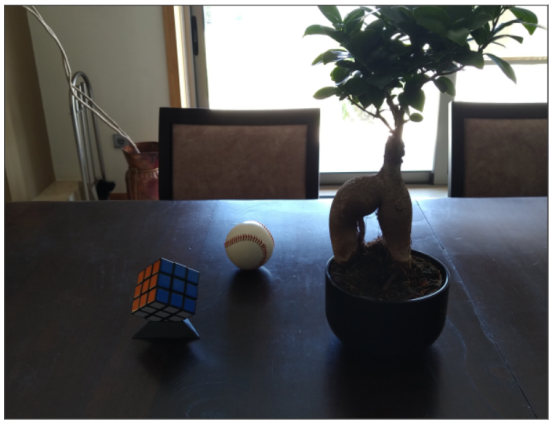


Figure 14: Original Image

Figure 15: Canny Edge Detector results

3.5 Hough Line Transform

The goal of the Hough Line Transform is to identify points that belong to a line on an image. For the purpose of this transform, lines are defined by Equation 3.1 instead of the traditional $y = a.x + b$.

$$x.cos\theta + y.sin\theta = r \tag{3.1}$$

In which r is the length of a second line that starts in the origin, ends in the intersection with the

original line and is perpendicular to it (see Figure 16) and θ is the angle that perpendicular makes with the x-axis.

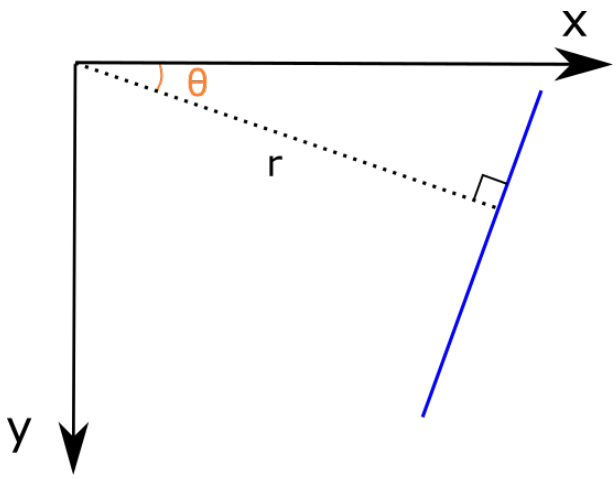


Figure 16: Graphical representation of r and θ

The algorithm goes through the image searching for edge points. When an edge point is found, it simulates all the possible values of the pair (θ,r) for the point in question. For each possible pair, the correspondent position on a θ X r matrix is incremented. When the algorithm reaches the end of the image, the points on the matrix with a value above a certain threshold denote lines present in the image (Burger and Burge, 2016).

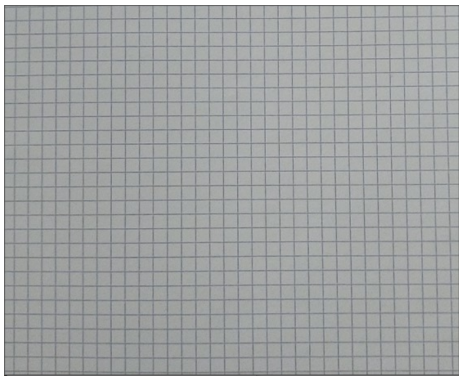


Figure 17: Original Image

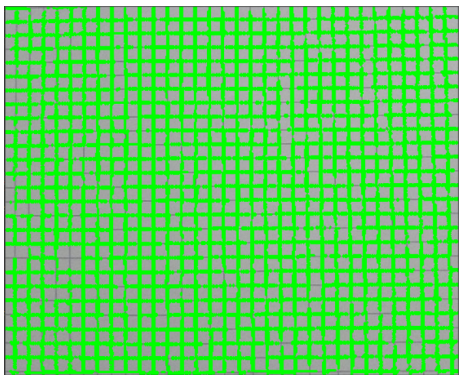


Figure 18: Hough Line Transform results

3.6 Image Equalization

Image equalization is a procedure used in Computer Vision to balance the histogram of an image. When an image has low contrast, that means its histogram is concentrated around a small range of values. The goal of image equalization is to take advantage of the entire channel range in order to spread the histogram and enhance contrast as a result.

3.6.1 CLAHE

CLAHE (Contrast Limited Adaptive Histogram Equalization) is an image equalization method used to improve contrast in images. Unlike the regular histogram equalization, CLAHE equalizes different areas of the image independently and interpolates the boundaries between each area in order to connect them smoothly. Each pixel is transformed in function of the pixels' values in a neighbourhood region. CLAHE also has a gain-limiting quality, preventing the over amplification of noise (Szeliski, 2010).

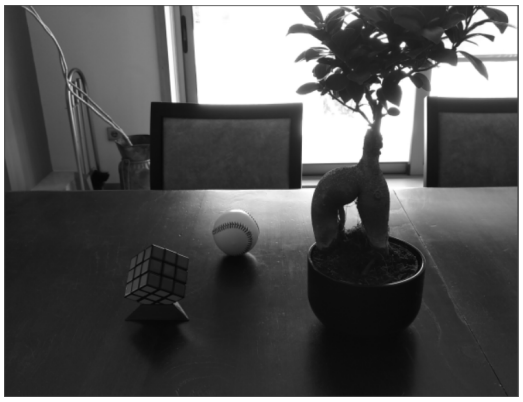


Figure 19: Original Image

Figure 20: After CLAHE

Chapter 4

Robotic System

In this Chapter are presented the robotic systems used for the Pallet Detection and the Object Detection tasks, with a description of the architecture, hardware and software used in both cases.

4.1 Stacker Vehicle

The Computer Vision system developed for the stacker vehicle follows the architecture of Figure 21. The information flows linearly on ROS (see Subsubsection 4.1.2.2). The positioning systems of the stacker are also used by the program when defining a dynamic Region of Interest (see Section 5.3).

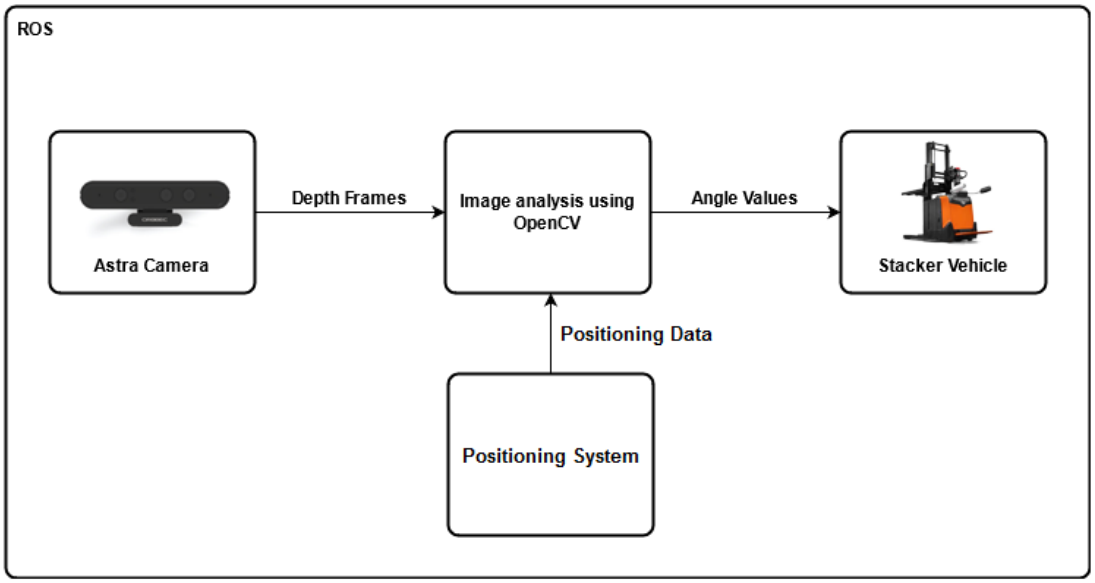


Figure 21: Simplified architecture of the vision system of the stacker vehicle

The vehicle used is a Toyota BT Staxio SPE200DN. It has front wheel drive with a single rotatable front

wheel and two passive back wheels, and is capable of carrying loads up to a maximum weight of 2 t. The stacker measures 2.2 X 0.748 m and has a maximum elevation reach of 2.1 m. The maximum speed varies between 9 km/h and 10 km/h depending on the load being carried (Toyoya Technical, 2021).

With the aim of endowing the vehicle with autonomous capabilities, extra instrumentation was added such as 2D laser range finders, sensors and encoders. A camera (as referenced in Subsection 4.1.1.1) was also included for the purpose of the work described on this dissertation.

Figure 22: Toyota BT Staxio SPE200DN (adapted from https://www.toyota-industries.com.ar/)

4.1.1 Hardware

4.1.1.1 Astra Camera

For the pallet detection functionality in the industrial stacker, the camera chosen was a model Astra from Orbbec. This camera produces both RGB and depth imaging, has a physical size of 165 X 30 X 40 mm and it is power supplied by the same USB port used to communicate with the computer (Orbbec, 2019). Table 2 presents the most important specifications of the camera.

Figure 23: Astra camera by Orbbec (adapted from https://orbbec3d.com/product-astra-pro/)

Resolution	Frame Rate	Range	Field of View
640 X 480 pixels	30 frames per second	0.6 m to 8 m	60° H X 49.5° V X 73° D

Table 2: Technical specifications of the Astra camera by Orbbec

4.1.2 Software

4.1.2.1 OpenCV

OpenCV is an open source library aimed at the development of computer visions applications. It has more than 2500 algorithms already optimized for a variety of intentions such as, for example, facial recognition, tracking of moving objects or generation of 3D point clouds from stereo inputs. It offers interfaces for C++, Python, Java and MATLAB, and is compatible with most operating systems such as Windows, Linux, Mac OS and Android (OpenCV, 2020).

For the purpose of this dissertation, OpenCV was used in the context of C++ programming using a Linux environment.

4.1.2.2 ROS

ROS (Robot Operation System) is an open source framework for development and implementation of robotic software. It integrates countless libraries and tools frequently used in state-of-the-art robotics, including OpenCV (see Subsection 4.1.2.1), and provides an uniformed and modular general-purpose environment that allows easy integration and sharing of solutions (ROS, 2020).

For the purpose of this dissertation, the ROS Kinectic Kame distribution was used in Ubuntu 16.04 LTS.

4.2 Sawyer Robot

Analogously to the system of the stacker vehicle, the one developed with the goal of working on the Sawyer Robot is also based on ROS. Figure 24 presents the simplified architecture that will be explained in more detail in Chapter 6.

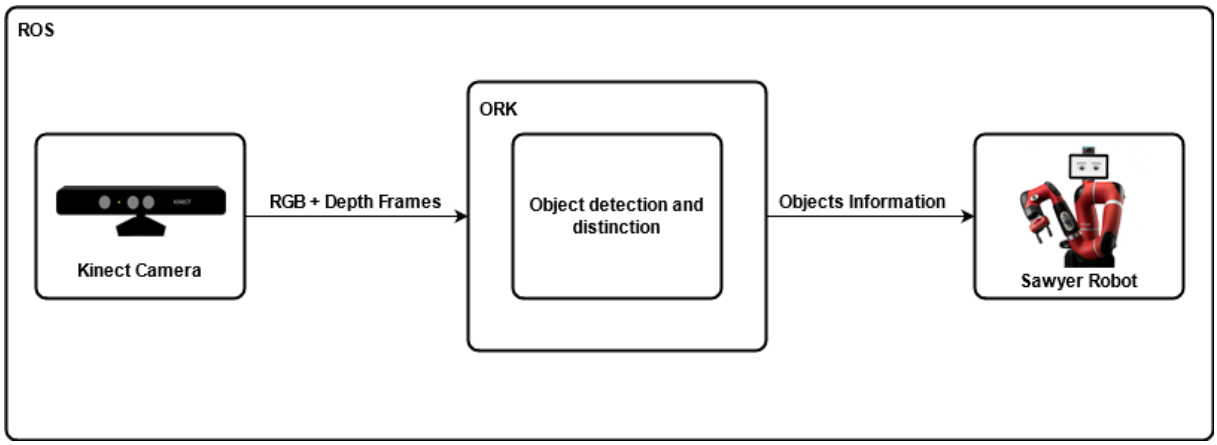


Figure 24: Simplified architecture of the vision system of the sawyer robot

The Sawyer Robot is a collaborative robotic arm made by Rethink Robotics. It has 7 degrees of freedom, a maximum range of 1.260 m, 4 kg payload and is certified to share the workspace with humans. The first fours joints, J0 to J3, have a 350° range, whereas J4 and J5 have 340° and J6 540° (Sawyer Technical, 2021).

It is extremely versatile and according to its manufacture, Sawyer can be used on numerous application from various industries: from plastic injection in the molding and packaging industry to fragile circuit board testing in electronics manufacturing or dangerous activities in metal fabrication (RethinkRobotics, 2021).

Figure 25: Sawyer Robot (adapted from https://www.rethinkrobotics.com/sawyer/)

4.2.1 Hardware

4.2.1.1 Microsoft Kinect

The Kinect is a camera developed by Microsoft. For this dissertation, the first version, the Kinect V1, was used for software compatibility reasons. It provides both RGB and depth imaging ((Cai et al., 2017)). Table 3 presents the most important specifications of the camera.

Figure 26: Microsoft Kinect V1 (adapted from https://3rd-strike.com/microsoft-to-discontinue-kinect-for-windows-v1/)

Resolution	Frame Rate	Range	Field of View
640 X 480 pixels	30 frames per second	1.2 m to 3.5 m	62° H X 48.6° V

Table 3: Technical specifications of the Microsoft Kinect

4.2.2 Software

Similar to the case of the stacker vehicle for pallet detection, OpenCV (see Subsection 4.1.2.1) and ROS (see Subsection 4.1.2.2) were also used with the Sawyer Robot for object detection and pose calculation.

4.2.2.1 ORK

The ORK framework presents different recognition pipelines, that can be use for object recognition. The "LINE-MOD" technique was chosen, which is based on fast template matching. The templates are generated by a random view generator that takes the mesh of an object and generates thousands of different views using variable angles, rotations and scales of the object (LineMod, 2021).

ORK uses a database to store and manage the objects that it needs to detect. Each object must have an associated 3D mesh that can be loaded using either a .stl or .obj file. As an alternative, ORK is itself able to generate a mesh by scanning the object using the camera and a template for reference. The result meshes obtained by this method proved to be less accurate and reliable than the ones loaded by file (see Subsection 6.1).

Part III

Design and Implementation

Chapter 5

Pallet Detection

The algorithm searches for the pallet using the depth images provided by the camera. The depth image is composed by a single 16-bit channel and the value of each pixel is the distance (in mm) that the area represented by said pixel is from the camera.

The algorithm runs through the image and searches for significant value differences between consecutive pixels. These differences, discontinuities in pixel value, are used to represent the outline of the frontal face of a pallet: three columns and two openings. As can be seen by examining Figure 27, when going horizontally through the lower half of a pallet it is expected to find six discontinuities. Each column is responsible for generating two of those discontinuities.

Figure 27: Frontal face of a pallet with three columns and the two gaps between them

For this particular case, the goal is to only detect discontinuities caused by pallets, so it was determined that the difference between consecutive pixels is required to be higher than 35 mm and smaller than 335 mm. As the camera is placed on a higher position, looking down at the pallet, the floor is going to be seen in the area of the openings. The previous values were determined taking that into account, offering a suitable range for pallets with various heights. Differences that are not in this range are ignored because they are not caused by a pallet (see flowcharts in Figures 30 and 31).

After finding six discontinuities along , the algorithm is now able to calculate the position of the center of the three columns: each pair of consecutive discontinuities represents a column and the center is the

point located in the middle of the pair.

With the three columns center points and given that the center of the forks of the vehicle is known and immovable in the image, it is possible to obtain the angles α and β and the distance between the pallet and the vehicle.

α is the angle that the center of the vehicle's forks makes with the central column of the pallet. The desired value for α is 0°.

β is the angle that the center of the vehicle's forks makes with the orientation of the front of the pallet. The desired value for β is 90°. Figure 28 shows an example with a negative α and a β higher than 90°.

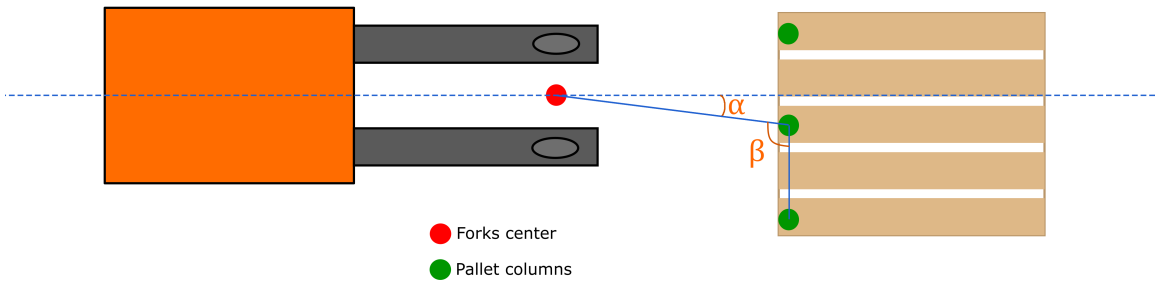


Figure 28: Top view example of α and β angles

The main structure of the program is described in Figure 29:

1. Initializes the node and the node handler;

2. Subscribes to the topics that will contain the depth images (*camera/depth/image_raw*), the vehicle position (*vehicle_pose*), the pallet position (*pallet_position*) and the topic for the start and stop instructions (*start_docking_orientation_error*);

3. Loops infinitely while *ros::ok()* returns *true*. A new frame is processed on each loop if both *allowedToSend* and *newImageFlag* flags are true. A new ROI (region of interest) is calculated if both *allowedToSend* and *newStackerPositionFlag* flags are true.

