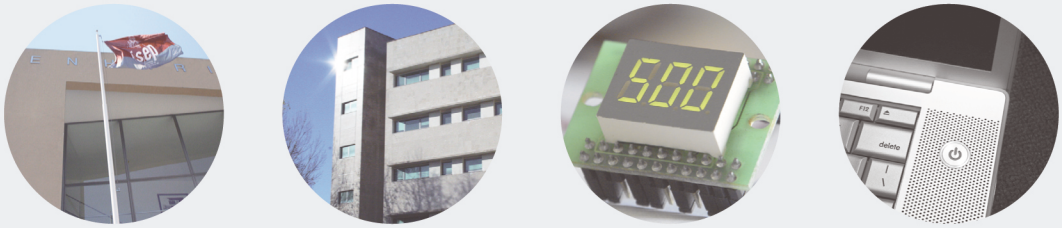
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



# Deep learning for bin picking object segmentation

**ARTUR JOSÉ VILARES CORDEIRO**
julho de 2022

POLITÉCNICO
DO PORTO

# Deep learning for bin picking object segmentation

**Artur José Vilares Cordeiro**


**Mestrado em Engenharia Eletrotécnica e de Computadores**
Área de Especialização de Sistemas Autónomos
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2022

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Artur José Vilares Cordeiro, Nº 1160952, 1160952@isep.ipp.pt
Orientação científica: Manuel Silva, MSS@isep.ipp.pt
Empresa: INESC TEC
Supervisão: Luís Rocha, luis.f.rocha@inesctec.pt & Carlos Costa, carlos.m.costa@inesctec.pt


**Mestrado em Engenharia Eletrotécnica e de Computadores**
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
July 20, 2022

# Acknowledgments

I would like to thank my supervisor, Manuel Silva, for all the attention given and the helpful ideas shared, especially with tasks and report organization.

I am also grateful for all the assistance, knowledge and attention provided by INESC TEC, especially Luís Rocha and Carlos Costa, helping me in several circumstances, allowing me to gather experience in different fields and giving me full support when needed.

Last but not the least, I would like to thank my family, friends and girlfriend for all the support throughout the dissertation work.

# Resumo

*Bin picking* tendo como base técnicas de *deep learning* é uma tecnologia que adota um conceito recente para melhorar as abordagens padrões utilizando redes neuronais. Estas abordagens estão cada vez mais a ser aplicadas em áreas *bin picking* devido ao potencial das técnicas de *deep learning* para melhorar a inteligência e as capacidades de aprendizagem de robôs.

Esta dissertação propõe uma técnica robusta e precisa para diversos objectos a granel usados em *bin picking*, aplicando uma *framework* com configuração fácil de ajustar de acordo com o objetivo e objetos pretendido. A *framework* é implementada em Robot Operating System (ROS) e é dividida em dois sistemas, de deteção e segmentação. O sistema de deteção recorre à rede neuronal de segmentação de instâncias Mask-RCNN para estimar as diversas máscaras, identificando e segmentando diversos objetos através de uma imagem de duas dimensões (2D) em tons de cinza. O sistema de segmentação utiliza como base *point cloud library* (PCL), manipulando informação de 3D *point clouds* conforme os resultados da deteção para selecionar os pontos específicos da *point cloud* original, gerando uma *point cloud* parcial como resultado. Além disso para completar o sistema de *bin picking* são implementadas abordagens de estimação de posição e localização baseado em algoritmos de correspondência, como Iterative Closest Point (ICP), assim como