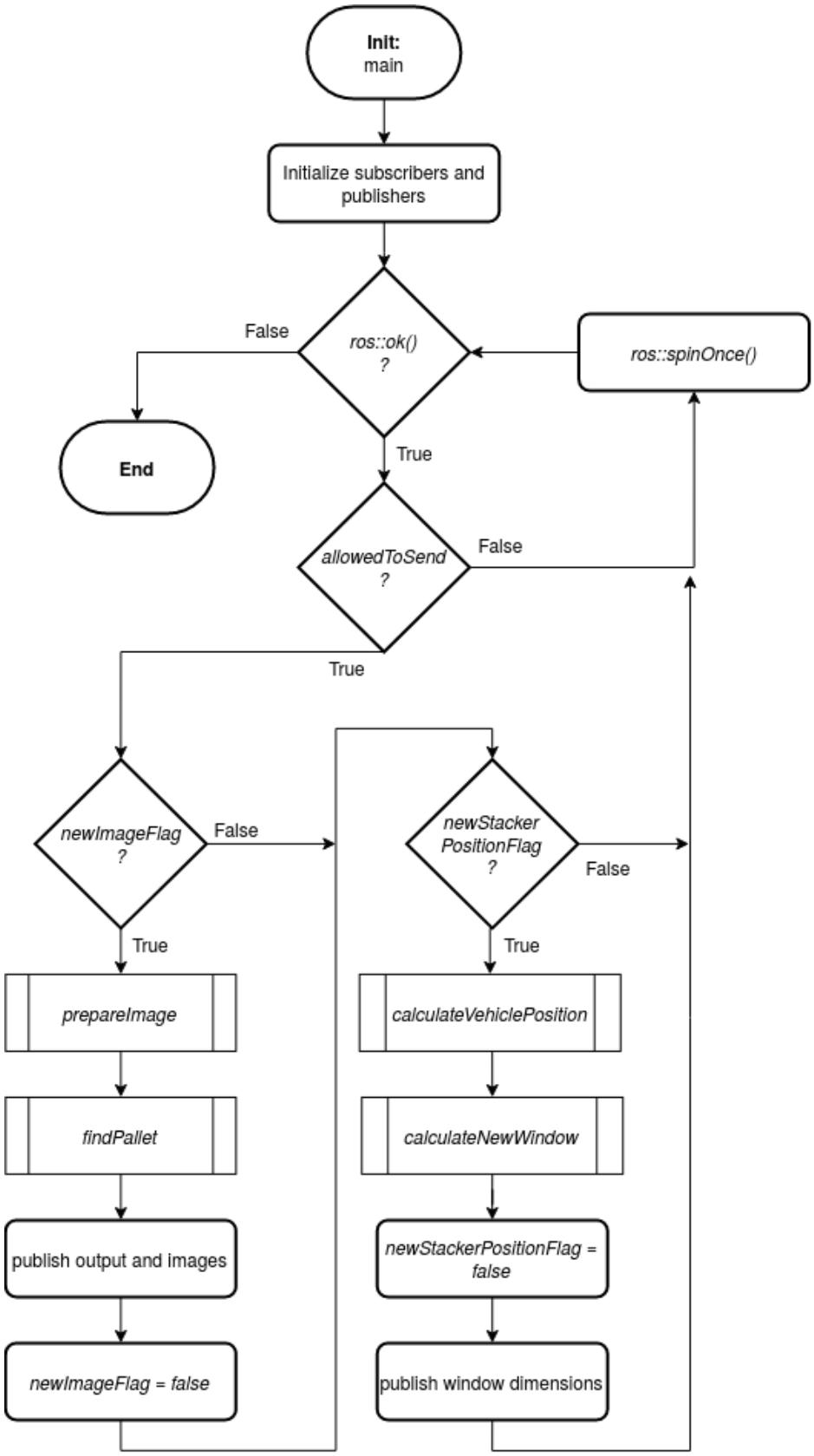


Figure 29: Flowchart of the main structure of the program

Finding a Pallet

The *findPallet* function is responsible for finding a valid pallet on the current ROI (the Region of Interest is explained in Section 5.3) (see Figure 30):

1. Check if the forks are inside the pallet (*areTheForksIn* function);
2. Goes through the ROI and searches for discontinuities using the *findDiscontinuities* function;
3. Once the algorithm has gone through the ROI or found 6 discontinuities, it draws the detected discontinuities on an auxiliary *imgDepthView*;
4. Then it checks if the detected discontinuities make a valid pallet with function *validatePallet*;
5. If the pallet is not valid, *palletFound* flag is assigned false. If it is valid, *palletFound* flag is assigned true and further calculations are made with the function *palletHasBeenDetected*.

Finding Discontinuities

The process of finding discontinuities is done with the function *findDiscontinuities* (see Figure 31):

1. Read the current pixel;
2. Read the next pixel of interest (12 pixels after the current one);
3. Check if there is already a discontinuity near the current area of search;
4. If there is not a discontinuity near, calculate the difference between the values of the current and the next pixel;
5. If the calculated difference is inside the predetermined range, check the value of n pixels before and after the discontinuity to safeguard for cases in which noise created false positives;
6. If the values are consistent, add the new discontinuity to the discontinuities array.

NOTE: In the *findDiscontinuities* function, the *nextPixel* is not the one immediately after the *currentPixel*. There is a 12-pixel distance to account for cases in which the discontinuity can be gradual over a group of consecutive pixels. The discontinuity if detected, is then considered to be in the middle point (6 pixels in front of the *currentPixel*).

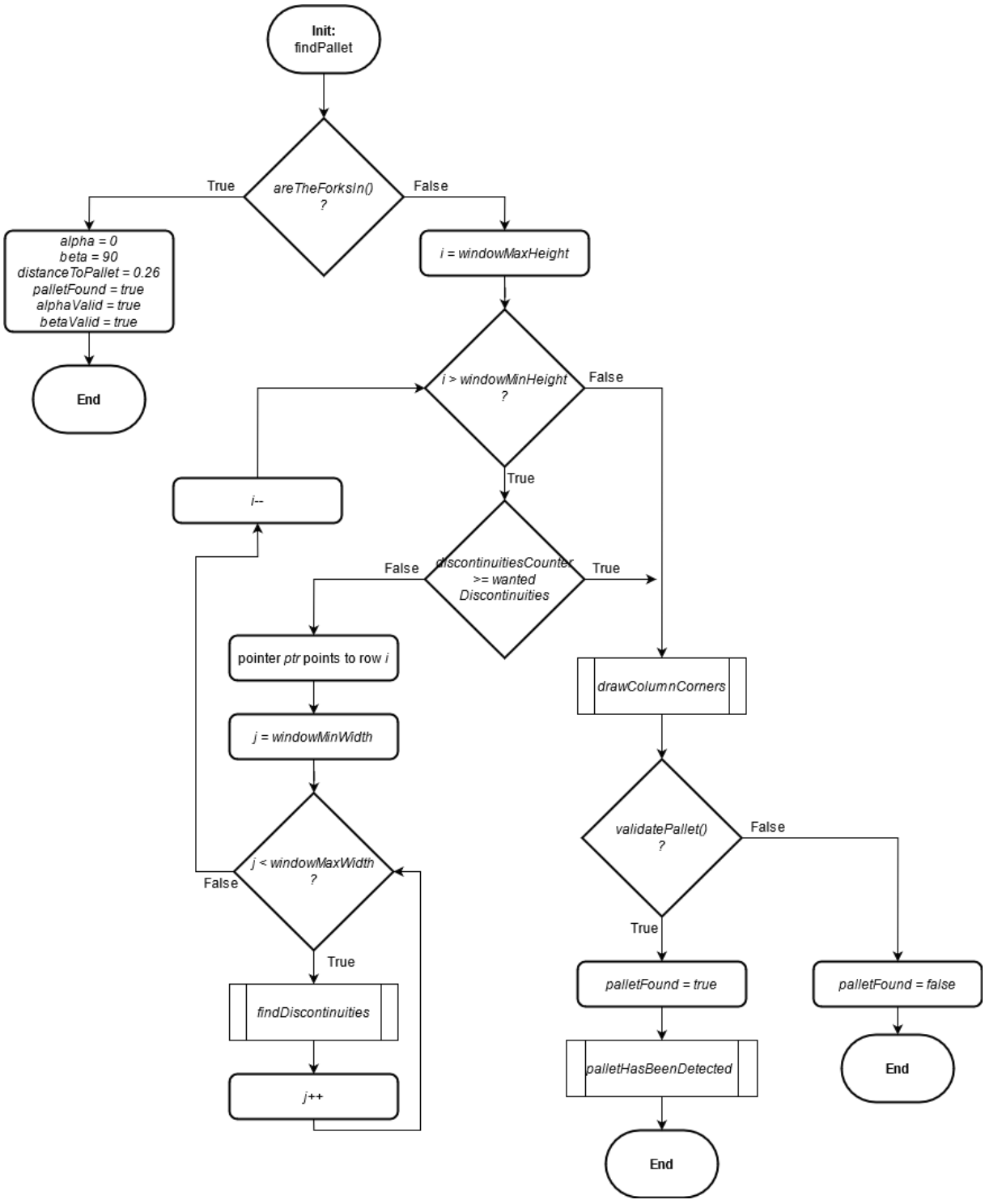


Figure 30: Flowchart of *findPallet* function

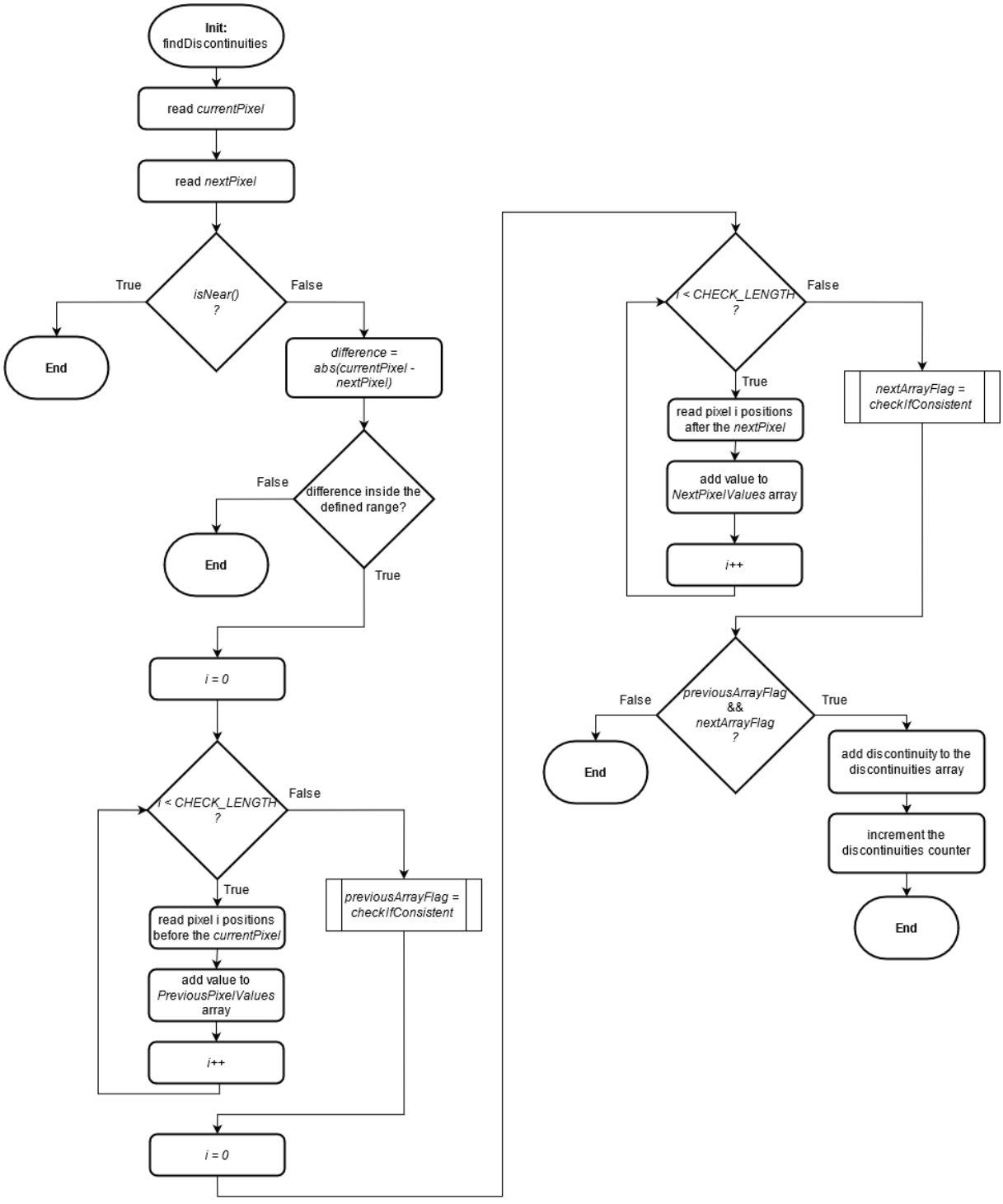


Figure 31: Flowchart of *findDiscontinuities* function

5.1 Misalignment Angles

5.1.1 Angle α

α is calculated using the coordinates of the center of the forks and the center column of the pallet. These two points and the straight line that passes on the center of the forks and has the same orientation as the stacker make a rectangle triangle as the one shown in Figure 33.

Figures 32 and 33 represent an example in which α is negative.

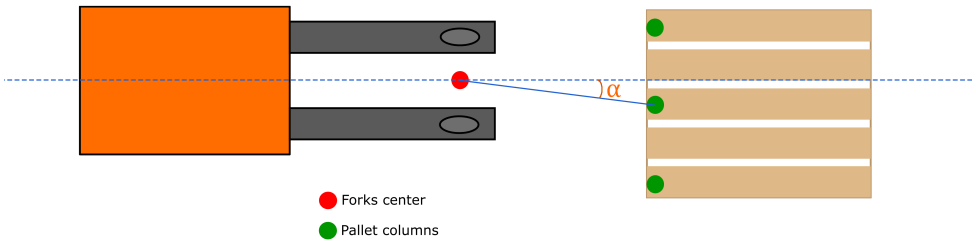


Figure 32: Example positioning

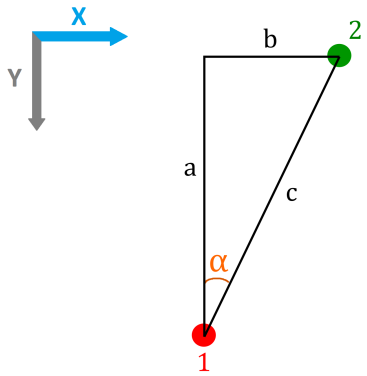


Figure 33: Geometric interpretation

Assuming that point 1 is given by coordinates (x_1, y_1) and point 2 by the coordinates (x_2, y_2), then:

$$\begin{aligned} a &= y_1 - y_2 \\ b &= x_1 - x_2 \\ \alpha &= \arctan(\frac{b}{a}) \end{aligned} \tag{5.1}$$

5.1.2 Angle β

β is calculated using the coordinates of the center of the forks and both center and right side columns of the pallet. The three points form a triangle as the one shown in Figure 35.

Figures 34 and 35 represent an example in which β is lower than 90°.

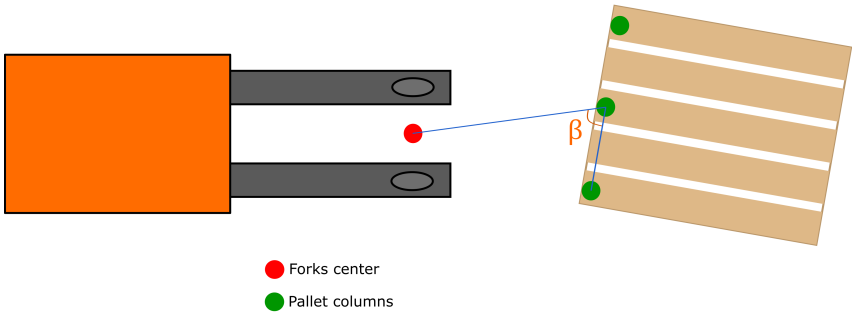


Figure 34: Example positioning

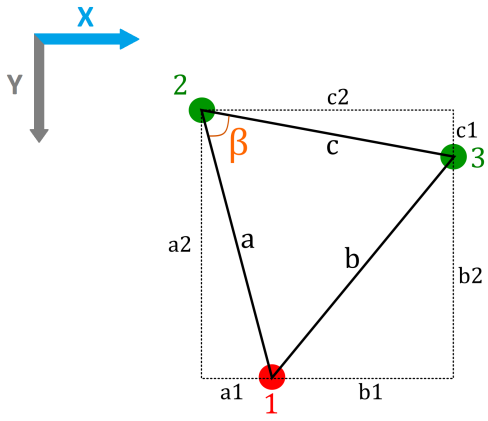


Figure 35: Geometric interpretation

All three sides of the triangle (a, b and c) are hypotenuses of adjacent rectangle triangles shown in Figure 35 and can be calculated with the Pythagorean Theorem.

Assuming that point 1 is given by coordinates (x_1, y_1), point 2 by the coordinates (x_2, y_2) and point 3 by the coordinates (x_3, y_3), then:

Calculating a:

$$\begin{aligned} a_1 &= x_1 - x_2 \\ a_2 &= y_1 - y_2 \\ a^2 &= a_1^2 + a_2^2 \end{aligned} \tag{5.2}$$

Calculating b:

$$\begin{aligned} b_1 &= x_1 - x_3 \\ b_2 &= y_1 - y_3 \\ b^2 &= b_1^2 + b_2^2 \end{aligned} \tag{5.3}$$

Calculating c:

$$\begin{aligned} c_1 &= x_2 - x_3 \\ c_2 &= y_2 - y_3 \\ c^2 &= c_1^2 + c_2^2 \end{aligned} \tag{5.4}$$

The Law of Cosines can now be applied in order to obtain β:

$$\begin{aligned} b^2 &= a^2 + c^2 - 2.a.c.\cos(\beta) \\ \beta &= \arccos(\frac{a^2 + c^2 - b^2}{2.a.c}) \end{aligned} \tag{5.5}$$

5.2 Pallet Distance

The distance to pallet is the distance that goes from the rotation center of the vehicle (center of the forks) to the center column of the pallet. It is calculated according to Figures 36 and 37, where it is represented by *d2*. *height* represents the height of the camera in relation to the ground and *d1* is the distance from the center of the forks to the position of the camera. Both are known and constant. The *hyp* variable is the value given by the camera.

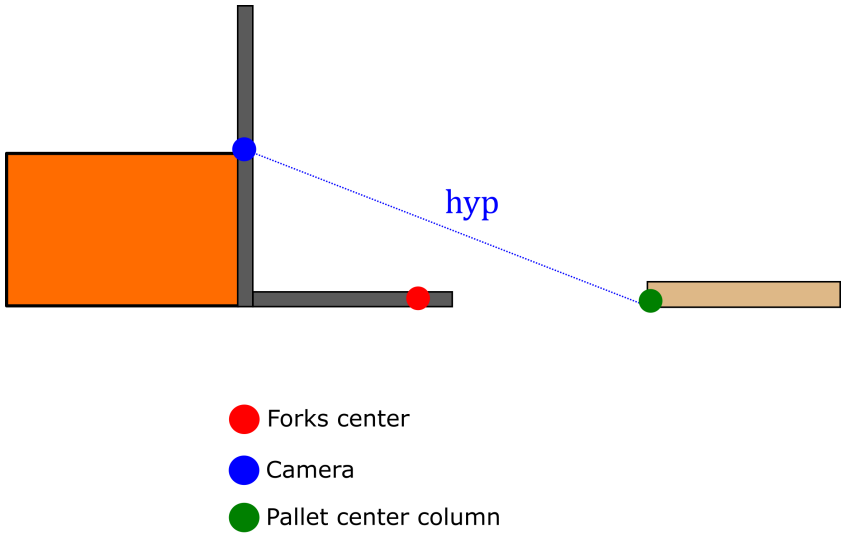


Figure 36: Side view example of the stacker and the pallet

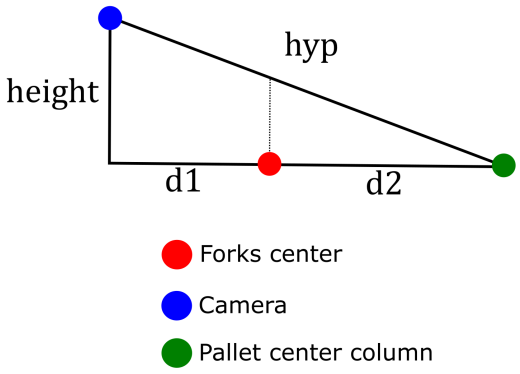


Figure 37: Geometric interpretation

As shown in Figure 37, the situation can be interpreted as a rectangle triangle and *d 2* can be calculated with the Pythagorean Theorem.

$$\begin{aligned} (d_1 + d_2)^2 &= hyp^2 - height^2 \\ d_2 &= \sqrt{hyp^2 - height^2} - d_1 \end{aligned} \tag{5.6}$$

5.3 Dynamic Region of Interest

For the purpose of detecting a pallet, it is required to develop a system capable of ignoring all the information present on the image that is not related to the pallet. A region of interest is used. This region of interest is calculated in real-time using information from the positioning and orientation systems of the vehicle.

The region of interest has the shape of a rectangle. The algorithm focuses only on calculating the left and right sides. The vertical limits can stay fixed, as the danger of non-pallet interference is much smaller (allowing the rectangle to be longer by default) and the vehicle starts the picking manoeuvre at a maximum distance of 70 cm from the pallet expected area, which results in a smaller height variation throughout the process. Three different components are taken into consideration for this purpose:

- The straight line distance between the vehicle and the area where the pallet is supposed to be;
- The position of the vehicle in relation to the position of the area where the pallet is supposed to be;
- The orientation of the vehicle in relation to the orientation of the area where the pallet is supposed to be.

5.3.1 Size of the Region of Interest

The size of the region of interest is determined by the distance between the vehicle and the area where the pallet is expected to be and affects the length of the rectangle. The further the vehicle is to the pallet, the smaller the length of the region of interest needs to be and vice versa.

The camera has a field view of 60° and produces images with a 640 X 480 resolution (see Figure 38) Orbbec (2019).

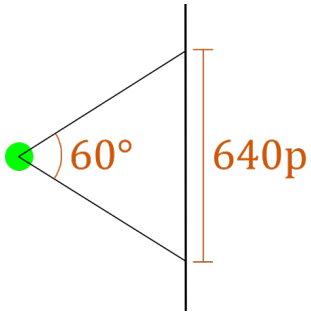


Figure 38: Top view of the camera field of view

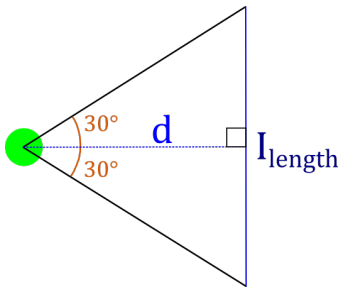


Figure 39: Schematic interpretation

Using the scheme from Figure 39, where d is the distance from the camera to the front of the pallet area and I_{length} is the length, in m, that the 640 pixels of the image cover, the following mathematical equation is used in order to obtain an expression that calculates I_{length} in function of d:

$$\begin{aligned} \tan(30°) &= \frac{I_{length}/2}{d} \\ I_{length} &= 2.d.\tan(30°) \end{aligned} \tag{5.7}$$

The distance d is acquired using the two dimensional positioning coordinates from both vehicle and pallet area:

$$d = \sqrt{(x_c - x_p)^2 + (y_c - y_p)^2} \tag{5.8}$$

Where x_c and y_c are the coordinates of the camera and x_p and y_p are the coordinates of the area of the pallet.

Having, in I_{length}, the metrical length that 640 pixels of the image cover, it is possible to determine how many pixels are needed to capture the entire area of the pallet.

$$\frac{I_{length}}{Pallet_{length}} = \frac{640}{P} \tag{5.9}$$

Where $Pallet_{length}$ is the length expected for the pallet area, in m, and P is the number of pixels that area occupies in the image.

$$P = \frac{Pallet_{length} * 640}{I_{length}} \tag{5.10}$$

Replacing I_{length} for its equivalent:

$$P = \frac{Pallet_{length} * 640}{2.d. \tan(30^\circ)} \tag{5.11}$$

The left and right sides of the region of interest have to distance P pixels between themselves.

5.3.2 Position of the Region of Interest

The position of the region of interest is divided into two different components: the positioning and the orientation of the vehicle relative to the pallet area. Both components cause a left or right displacement in the center of the region of interest, forcing the left and right sides of it to also suffer the same displacement.

Positioning component

The position component compensates for cases in which the vehicle is not in line with the pallet area as shown in the example of Figure 40.

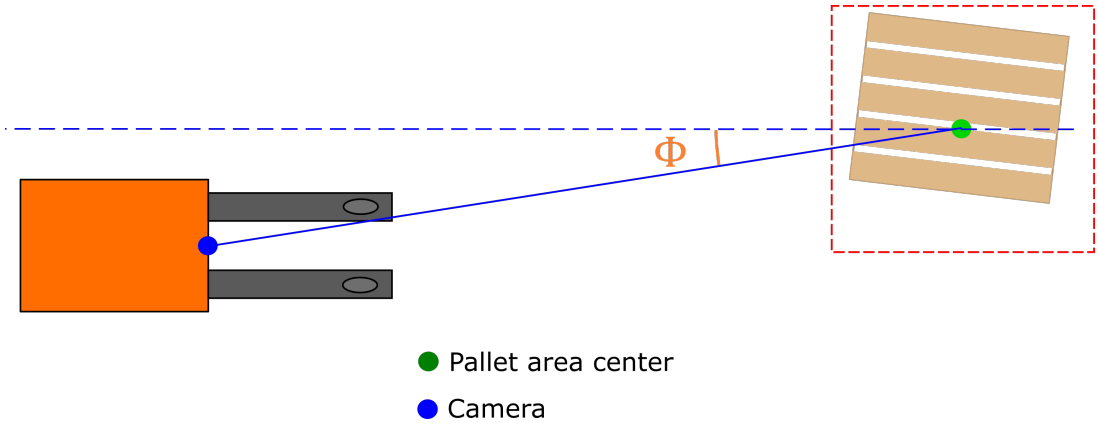


Figure 40: Vehicle not in line with the pallet

The angle Φ is defined by following equation:

$$\tan(\Phi) = \frac{x_c - x_p}{y_c - y_p} \tag{5.12}$$

Where x_c and y_c are the coordinates of the camera and x_p and y_p are the coordinates of the center of the area of the pallet.

With Φ is it possible to obtain the displacement the vehicle has from the plain of the pallet expected area:

$$Position_{dislocation} = d.\sin(\Phi) \tag{5.13}$$

Where $Position_{dislocation}$ is how much the vehicle is dislocated from the plain of the pallet area, in m, and d is the distance as calculated in Equation 5.8 from Section 5.3.1.

The value of $Position_{dislocation}$ needs to be transformed to pixels so that it can be applied to the image. Using the I_{length} from Equation 5.7 from Section 5.3.1 the following relation can be established:

$$\frac{I_{length}}{Position_{dislocation}} = \frac{640}{Position_{component}} \tag{5.14}$$

$$Position_{component} = \frac{Position_{dislocation} * 640}{I_{length}} \tag{5.15}$$

The $Position_{component}$ represents the value, in pixels, that the center of the region of interest needs to dislocate along the x axis of the image to compensate for the different positions of the vehicle and the pallet area.

Orientation component

The orientation component is result of the orientations of the vehicle and the area of the pallet not coinciding. It is obtained by calculating the difference between the absolute orientation of the vehicle and the absolute orientation of the expected pallet area, which are given by the vehicle positioning system:

$$\Delta_\phi = \phi_v - \phi_p \tag{5.16}$$

Where ϕ_v is the absolute angle of orientation of the vehicle, in rad, ϕ_p is the absolute angle of orientation of the expected pallet area, in rad, and Δ_ϕ is the difference between the two.

Knowing that the camera has a field of view of 60° for an image with 640 pixels of length, the following relation can be made:

$$\frac{60° * \pi/180°}{\Delta_\phi} = \frac{640}{Orientation_{component}} \tag{5.17}$$

$$Orientation_{component} = \frac{\Delta_\phi * 640}{60° * \pi/180°} \tag{5.18}$$

The $Orientation_{component}$ represents the value, in pixels, that the center of the region of interest needs to displace along the x axis of the image to compensate for the angle difference.

5.4 Implementation Constraints

The pallet detection module was designed taking into account limitations and requirements of the scenario in which the vehicle is intended to operate.

Due to restrictions imposed by the area the stacker has at its disposal to operate in the pallet picking manoeuvres, there is a distance range prerequisite comprised between 20 and 70 cm between the vehicle and the pallet in order for the picking to proceed.

For a correct execution of the picking task, there is a maximum initial error for the α angle of the pallet. This condition is directly related to the maximum distance to the pallet mentioned in the previous paragraph, as increased angle corrections require more area of operation and bigger starting distances that are not achievable. The maximum value for α is $\pm\ 30°$.

Chapter 6

Object Detection and Pose Calculation

In this chapter it is explained how the software for object detection was developed while integrating the ORK framework and its functionalities. It is also presented a test study case that required the creation of an extra software layer.

The full architecture of the system is represented in Figure 41.

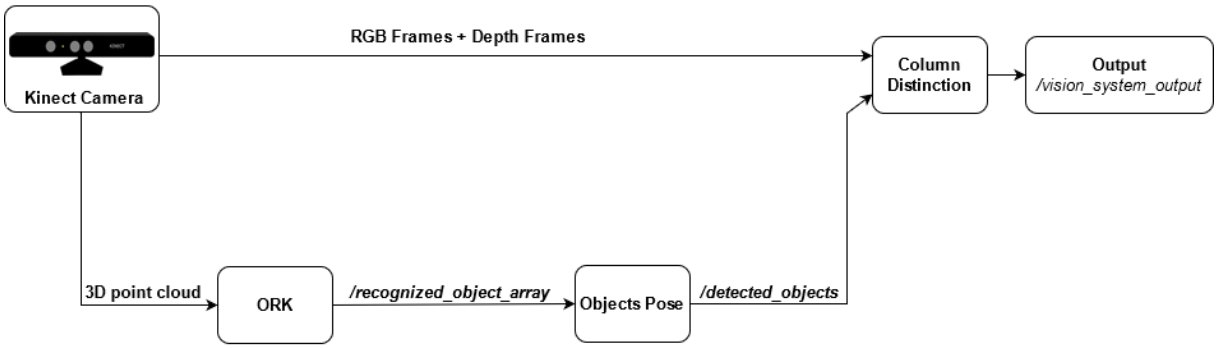


Figure 41: Scheme of the complete column detection process

6.1 Generic implementation using ORK

The first step when using ORK is to introduce the objects on the database. This can be done either by loading an .stl or .obj file or using the built-in function for manually scanning objects using the Microsoft Kinect (see Subsection 4.2.2.1). The preliminary tests with the scanning method, using a can of soda as test subject, provided worse results compared to the case where a mesh of the object is loaded by file into the database. Figures 42 and 43 present the resulting meshes from both approaches.

Figure 42: Soda can model from .obj file

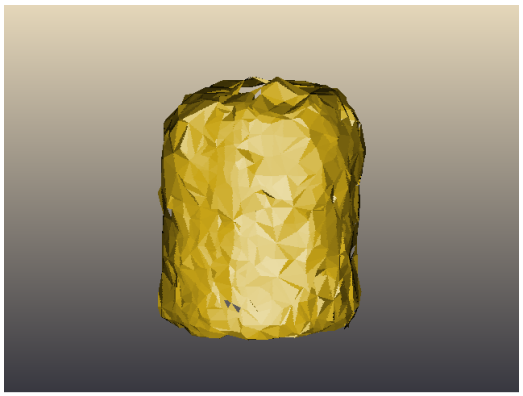


Figure 43: Soda can scanned by the camera

6.1.1 Processing ORK information

The output generated by ORK detection software is published to a ROS topic so that it is widely available for other processes sharing the same ROS environment.

A software module was developed in order to act as an extra output layer for ORK. It has two main purposes:

- Filter the data, eliminating both positive and negative detection outliers;
- Transform the default coordinates provided by ORK into coordinates of a predetermined world frame.

6.1.1.1 Filtering the data

Given that the information provided by ORK is not perfect, a simple filtering mechanism was designed using counters. Each new object present in the output of ORK gets assigned a counter, whose value may be changed with every new incoming message. In order to be considered a "real" detection and not a false positive, an object needs to appear on at least three out of every four consecutive messages. Once that happens, the opposite is required in order to delete the object: it needs to not be included on at least three

out of every four consecutive messages. These values result in a 75 % threshold to determine whether an object should be considered valid or not, which provide a good balance between obtaining good results and keeping the delay introduced to the program low.

The *receivedDataFiltering* function (see Figure 44) is responsible for this procedure:

1. Check if the filter already has objects. If yes, decide if the received information is a new instance of objects that already exist or if it regards new objects. Only the new ones are added to the provisory array. If not, all received objects are added.

2. Update provisory counters. Delete for the provisory objects array the objects whose counter is above the max limit;

3. Decide if the provisory objects are new instances of objects that already exist in the real objects or if they are new ones;

4. For each object in provisory that is not yet in the real objects, check if its counter is above the threshold. If yes, add it to the real objects;

5. Update real counters. Delete for the real objects array the objects whose counter is above the max limit.

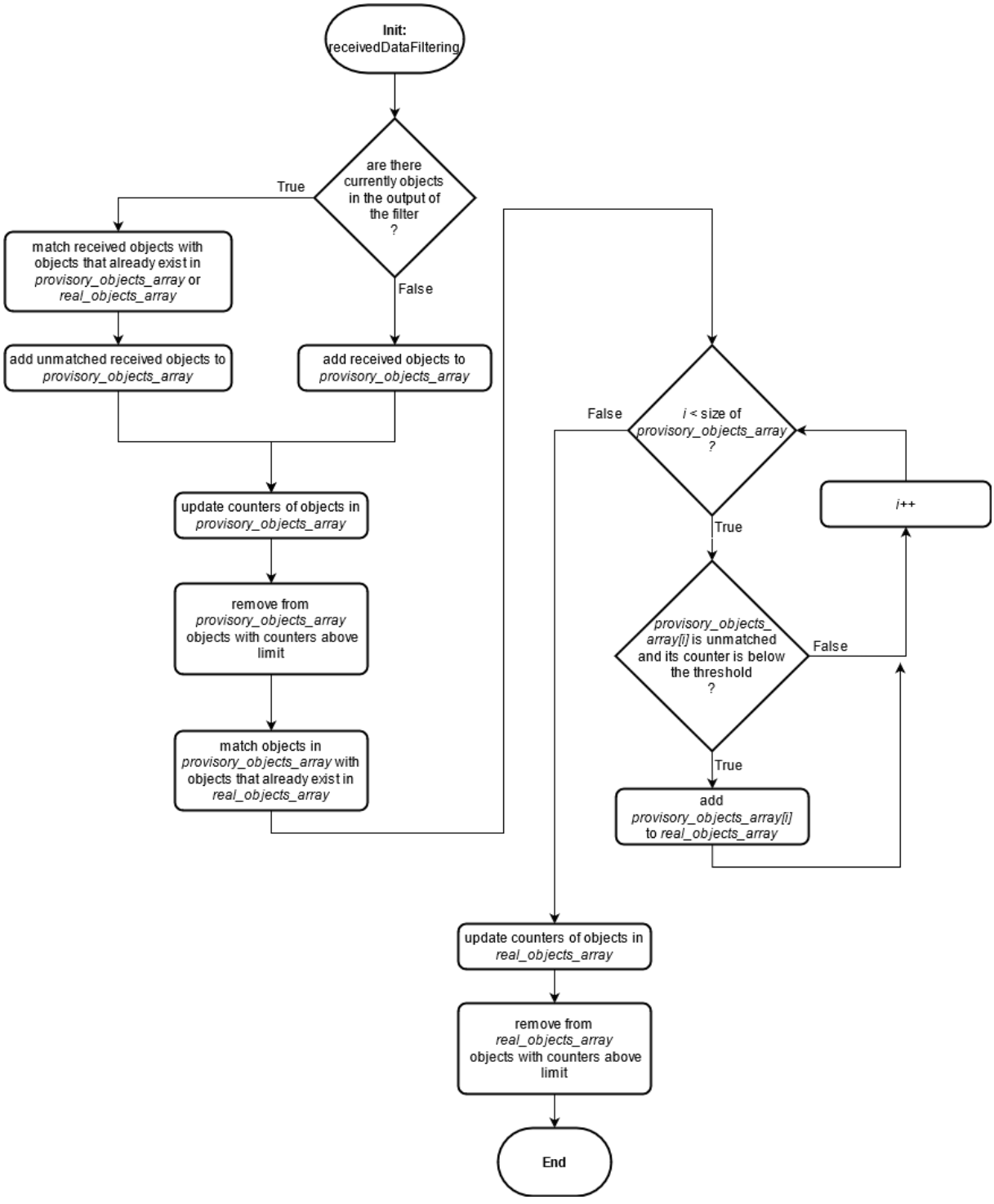


Figure 44: Flowchart of the filtering algorithm

6.1.1.2 Coordinates transformation

ORK publishes information on the ROS topic using objects. Each real life item detected has a corresponding software object published with attributes such as name, confidence and pose. The pose attribute is a standardised ROS feature composed of position and orientation (ROS pose, 2021).

The position coordinates are calculated by ORK using the camera as the origin of the frame as presented in Figure 45.

(a) Front view (b) Side view

Figure 45: Original ORK coordenate axis

In the test scenario using the Sawyer Robot, the camera is placed on a metal structure slightly above and behind the robot and pointing downwards at a 45° angle.

The wanted world frame has its origin on the base of the robot, at table height and is shown in Figure 46.

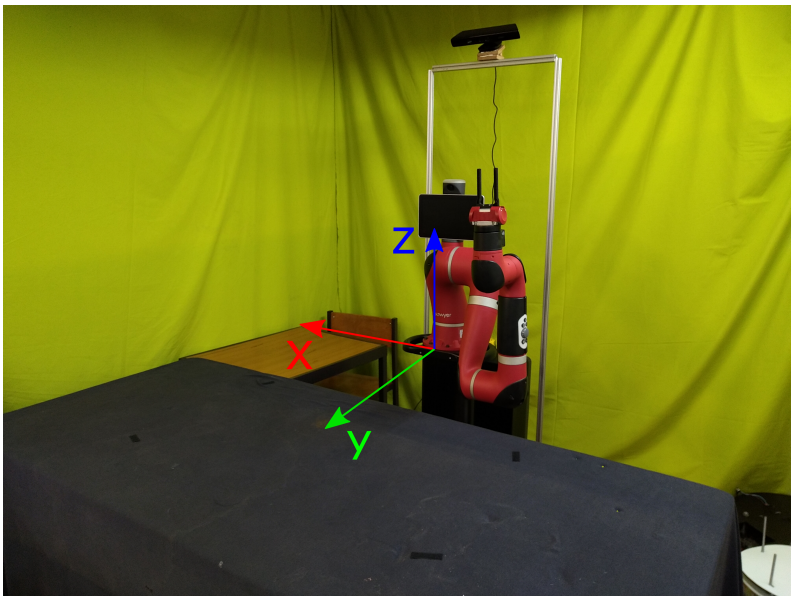


Figure 46: Work table, Sawyer and camera with world coordinates frame

The transformation in question is a rotation around the x-axis of 90° plus the 45° inclination of the camera. A θ rotation around the x-axis is given by the following matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

Additionally, the camera is not placed on the origin of the frame, being 0.83 m above table height and 0.20 m behind the robot.

The full transformation is given by Equation 6.1

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(135°) & -\sin(135°) & 0.83 \\ 0 & \sin(135°) & \cos(135°) & -0.20 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{6.1}$$

In which (x_c,y_c,z_c) are the coordinates of an object relative to the camera frame from Figure 45 and (x_w,y_w,z_w) are the coordinates of an object relative to the world frame.

Furthermore, a rotation around the z-axis is also implemented for cases in which the camera is not capturing the work area from the same side as the robot. It is not used for the scenario at hand. A θ rotation around the z-axis is given by the following matrix:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The *Objects Pose* module is structured as follows (see Figure 47):

1. Check if new objects were received. If yes, pass them through the filter;
2. Check if there are objects in the *real objects array*. If yes, transform their coordinates with the function *transformCoordinates* and publish the data.

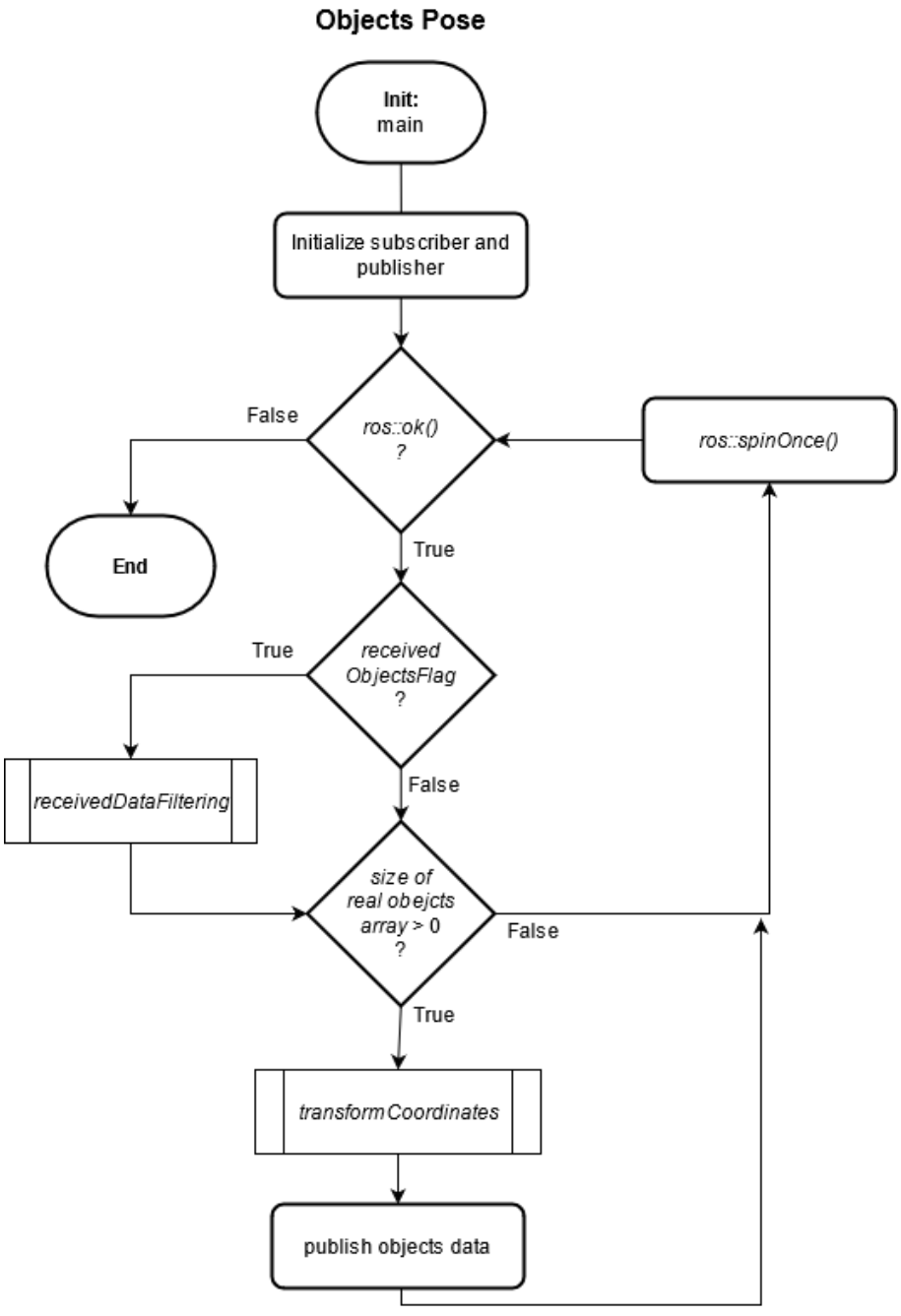


Figure 47: Flowchart of the Objects Pose module

6.2 Test study case

With the purpose of applying the ORK to a real example, a test study case was used with a pre-existent assembling structure. This model consists of columns that are mounted as the edges of a rectangular prism (see Figure 48).

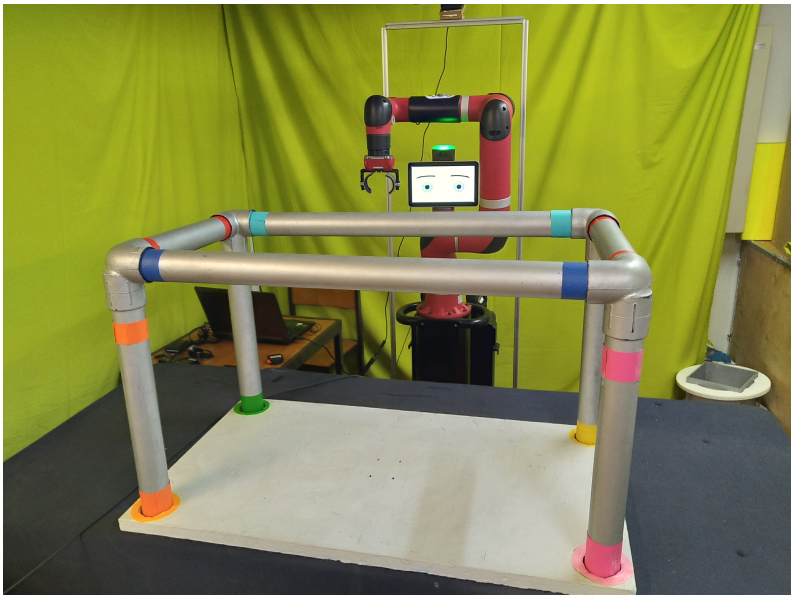
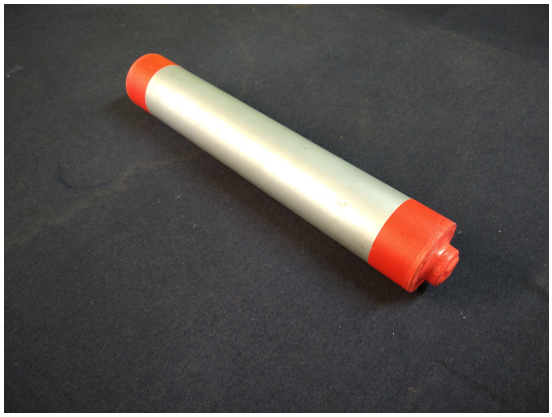
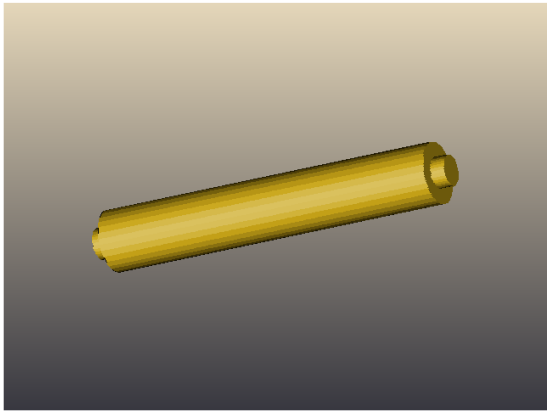


Figure 48: Image of the fully assembled structure

There are two main types of columns present in this assembly: horizontal (see Figure 49) and vertical corner (see Figure 50).



(a) Real column used

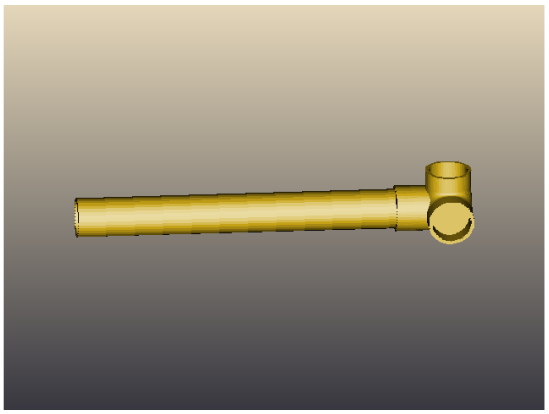


(b) Model of the column

Figure 49: Example of horizontal column of the structure

In the case of the horizontal columns, they appear in two different versions: short with 37.5 cm of lenght and long with 85.5 cm of length. This measurement is the only one differing between them and for the purposes of ORK, one database model is sufficient for detecting both.

(a) Real lateral column used



(b) Model of the column

Figure 50: Example of vertical corner column of the structure

Furthermore, some components of the object orientation are limited depending on the type of column. For horizontal columns, *roll* and *pitch* are constant when the column is on the table, so they are assumed to have the value of 0°. And, because horizontal columns are symmetric, *yaw* can be limited to [0,180]°. For the vertical columns, only *pitch* is constant and assumed as 0°.

6.2.1 Column distinction

Given that ORK only differentiates between objects with different forms, a software model was developed to identify the colour of each column. It uses the data published by ORK as guidelines to help identify the columns on the RGB image. The algorithm runs every time new information from the output of Subsection 6.1.1 and camera frames from the Kinect are published on the respective ROS topic.

6.2.1.1 Colour and depth images alignment

Given that colour (RGB) and depth are captured by different hardware inside the Kinect, the resulting images are not an exact match. In order to facilitate the handling of the images and have the pixels match, the transformation represented in Equation 6.2 was applied to the depth image. The translation of -2 pixels along the x-axis was deemed good enough for the problem at hand, in function of information about the Kinect obtained in other works.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{6.2}$$

In which (x,y) are the original coordinates of the pixels in the depth image and (x',y') are the coordinates of the pixels in the corrected depth image.

6.2.1.2 Image preparation

The first step on the algorithm is to pre-process the image. Now that the colour and depth images are aligned (see Subsubsection 6.2.1.1), the pixels on the RGB image that correspond to depth pixels that are more than 3 m away from the camera are "eliminated" (their value is set to (0,0,0) in the RGB colour space). This aims to remove any information from the background, leaving only data from the workspace. Lastly, some blur is applied to the image. The blur works as a low-pass filter, removing noise.

6.2.1.3 Image position approximation from 3D

The distinction between columns is made by resorting mainly to the colour image. With that in mind, it needs to be established a relation between the positions of the columns in relation to one of the two known frames and their approximated position in the colour image. For the purposes of the algorithm developed for the differentiation, this step is not required to be exact, so the conversion equations were obtained by extrapolation.

$$x_{colour} = 320 + 333.x_c \tag{6.3}$$

$$y_{colour} = 240 + 286.y_c \tag{6.4}$$

In which (x_{colour},y_{colour}) are the coordinates of the pixels in the colour image and (x_c,y_c) are the two coordinates on the original camera frame used by ORK (see Figure 45).

The z_c coordinate also influences the positioning in the colour image, as a depth increment brings the object closer to the center of the image, but given the small distance from the camera to the table this effect was disregarded.

Images have their origin in the top left corner, with the positive x-axis being right and the positive y-axis being downwards. The images have a resolution of 640 X 480 pixels, so the origin needs to be shifted 320 pixels along the x-axis and 240 pixels along the y-axis in order to adjust for the new origin position.

6.2.1.4 Column search in colour images

In theory, the positioning data provided by ORK should be given in reference to the origin of the frame of the respective .obj file. However, this is not consistently observed in practice, with the position point varying along the column. For that reason, in order to locate an object in the colour image, the ORK data can only be used as starting point for a search algorithm.

The edges of the colour image are extracted using the Canny Edge Detector (see Subsection 3.4.1). The resulting edges are slightly dilated (see Section 3.3) in order to eliminate small gaps and close figures that might have been left open by the detector. From these edges are obtained resulting contours that represent all the shapes on the image.

The algorithm proceeds to find and select suitable contours for each column detected by ORK. In order for a contour to be suitable to be a column, it needs to have an area higher than 800 $pixel^2$ and lower than 5000 $pixel^2$. This step is done with the goal of immediately eliminating contours that are too small or too big. For the remaining, it is calculated the distance to each point from Subsubsection 6.2.1.3. The suitable contours for each point are the ones that have that point located inside them or are less than 30 pixels away from it. The 30 pixels value was taken as the maximum error of the conversion in Subsubsection 6.2.1.3. If this value is too high, the program may not work properly in cases where the objects are close to each other, so this balance was found.

Finally, to each point it is assigned only one contour from the suitable ones, the one with the smallest area. The aim of this procedure is to find the contour of the rectangle located between the two colour stripes of each column (see Figures 51 and 52).

Figure 51: Selected contour

Figure 52: Rectangle obtained from contour

Using the OpenCV function *minAreaRect*, objects of the class *RotatedRect* are obtained from the selected contours. The objects of this class have as attributes the center of the rectangle, the size and the

rotation angle (RotatedRect, 2021).

6.2.1.5 Colour analysis

The colour identification of each column is done by taking the rectangles obtained in Subsubsection 6.2.1.4 as reference. In most cases, the colour stripes of a column are situated immediately after the shorter sides of these rectangles. A 20 pixels elongation in both extremes in the direction of the longer side of the rectangle (see Figure 53) was deemed good enough in order to capture the coloured areas. This elongation can not be very large, as it may start acquiring data that is not related to the column.

Figure 53: Resulting image after elongating the rectangle to its sides

There are some times though, where the original rectangle is larger than expected, covering the entire column. This is the result of a mix of lighting conditions and the colour of the column.

Taking into consideration this two aspects, the colour analysis is done by extracting one image, with 40 pixels of width, from each extreme of the rectangle (see Figures 54 and 55). That way it is assured that the colour stripes are part of the images.

Figure 54: Image of the left side

Figure 55: Image of the right side

The histograms of those two images are calculated (see Section 3.2) and split into the possible colours. Table 4 shows the Hue range chosen for each colour.

Colour	Red	Orange	Yellow	Green	Light Blue	Dark Blue	Purple
Colour ID	1	2	3	4	5	6	7
Min Hue Range	2	15	28	40	85	100	140
Max Hue Range	13	25	34	70	99	130	170

Table 4: Hue range for each colour (0 to 180 scale)

The total number of pixels in both images for each colour is added and the colour with the highest number is the chosen one, as long as the value is higher than 50 pixels. If no colour is able to surpass the 50 pixels threshold, then the column's colour is deemed $Undefined$.

The *Column Distinction* module is structured as follows (see Figure 56):

1. Check if new frames and objects were received. If yes, pre-process the frames, approximate the objects position to 2D points and apply the Canny Edge Detector;
2. Goes through all 2D points (each point corresponding to an object);
3. For each 2D point, one contour is selected and the equivalent rectangle is obtained. That rectangle is extended and a new RGB image is extracted from it;
4. Calculate the histogram of the image and determine the colour of the column. Limit and simplify orientations that do not make sense for that type;
5. Publish column data.

6.3 Limitations

Even though this solution offers good results, there are some limitations that need to be taken into account that may affect the normal function of the solution.

The most important limitation regards range. ORK uses the 3D point cloud generated by the Kinect to detect objects, but this cloud has a smaller field of view compared to the colour and depth images. As a result, objects that are in both the left and right edges of the images are not detected. This problem could be solved as future work with the installation of a pan-tilt device. Still in the range feature, the detection loses quality at a fast rate as the object gets further away from the camera (as mentioned in Subsection 2.2.3). This presents a challenge for the placement of the camera, as a balance needs to be met between making sure the horizontal range is acceptable and the distance is not excessive.

Other limitation concerns the orientation of objects. The extraction of orientations is not very precise and the resulting values are inconsistent approximations. Some measures are taken with the goal of minimizing this aspect, such as artificially limiting the angles of some columns, but it is still possible to observe the effects on the results.

Some delay can also be experienced, specially when there are several columns in the scenario. This happens mostly because of ORK, whose execution gets heavier with an increased number of objects. The filtering process described in Subsubsection 6.1.1.1 also introduces delay to the system, but in a much smaller magnitude.

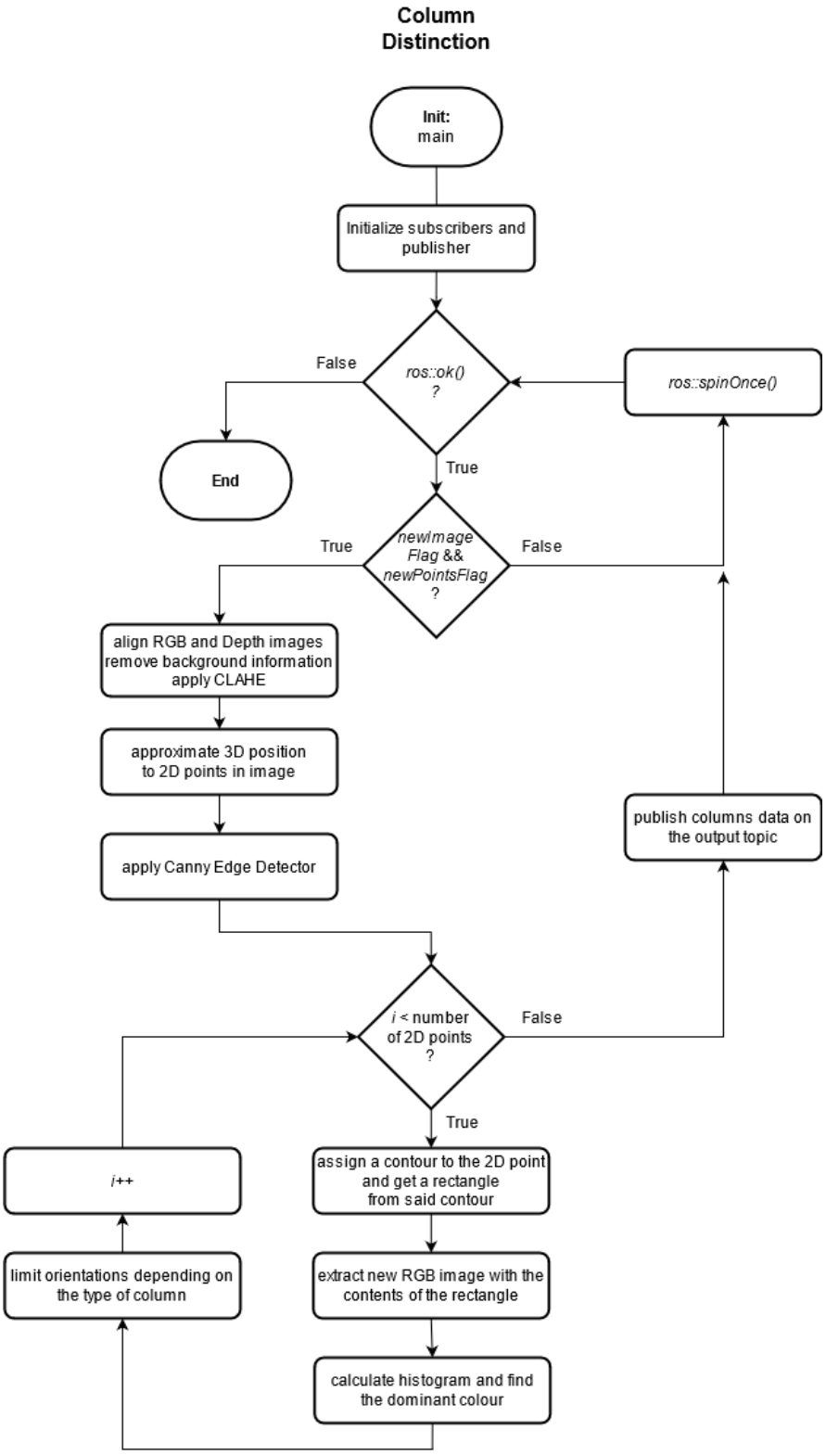


Figure 56: Flowchart of the Column Distinction module

Finally, there are limitations which are characteristic of the scenario itself. With the camera located behind the robot, obstructions are a frequent occurrence during the completion of a task, as the arm is constantly in motion. The size of the structure can also be a limitation, particularly in latter moments of the task when the structure is almost complete and obstructs most of the image.

Part IV

Tests and Results

Chapter 7

Pallet Detection

Testing of the Pallet Detection module was divided in two major steps:

- Testing with pre-recorded videos
- In-vehicle implementation in an industrial environment

7.1 Pre-recorded videos

For the process of testing and validation during the development of the program, sample videos were recorded using the chosen camera (see Subsection 4.1.1.1) in both placement and environmental conditions that aim to emulate real-life scenarios in the most accurately state possible.

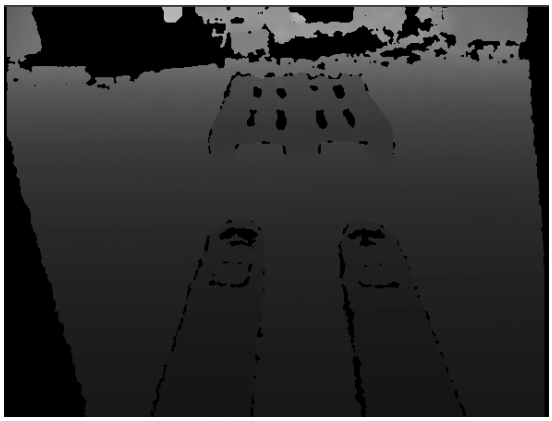


Figure 57: Raw depth image captured

Figure 58: Detection of the edge of the columns

In Figure 58 it is possible to observe, as grey circumferences, the edges of columns that are being detected by the program. This six points are then used to calculate the smaller green circumferences from Figure 59 that represent the columns of a valid pallet.

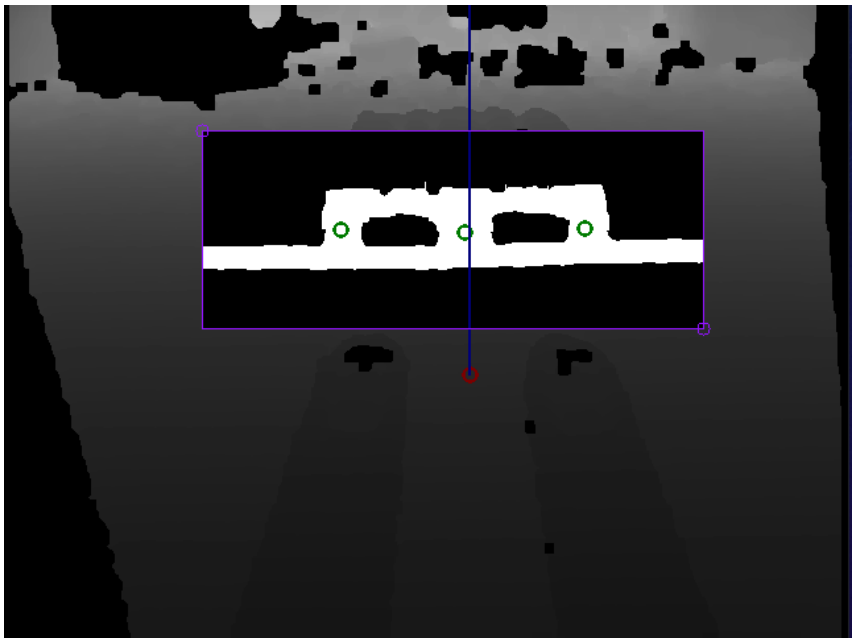


Figure 59: Full representation of points of interest

Furthermore, in Figure 59 it is possible to perceive the red circumference representing the rotation center of the vehicle, the blue line demonstrating the forward direction of the vehicle and a pink rectangle delineating the region of interest (see Section 5.3). For this stage of testing, the region of interest was static and pre-defined given that the information from the navigation system of the vehicle was not being used. In order to ease the visualization by the user, a binary threshold was applied to the area of interest: pixels with depth values close to the depth value of the columns are white, while all the others are black.

```
[ INFO] [1600972523.327338088]: alpha (deg): 3.333851
[ INFO] [1600972523.327353272]: beta (deg): 88.494118
[ INFO] [1600972523.327375233]:
[ INFO] [1600972523.396647900]:
[ INFO] [1600972523.396695493]: alpha (deg): 4.807954
[ INFO] [1600972523.396719189]: beta (deg): 85.192047
[ INFO] [1600972523.396738370]:
[ INFO] [1600972523.466708671]:
[ INFO] [1600972523.466755974]: alpha (deg): 3.742989
[ INFO] [1600972523.466781428]: beta (deg): 86.853821
[ INFO] [1600972523.466799386]:
[ INFO] [1600972523.533522888]:
[ INFO] [1600972523.533572285]: alpha (deg): 4.275841
[ INFO] [1600972523.533587933]: beta (deg): 86.943039
[ INFO] [1600972523.533600047]:
[ INFO] [1600972523.596842683]:
[ INFO] [1600972523.596906501]: alpha (deg): 2.675427
[ INFO] [1600972523.596942225]: beta (deg): 89.841446
[ INFO] [1600972523.596963708]:
[ INFO] [1600972523.668157417]:
[ INFO] [1600972523.668211004]: alpha (deg): 2.121096
[ INFO] [1600972523.668252916]: beta (deg): 89.767090
[ INFO] [1600972523.668269606]:
```

Figure 60: α and β calculated for the shown example

Figure 60 presents the outputs of the algorithm. The program is continuously evaluating the frames fed by the camera and produces new outputs with every iteration for real-time adjustments by the navigation system of the vehicle. For this specific case, the pallet is almost in a perfect position in which the values

of α and β are close to the ideal 0° and 90°, respectively (see Chapter 5). In this stage of testing, the distance to the pallet was not yet being measured.

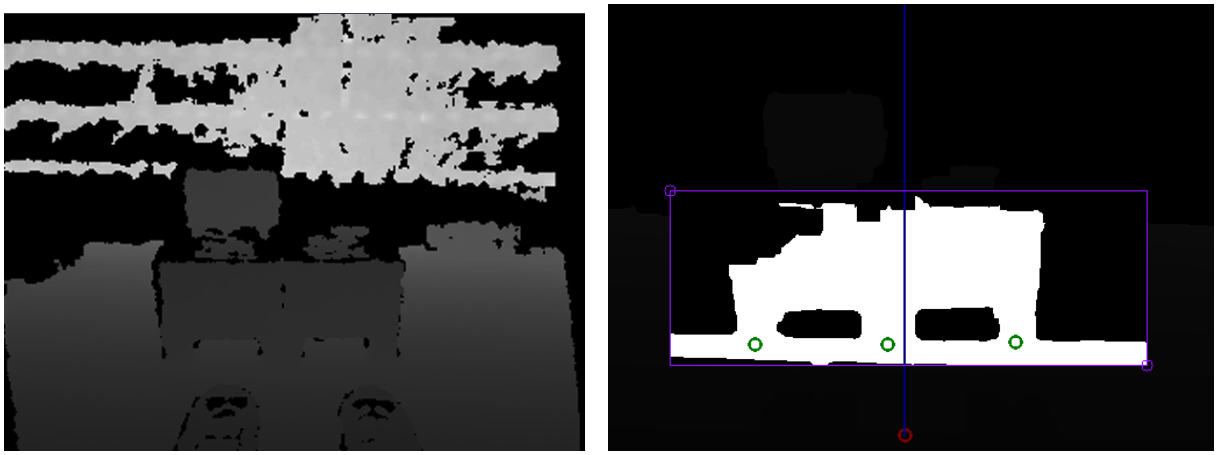
7.2 In-vehicle Implementation

Real life testing took place in an industrial facility in a relevant environment. The vehicle was able to successfully complete full services with pick, drop and park maneuvers. In order to guide the process of picking, the developed program was used.

In the first (Figures 61 and 62), the pallet started located to the left of the vehicle. In the second (Figures 65 and 66), the opposite happened with the pallet starting to the right of the vehicle.

7.2.1 Correction to the left

In this case the pallet is slightly to the left of the forks of the vehicle, as can be seen in Figure 61. α has a value higher than 0° while β is an acute angle. The results in Figure 62 also attest to that. Moreover, it is also provided the distance to the pallet, in m, which was not the case in the previous testing from Section 7.1.



(a) Raw depth image captured by the camera (b) Full representation of points of interest

Figure 61: Detection of the pallet in real life scenario (pallet to the left of vehicle)

pallet found, *alpha valid* and *beta valid* are boolean variables for guidance to the navigation systems:

- *pallet found* returns **True** when a pallet is detected and **False** when it is not detected;
- *alpha valid* and *beta valid* are both variables to account for the existence of outliers. Given the iterative nature of the detection process, it is possible that sporadic errors occur. When a value of α

or β differs greatly from recent values of that same variable, $alpha\ valid$ or $beta\ valid$ become **False** in order to signalize it.

```
---
alpha: 13.7069610839
beta: 71.782895565
distance_to_pallet: 0.504401621129
pallet_found: True
alpha_valid: True
beta_valid: True
---
alpha: 11.8530044177
beta: 73.7688617706
distance_to_pallet: 0.485838198703
pallet_found: True
alpha_valid: True
beta_valid: True
---
alpha: 10.8855269048
beta: 75.2341518402
distance_to_pallet: 0.477090124489
pallet_found: True
alpha_valid: True
beta_valid: True
---
```

Figure 62: Program output

Figure 62 has the numerical results obtained from the example shown in Figure 61. Three consecutive valid iterations are presented. The stacker is approaching the pallet, performing the picking movement and correcting the α angle. Figure 63 shows a graph with the evolutions of $alpha$ and $beta$ in regards to their expected values throughout a correction to the left. Even though the angles are corrected in sequence, they are not independent with $alpha$ affecting $beta$ and vice-versa.

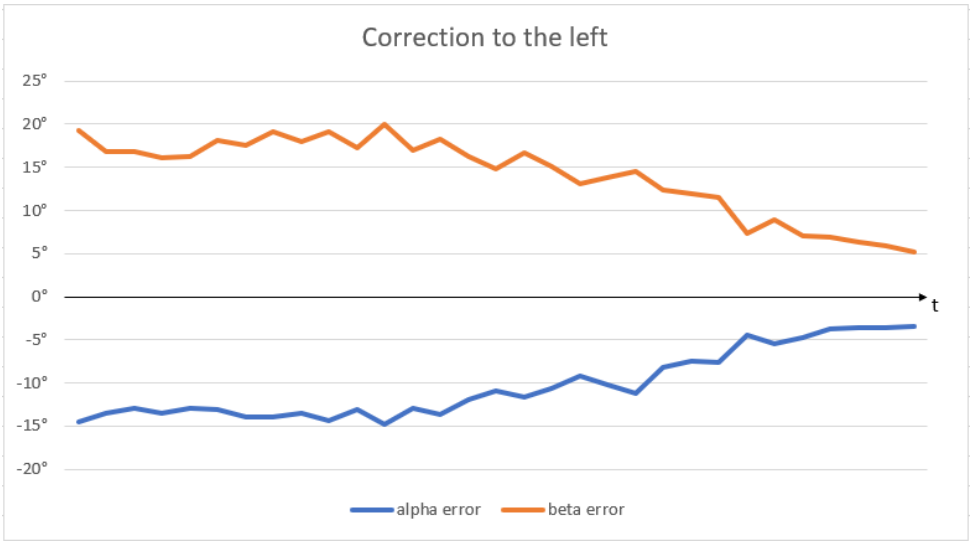


Figure 63: Evolution of the error of α and β during the manoeuvre

By analysing the snapshots of Figure 64, it is possible to observe the evolution of the angles during the correction manoeuvres. In snapshot A the vehicle starts its approach to the pallet, correcting α by

moving the front to the left side. This process continues in B and is complete in C. In snapshots D and E, the vehicle continues to move towards the pallet, this time aligning its orientation with the orientation of the pallet and correcting β. From snapshots F to I the vehicle moves forward inserting the forks in the pallet.

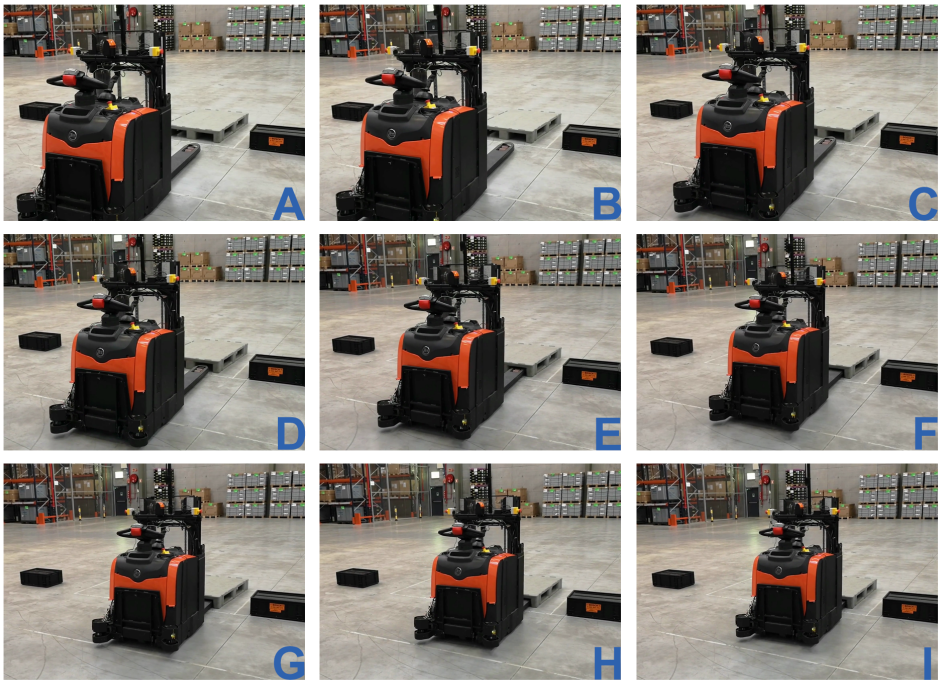


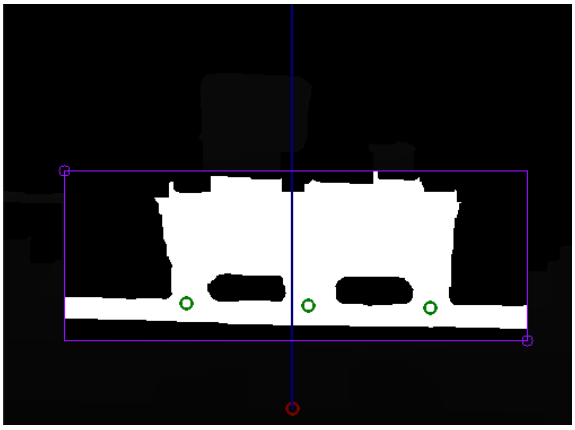
Figure 64: Snapshots of stacker picking a pallet and correcting its position (https://youtu.be/ql-6Jm4FQ9A)

7.2.2 Correction to the right

In this case the opposite of Subsection 7.2.1 is verified. With the pallet to the right of the forks of the vehicle, α is negative while β is an obtuse angle, as can be seen in the values of the respective variable in Figure 66.

Figure 66 has the numerical results obtained from the example shown in Figure 65. Three consecutive valid iterations are presented. In this case, the stacker is further away from the pallet compared to the case in Subsection 7.2.1. Figure 67 shows a graph with the evolutions of $alpha$ and $beta$ in regards to their expected values throughout a correction to the right.

(a) Raw depth image captured by the camera



(b) Full representation of points of interest

Figure 65: Detection of the pallet in real life scenario (pallet to the right of vehicle)

```
---
alpha: -9.86580698247
beta: 103.995522022
distance_to_pallet: 0.582984400662
pallet_found: True
alpha_valid: True
beta_valid: True
---
alpha: -8.65254169718
beta: 101.402041197
distance_to_pallet: 0.578663107545
pallet_found: True
alpha_valid: True
beta_valid: True
---
alpha: -8.74616224636
beta: 102.330787182
distance_to_pallet: 0.570040645686
pallet_found: True
alpha_valid: True
beta_valid: True
---
```

Figure 66: Program output

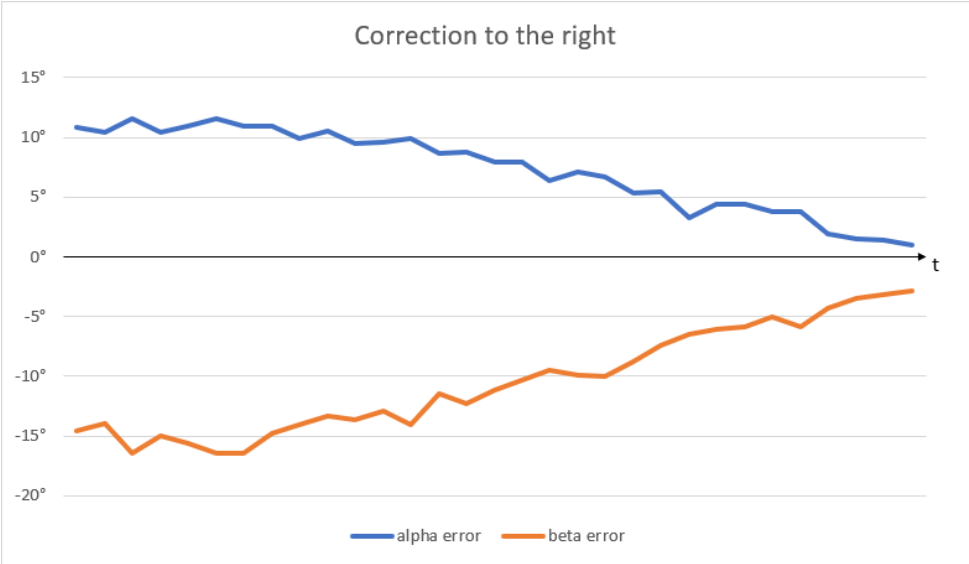


Figure 67: Evolution of the error of α and β during the manoeuvre

Figure 68 shows the image sequence of the stacker picking a pallet while making a correction to the right. In snapshot A the vehicle starts its approach to the pallet, correcting α by moving the front to the right side. This process continues until D. In snapshots E and F, the vehicle continues to move towards the pallet, this time aligning its orientation with the orientation of the pallet and correcting β. From snapshots G to I the vehicle moves forward inserting the forks in the pallet.

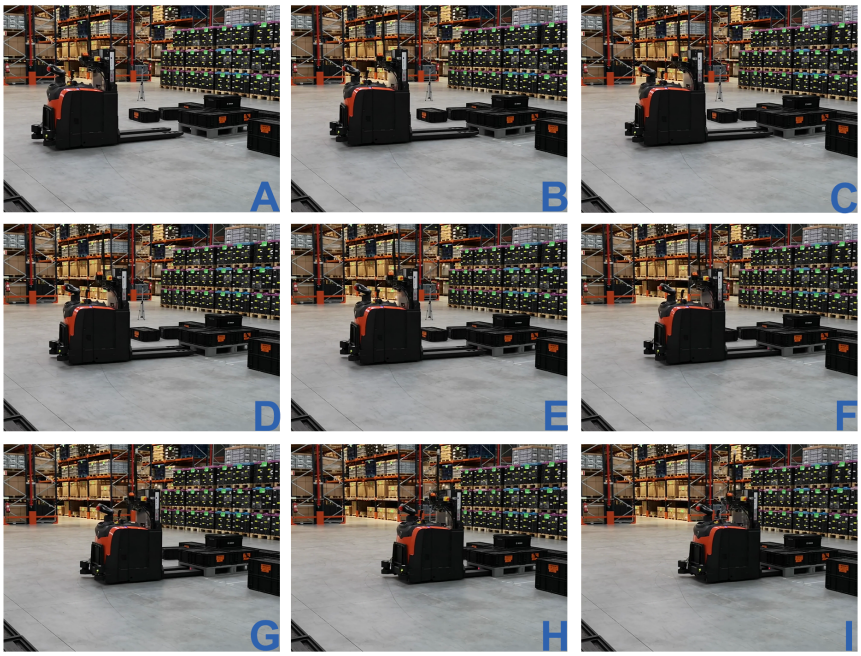


Figure 68: Snapshots of stacker picking a pallet and correcting its position (https://youtu.be/V1X54lX9TYM)

Chapter 8

Object Detection and Pose Calculation

Testing for the Object Detection module was done in a controlled environment in the MARLab (Mobile and Anthropomorphic Robotics Laboratory). On the first stages of testing, the columns were tested individually and in groups. In the later stages, the software was tested with the assembling task in progress.

8.1 Individual Testing

This stage of testing had the aim of evaluating the successful operation of the software in ideal conditions, with little to none moving parts and interference.

8.1.1 Small Red Column

The small red column is the smallest and simplest column of all, making it a good subject for the first tests. With the column placed alone on the table, the software was run.

Figure 69: Raw frame received from camera

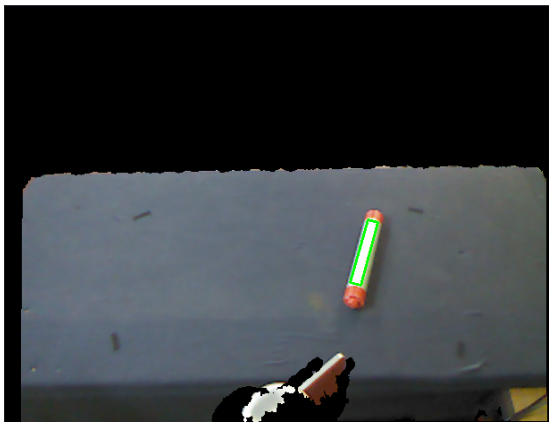


Figure 70: Column detection

Figure 71 shows the output of the *Objects Pose* module, where the data received from ORK is filtered and the coordinates transformed (see Subsubsections 6.1.1.1 and 6.1.1.2).

Two consecutive iterations are presented of just one column being detected. *Image Position* refers to the original position coordinates from ORK (input of this module) and *World Position* refers to the world frame coordinates. *Object ID* is a unique ID given to each object that gets detected during the execution of the software. In this case, only one column ever appears on frame so its ID is 0.

```
[ INFO] [1624572750.093622162]: ##################################################
[ INFO] [1624572750.160082878]:
[ INFO] [1624572750.160118878]: size: 1
[ INFO] [1624572750.160134639]: counter: 0
[ INFO] [1624572750.160149300]: ------------------------------------------------
[ INFO] [1624572750.160161006]: IMAGE POSITION | Object ID: 0
[ INFO] [1624572750.160175576]: x=0.235950    y=0.128873    z=1.637052
[ INFO] [1624572750.160188433]: ------------------------------------------------
[ INFO] [1624572750.160202330]: ------------------------------------------------
[ INFO] [1624572750.160212247]: WORLD POSITION | Object ID: 0
[ INFO] [1624572750.160222917]: x=0.235950    y=0.880334    z=0.058827
[ INFO] [1624572750.160233651]: ------------------------------------------------
[ INFO] [1624572750.160244565]:
[ INFO] [1624572750.160278187]: ##################################################
[ INFO] [1624572750.226776637]:
[ INFO] [1624572750.226815525]: size: 1
[ INFO] [1624572750.226828848]: counter: 0
[ INFO] [1624572750.226841616]: ------------------------------------------------
[ INFO] [1624572750.226855660]: IMAGE POSITION | Object ID: 0
[ INFO] [1624572750.226871206]: x=0.236897    y=0.129376    z=1.643092
[ INFO] [1624572750.226884829]: ------------------------------------------------
[ INFO] [1624572750.226899189]: ------------------------------------------------
[ INFO] [1624572750.226914956]: WORLD POSITION | Object ID: 0
[ INFO] [1624572750.226938102]: x=0.236897    y=0.884485    z=0.054412
[ INFO] [1624572750.226958263]: ------------------------------------------------
[ INFO] [1624572750.226981621]:
[ INFO] [1624572750.227040772]: ##################################################
```

Figure 71: Results from the position algorithm

Figure 72 shows the output of the *Column Distinction* module, where the data received from *Objects Pose* and the RGB and Depth frames from the Kinect are evaluated in order to find out which specific column is there.

Two consecutive iterations of one column being evaluated are also presented. The number of pixels found for each colour are displayed, of which the colour red is clearly dominant.

At the end of *Column Distinction*, the final results are published on the respective ROS topic, *vision_system_output*, using a custom ROS message with all the objects detected in the current iteration of the full algorithm. Figure 73 exhibits one of the messages published on said topic. It is correctly indicated that it is a *Small Red Column* and the colour ID is 1, which stands for red (see Table 4). The confidence variable is supplied by ORK when detecting the column. The positioning and pose variables are provided by the *Objects Pose* module. For this type of column only the *yaw* (*world_orientation_z*) is supplied. In regards to the position, the column is placed slightly to the right of the center of the table, which the results confirm as the column is 88 cm in front and 24 cm to the right of the base of the robot (origin of the world frame).

```
[ INFO] [1624572750.128479919]: ##############################################
[ INFO] [1624572750.189355991]: Table Objects Array Size = 1
[ INFO] [1624572750.192998231]:
[ INFO] [1624572750.193019298]: OBJECT No. 0
[ INFO] [1624572750.193028165]: Red 233
[ INFO] [1624572750.193057772]: Orange 6
[ INFO] [1624572750.193065890]: Yellow 105
[ INFO] [1624572750.193074804]: Green 20
[ INFO] [1624572750.193090946]: Light Blue 0
[ INFO] [1624572750.193103099]: Dark Blue 9
[ INFO] [1624572750.193117715]: Purple 15
[ INFO] [1624572750.197609077]:
[ INFO] [1624572750.197656741]: ##############################################
[ INFO] [1624572750.258544931]: Table Objects Array Size = 1
[ INFO] [1624572750.262061018]:
[ INFO] [1624572750.262085778]: OBJECT No. 0
[ INFO] [1624572750.262106659]: Red 182
[ INFO] [1624572750.262125604]: Orange 0
[ INFO] [1624572750.262142211]: Yellow 62
[ INFO] [1624572750.262157994]: Green 15
[ INFO] [1624572750.262174476]: Light Blue 0
[ INFO] [1624572750.262191834]: Dark Blue 5
[ INFO] [1624572750.262205869]: Purple 19
[ INFO] [1624572750.265352006]:
[ INFO] [1624572750.265381323]: ##############################################
```

Figure 72: Results from the differentiation algorithm

```
---
header:
  seq: 162
  stamp:
    secs: 0
    nsecs:         0
  frame_id: ''
number_of_objects: 1
key: - '1'
type_name: - Small Red Column
colour_id: [1]
confidence: [95.03968048095703]
image_position_x: [0.23689690232276917]
image_position_y: [0.1293761432170868]
image_position_z: [1.6430915594100952]
world_position_x: [0.23689690232276917]
world_position_y: [0.8844854235649109]
world_position_z: [0.054411936551332474]
world_orientation_x: [0.0]
world_orientation_y: [0.0]
world_orientation_z: [74.60445404052734]
---
```

Figure 73: Results published on the ROS topic (https://youtu.be/pCc9D1X7If4)

With the goal of testing the sturdiness of the software, the column was placed in 3 different positions and 10 samples were registered for each position. The position of the column was measured manually and the average error and larger deviation were calculated. Table 5 presents the results.

The average positional error for this case has values up to $\pm$ 2 cm while for *yaw* the error is around the 5° range.

Position	**Manually Measured Positions**				**Average Values Obtained**			
	x_m	y_m	z_m	yaw_m	$\bar{x}$	$\bar{y}$	$\bar{z}$	$y\bar{a}w$
1	26	83	4	70	24,58	85,21	2,75	75,16
2	-39	74	4	20	-39,94	75,46	4,35	24,20
3	30	54	4	170	29,10	55,037	5,21	166,07
Position	**Average Error**				**Larger Deviation**			
	e_x	e_y	e_z	e_{yaw}	$\lvert\sigma_x\rvert$	$\lvert\sigma_y\rvert$	$\lvert\sigma_z\rvert$	$\lvert\sigma_{yaw}\rvert$
1	1,42	-2,21	1,25	-5,16	2,50	3,69	2,51	7,54
2	0,94	-1,46	-0,35	-4,20	1,49	2,04	0,86	6,80
3	0,89	-1,037	-1,21	3,93	1,55	1,92	1,84	4,99

Table 5: Summary of the sturdiness tests for the red column (all positions are in cm and all angles in degrees)

8.1.2 Corner Green Column

The corner green column is geometrically identical to the other three corner columns (yellow, orange and purple). As in Subsection 8.1.1, the column was placed alone on the table.

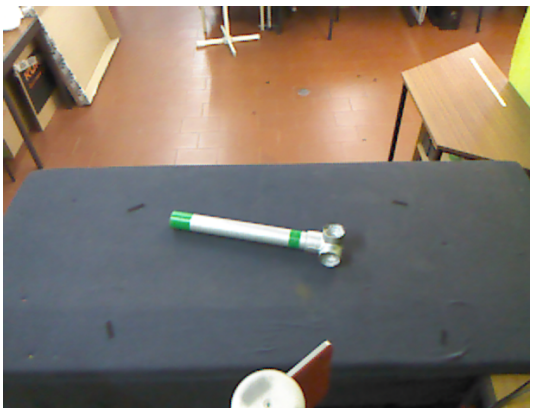


Figure 74: Raw frame received from camera

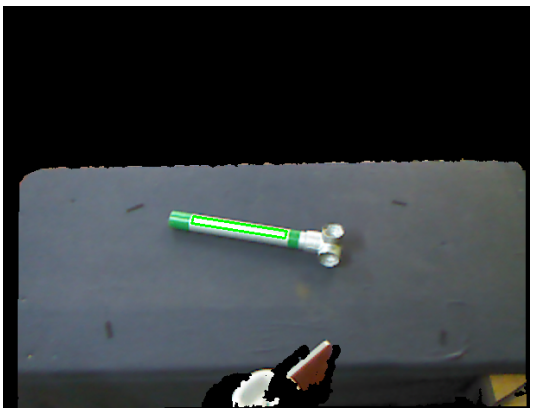


Figure 75: Column detection

Figure 76 shows the output of the *Objects Pose* module, where the data received from ORK is filtered and the coordinates transformed (see Subsubsections 6.1.1.1 and 6.1.1.2).

Image Position refers to the original position coordinates from ORK (input of this module) and *World Position* refers to the world frame coordinates.

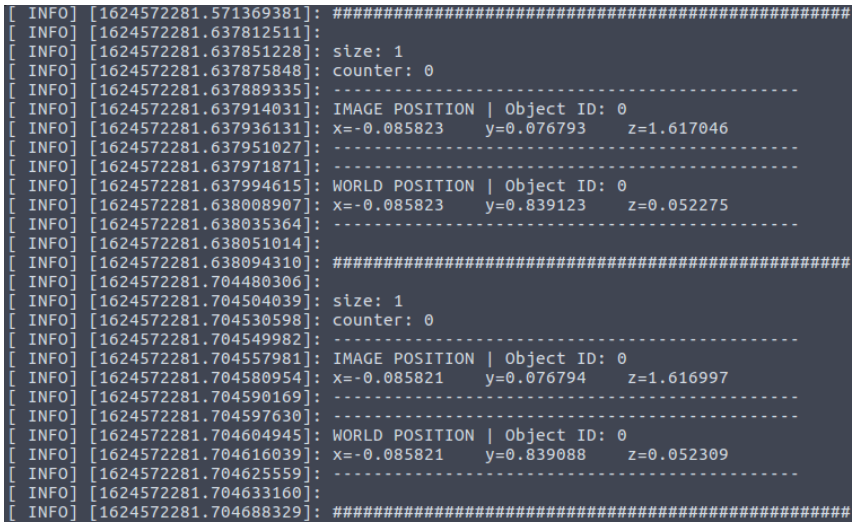


Figure 76: Results from the position algorithm

Figure 77 shows the output of the *Column Distinction* module, where the data received from *Objects Pose* and the RGB and Depth frames from the Kinect are evaluated in order to find out which specific column is there.

The number of pixels found for each colour are displayed, of which the colour green is clearly dominant.

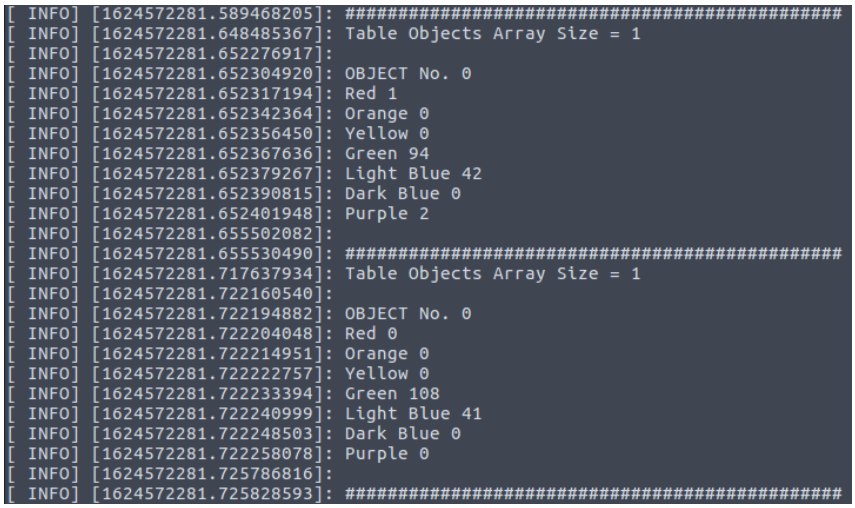


Figure 77: Results from the differentiation algorithm

Figure 78 exhibits one of the messages published on the output topic. It is correctly indicated that it is a *Corner Green Column* and the colour ID is 4, which stands for green (see Table 4). This time the column is on the center of the table and slightly to the left which is shown by the *world_position_x* being -9 cm. In regards to the orientation, this being a corner column, the *pitch* (*world_orientation_y*) is also provided as the orientation of the joint is important information.

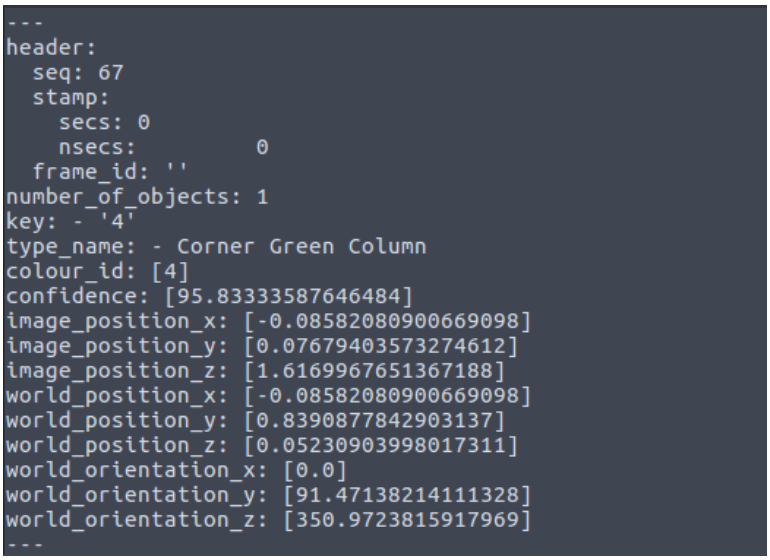


Figure 78: Results published on the ROS topic (https://youtu.be/bfLiaYDwCeA)

With the goal of testing the sturdiness of the software, the column was placed in 3 different positions and 10 samples were registered for each position. The position of the column was measured manually and the average error and larger deviation were calculated. Table 6 presents the results.

The average positional error for this case has values up to $\pm$ 2 cm while for *yaw* the error is around the $\pm$ 5° range and for *pitch* closer to the $\pm$ 10° mark. *pitch* also has deviation values higher than *yaw*.

Position	**Manually Measured Positions**					**Average Values Obtained**				
	x_m	y_m	z_m	yaw_m	$pitch_m$	$\bar{x}$	$\bar{y}$	$\bar{z}$	$\bar{yaw}$	$\bar{pitch}$
1	-10	87	4	350	90	-9,71	88,71	5,78	354,72	98,65
2	49	81	4	85	180	49,65	79,54	5,49	90,53	173,89
3	-25	63	6	200	0	-23,47	63,65	8,71	195,25	9,05
Position	**Average Error**					**Larger Deviation**				
	e_x	e_y	e_z	e_{yaw}	e_{pitch}	$\lvert\sigma_x\rvert$	$\lvert\sigma_y\rvert$	$\lvert\sigma_z\rvert$	$\lvert\sigma_{yaw}\rvert$	$\lvert\sigma_{pitch}\rvert$
1	-0,29	-1,71	-1,78	-4,72	-8,65	1,64	2,42	2,09	5,86	9,92
2	-0,65	1,46	-1,49	-5,53	6,11	1,92	4,29	2,78	10,56	14,47
3	-1,53	-0,65	-2,71	4,75	-9,05	3,60	1,34	3,71	7,62	12,20

Table 6: Summary of the sturdiness tests for the green column (all positions are in cm and all angles in degrees)

8.1.3 Columns in the air

Using the same methods, it is also possible to detect objects in the air being held by humans. It does not have the same level of effectiveness compared to when the objects are on the table and, consequently, the results are erratic and not all iterations produce useful output data. Nevertheless, it is possible to make use of the information if the receiver takes this into account and adapts its procedures accordingly by, for

example, using a structure with memory to accommodate and read the data, so that the iterations with unusable results can be inferred with previous iterations. Figures 79 and 80 offer an indication of what said results look like.

Figure 79: Raw frame received from camera

Figure 80: Column detection

Changeable shadowing and lighting conditions are the main issue that affects this aspect of detection. Unlike the cases where objects are on the table (a constant plane), the situation is considerably different when the objects are on the air. The data coming from the objects pose module continues to flow constantly, but the object distinction has difficulties extracting the proper edges of the column. As can be seen in the pairs of Figures 81 and 82 and Figures 83 and 84, the shadowing produces fictional edges in the middle of the body of the column. In some cases this occurrence affects the good attainment of the central rectangle (see Subsubsection 6.2.1.4) which is a vital piece in the unfold of the rest of the algorithm. As consequence, the results are less reliable.

Figure 81: Original Image

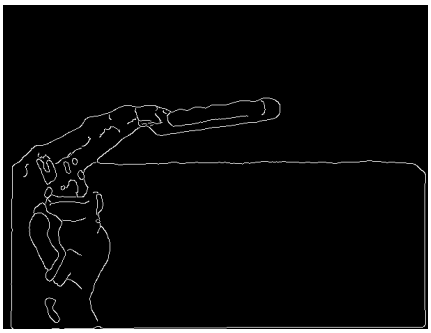


Figure 82: Edge extraction example No.1

Similar to the previous examples, the red column was held in 3 different positions and 10 samples were registered for each position. The position of the column was measured manually and the average error and larger deviation were calculated. Table 7 presents the results.

Figure 83: Original Image

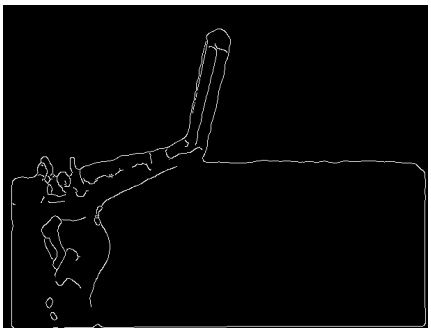


Figure 84: Edge extraction example No.2

The average positional error for this case has values up to $\pm$ 3 cm (1 cm more than the table example) while for *yaw* the error is around the $\pm$ 10° (double compared to the table example). The most evident difference of this case when compared to the previous two resides on the deviation values. These are much higher, specially for *yaw*.

Position	**Manually Measured Positions**				**Average Values Obtained**			
	x_m	y_m	z_m	yaw_m	$\bar{x}$	$\bar{y}$	$\bar{z}$	$\bar{yaw}$
1	-22	55	64	40	-25,34	85,2158,17	66,28	48,43
2	41	85	41	20	43,41	87,39	42,84	29,54
3	-8	86	57	90	-10,13	82,93	58,79	101,44
Position	**Average Error**				**Larger Deviation**			
	e_x	e_y	e_z	e_{yaw}	$\lvert\sigma_x\rvert$	$\lvert\sigma_y\rvert$	$\lvert\sigma_z\rvert$	$\lvert\sigma_{yaw}\rvert$
1	3,34	-3,17	-2,28	-8,43	13,92	10,60	22,64	-32,01
2	-2,41	-2,39	-1,84	-9,54	6,42	9,07	6,96	30,49
3	2,13	3,07	-1,79	-11,44	6,64	10,09	3,99	24,42

Table 7: Summary of the sturdiness tests for the red column in the air (all positions are in cm and all angles in degrees)

8.2 Testing various columns at the same time

This time, the small red column and three corner columns (green, yellow and orange) were laid on the table.

As mentioned in Subsubsection 6.2.1.5, it is possible to observe that the rectangle for the yellow column does not follow the same pattern as the others. Because said colour is extremely bright, the edges between the inner rectangle and the colour stripes are not solid enough. Nevertheless, the yellow colour is still correctly identified because the algorithm also takes into account part of the area inside the rectangle.

Figure 87 exhibits one of the messages published to the output topic. All four columns are correctly identified. The *Small Red Column* is the leftmost of all columns and has the highest *world_position_x*

Figure 85: Raw frame received from camera

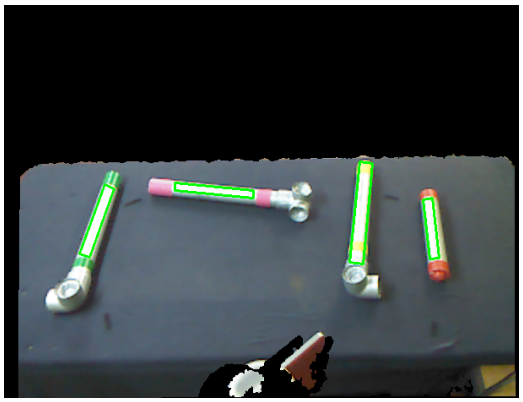


Figure 86: Column detection

(57 cm). In contrast, the *Corner Green Column* is on the other edge of the table (-64 cm). The *Corner Purple Column* is the one further away from the robot, with a *world_position_x* of 106 cm, approximately 15 cm more than the other columns.

```
---
header:
  seq: 121
  stamp:
    secs: 0
    nsecs:         0
  frame_id: ''
number_of_objects: 4
key: - '7'
- '1'
- '4'
- '3'
type_name: - Corner Purple Column
- Small Red Column
- Corner Green Column
- Corner Yellow Column
colour_id: [7, 1, 4, 3]
confidence: [94.64286041259766, 94.44444274902344, 94.04762268066406, 92.85713958740234]
image_position_x: [-0.1575842797756195, 0.5702253580093384, -0.640331506729126, 0.3224635720252991]
image_position_y: [-0.030868273228406906, 0.1152985543012619, 0.11431703716516495, 0.09671255201101303]
image_position_z: [1.7439820766448975, 1.6669285297393799, 1.623888611793518, 1.623937964439392]
world_position_x: [-0.1575842797756195, 0.5702253580093384, -0.640331506729126, 0.3224635720252991]
world_position_y: [1.0666861534118652, 0.9116195440292358, 0.8802914023399353, 0.8921077847480774]
world_position_z: [0.10598780959844589, 0.04892357811331749, 0.07845231145620346, 0.09150197356939316]
world_orientation_x: [0.0, 0.0, 0.0, 0.0]
world_orientation_y: [102.55984497070312, 0.0, 95.2187728881836, 192.4686279296875]
world_orientation_z: [354.47247314453125, 96.5197982788086, 250.30137634277344, 264.67425537109375]
---
```

Figure 87: Results published on the ROS topic (https://youtu.be/jjiYrZP59ak)

Table 8 offers a clearer summary of the information from Figure 87.

Column	**Position (cm)**			**Orientation (deg)**		**Confidence (%)**
	x_w	y_w	z_w	$pitch_w$	yaw_w	
Small Red	0,57	0,91	0,05	0	96,52	94,44
Corner Yellow	0,32	0,89	0,09	192,47	264,67	92,86
Corner Purple	-0,16	1,07	0,11	102,56	354,47	94,64
Corner Green	-0,64	0,88	0,08	95,22	250,30	94,05

Table 8: Summary of the pose of the columns

8.3 Testing during task

This stage of testing was done with the assembling of the structure in progress with the robot and a human working together. In order to facilitate this process for the demonstration and avoid obstructions, only the corner columns were assembled. This procedure is comprised by two stages. In each one the robot firstly moves its arm to the side so that the vision system can operate correctly and supply the data to the rest of the systems so that the robot can operate accordingly.

In the first stage, the yellow column is already in place and the other three are around the base. The software identifies and obtains information about those three columns. The purple column is placed by the robot, since its place is on the side of the workplace of the robot. For the same reason, the green column is placed by the human.

At the start of the second stage, the process is repeated with only the orange column yet to be placed. This robot is also responsible for the placement of this piece.

First Stage

As described above, Figures 88 and 89 present the beginning of the first stage of the task. The arm of the robot is not in the field of view, allowing for a clear image of the entire workspace.

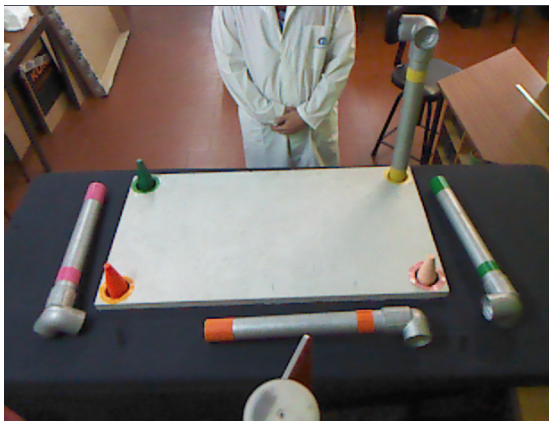


Figure 88: Raw frame received from camera

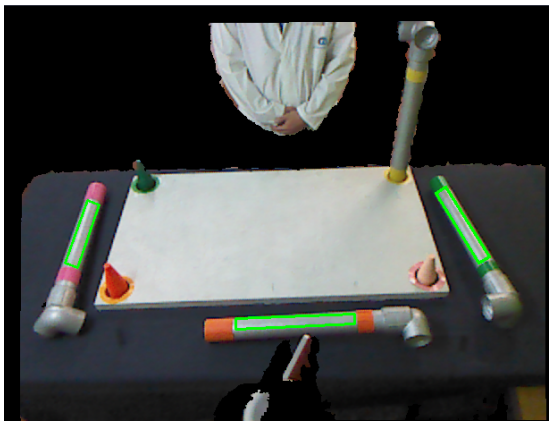


Figure 89: Column detection first stage

Figure 90 portrays one of the messages published on the output topic. The purple, green and orange columns are correctly identified, along their poses and a confidence variable. The yellow column already in place is ignored.

```
---
header:
  seq: 317
  stamp:
    secs: 0
    nsecs:         0
  frame_id: ''
number_of_objects: 3
key: - '7'
- '4'
- '2'
type_name: - Corner Purple Column
- Corner Green Column
- Corner Orange Column
colour_id: [7, 4, 2]
confidence: [94.64286041259766, 93.45237731933594, 92.26190185546875]
image_position_x: [-0.6921880841255188, 0.6124707460403442, 0.10521205514669418]
image_position_y: [0.153482586145401, 0.11783666908740997, 0.2928272485733032]
image_position_z: [1.5428954362869263, 1.5529110431671143, 1.364021897315979]
world_position_x: [-0.6921880841255188, 0.6124707460403442, 0.10521205514669418]
world_position_y: [0.79389488697052, 0.8251897096633911, 0.5677261352539062]
world_position_z: [0.10354164987802505, 0.12332998216152191, 0.11967814713716507]
world_orientation_x: [0.0, 0.0, 0.0]
world_orientation_y: [302.1499328613281, 1.001211166381836, 0.28316593170166016]
world_orientation_z: [251.0753631591797, 301.0716552734375, 3.468229293823242]
---
```

Figure 90: Results published on the ROS topic

Second Stage

In the second stage the orange column is the one remaining. The robot places its arm outside of the field of view and once more the developed software provides information about the objects on the table.

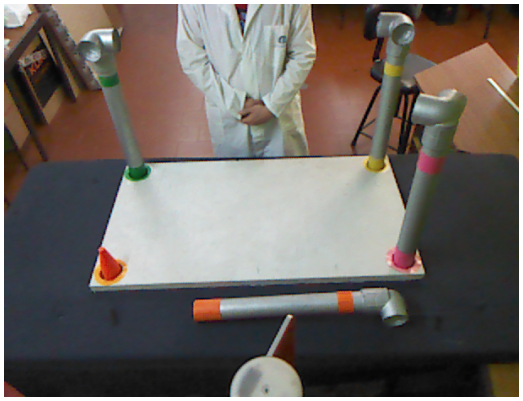


Figure 91: Raw frame received from camera

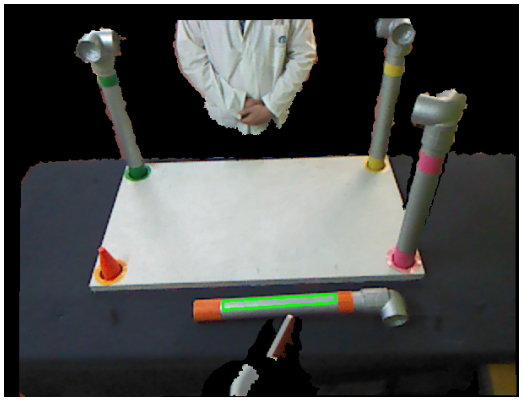


Figure 92: Column detection second stage

In similar fashion, the final output is shown in Figures 93. The data about the *Corner Orange Column* is consistent with the data from the First Stage, as the column stayed untouched.

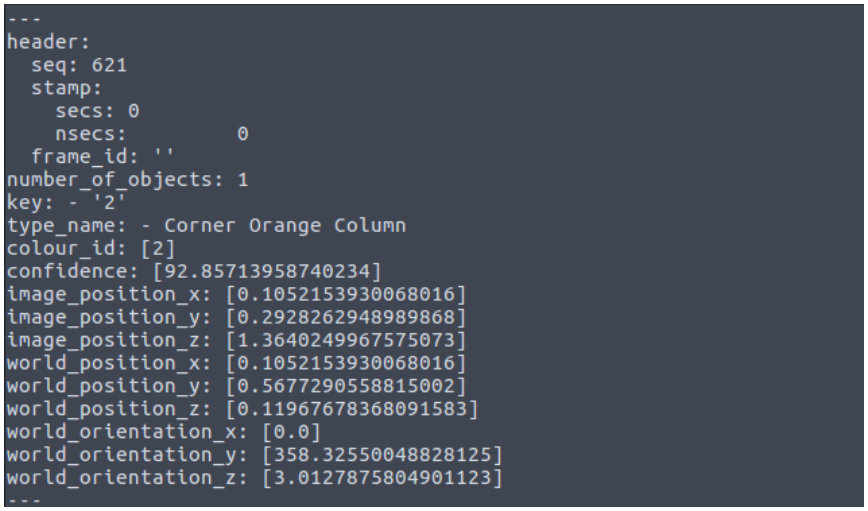


Figure 93: Results published on the ROS topic (https://youtu.be/W9stEeE7UPc)

Part V

Conclusion

Chapter 9

Discussion and Future Work

In this dissertation two very different problems were resolved using Computer Vision based technologies.

On one side, it was given to an autonomous stacker the ability to detected and extract the pose of a pallet by only analysing and manipulating frames from a depth camera using the OpenCV library. The approach produced quality results, capable of fulfilling the requirements and contributing to the normal operation of said vehicle when picking pallets. The developed solution also satisfied the conditions set in the beginning, that aimed for a solution as generic as possible, not depending on size, colour or identification markers in the pallet to run properly. A noteworthy limitation has surfaced where the extraction of the angle values may fail, more specifically when the initial values of α and β are too far away from the desired 0° and 90°, respectively. Nonetheless, the range of angle values where that limitation presents itself is outside the required for the picking manoeuvres in question. It is, however, something to keep in mind when trying to adapt the methods here portrayed to radically different setups.

As for the second part, regarding object detection, a system was developed having as foundation an external framework using template matching. The purpose of this task was to endow the Sawyer Robot with the ability to recognize objects used in assembling tasks and acquire their pose attributes. The results were good and the robot was capable of providing assistance during a task. Nevertheless, some limitations were exposed with both limited camera range and slow data update speed when many objects were present in the scenario. This can be an obstacle when performing tasks, as movements may have to be slowed down for the vision system to be able to keep up.

9.1 Future Work

Both parts of the dissertation present opportunities for improvement and development as future work.

For the pallet detection module, the sturdiness of the algorithm is the main point of interest. Obtaining more features of the pallet besides the depth discontinuities between columns allows for better methods of pallet validation (methods that ensure that what is being captured by the camera is indeed a *pickable* pallet).

For the object detection module, future work is predominantly related to performance issues. The software needs to be able to evaluate a larger area, which can be achieved by installing the camera on a pan-tilt. The search for objects would be done by steps, with the camera changing its orientation with every step and performing a sweeping motion along the work area. The column identification can also be refined, mainly in the extraction of contours, where a plethora of different techniques can be employed. This improvements would contribute for a better functioning of the column distinction when columns are in the air, as the main concern revolves around the variable shadowing and light conditions that affect the edge extraction.

From a broader point of view, the next step would be to design and implement a module for human gesture recognition and identification. The robot would have more information at its disposal, allowing it to also operate with intention based commands and expediting the collaborative assembling.

Bibliography

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 1st edition.

Burger, W. and Burge, M. (2016). *Digital Image Processing: An Algorithmic Introduction Using Java (Texts in Computer Science)*. Springer, 2nd ed. 2016 edition.

Byun, S. and Kim, M. (2008). Real-time positioning and orienting of pallets based on monocular vision. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 2:505–508.

Cai, Z., Han, J., Liu, L., and Shao, L. (2017). RGB-D datasets using microsoft kinect or similar sensors: a survey. *Multimedia Tools and Applications*, 76(3):4313–4355.

Chantara, W. and Ho, Y.-S. (2015). Object Detection based on Fast Template Matching through Adaptive Partition Search. *12th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, (December):1–6.

Chen, G., Peng, R., Wang, Z., and Zhao, W. (2012). Pallet recognition and localization method for vision guided forklift. *2012 International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM 2012*, 2:1–4.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:580–587.

Haanpaa, D., Beach, G., and Cohen, C. J. (2017). Machine vision algorithms for robust pallet engagement and stacking. *Proceedings - Applied Imagery Pattern Recognition Workshop*, pages 1–8.

Hinterstoisser, S., Lepetit, V., Ilic, S., Fua, P., and Navab, N. (2010). Dominant orientation templates for real-time detection of texture-less objects. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2257–2264.

I-SCOOP (2021). Industry 4.0: the fourth industrial revolution – guide to Industrie 4.0. https://www.i-scoop.eu/industry-4-0/. [Online; accessed 15-February-2021].

Kadambi, A., Bhandari, A., and Raskar, R. (2014). Computer Vision and Machine Learning with RGB-D Sensors. Advances in Computer Vision and Pattern Recognition. *Computer Vision and Machine Learning with RGB-D Sensors.*, pages 3–26.

Kanimozhi, S., Gayathri, G., and Mala, T. (2019). Multiple real-time object identification using single shot multi-box detection. *ICCIDS 2019 - 2nd International Conference on Computational Intelligence in Data Science, Proceedings*.

Keras (2019). Keras official website. https://keras.io/. [Online; accessed 20-November-2019].

Kim, J. U., Kwon, J., Kim, H. G., Lee, H., and Ro, Y. M. (2018). Object Bounding Box-Critic Networks for Occlusion-Robust Object Detection in Road Scene. *Proceedings - International Conference on Image Processing, ICIP*, pages 1313–1317.

LineMod (2021). LINE-MOD pipeline. https://wg-perception.github.io/ork_renderer/index.html#renderer. [Online; accessed 27-January-2021].

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., and Berg, A. C. (2016). SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37.

Mane, S. and Mangale, S. (2019). Moving Object Detection and Tracking Using Convolutional Neural Networks. *Proceedings of the 2nd International Conference on Intelligent Computing and Control Systems, ICICCS 2018*, (Iciccs):1809–1813.

Muthukrishnan, R. and Radha, M. (2011). Edge Detection Techniques For Image Segmentation. *International Journal of Computer Science and Information Technology*, 3(6):259–267.

OpenCV (2020). OpenCV official website. https://opencv.org/. [Online; accessed 08-September-2020].

Orbbec (2019). Orbbec official website. https://orbbec3d.com/product-astra-pro/. [Online; accessed 22-November-2019].

ORK (2020). Object Recognition Kitchen official website. https://wg-perception.github.io/object_recognition_core/#developers-corner. [Online; accessed 12-July-2020].

Phadnis, R., Mishra, J., and Bendale, S. (2018). Objects Talk - Object Detection and Pattern Tracking Using TensorFlow. *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2018*, pages 1216–1219.

Python Programming (2019). TensorFlow Object Detection Tutorial Python. https://pythonprogramming.net/training-custom-objects-tensorflow-object-detection-api-tutorial/. [Online; accessed 20-November-2019].

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv*.

RethinkRobotics (2021). RethinkRobotics official website. https://www.rethinkrobotics.com/sawyer/applications. [Online; accessed 27-January-2021].

ROS (2020). ROS official website. https://www.ros.org/. [Online; accessed 15-September-2020].

ROS pose (2021). ROS Pose Message. https://docs.ros.org/en/jade/api/geometry_msgs/html/msg/Pose.html. [Online; accessed 23-June-2020].

RotatedRect (2021). cv::RotatedRect Class Reference. https://docs.opencv.org/3.4/db/dd6/classcv_1_1RotatedRect.html. [Online; accessed 09-June-2021].

Sawyer Technical (2021). Sawyer Technical Specifications. https://www.rethinkrobotics.com/fileadmin/user_upload/sawyer/rr-blackedition-brochure_low.pdf. [Online; accessed 27-January-2021].

Skansi, S. (2018). *Introduction to Deep Learning: from logical calculus to artificial intelligence*. Springer.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications (Texts in Computer Science)*. Springer, 2011 edition.

Takahashi, M., Ji, Y., Umeda, K., and Moro, A. (2020). Expandable YOLO: 3D Object Detection from RGB-D Images. *2020 21st International Conference on Research and Education in Mechatronics, REM 2020*, pages 3–5.

TensorFlow (2019). TensorFlow official website. https://www.tensorflow.org/. [Online; accessed 18-November-2019].

TensorFlowObjectDetection (2019). TensorFlow Lite Object Detection. https://www.tensorflow.org/lite/models/object_detection/overview. [Online; accessed 21-November-2019].

Toyoya Technical (2021). Toyota SPE200DN Technical Specifications. https://www.toyota-industries.com.ar/Especificaciones/Catalogo-Tecnico-Toyota-BT-Staxio-P-series-SPE200DN.pdf. [Online; accessed 26-February-2021].

Xiao, Q., Hu, X., Gao, S., and Wang, H. (2010). Object detection based on contour learning and template matching. *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, pages 6361–6365.

Yata, K., Koutaki, G., and Uchimura, K. (2015). Image matching by eigen template method for multi-class classification. *2015 Frontiers of Computer Vision, FCV 2015*, pages 0–3.

Zhang, X., Zhao, J., and Lecun, Y. (2015). Character-level convolutional networks for text classification. *Advances in Neural Information Processing Systems*, 2015-January:649–657.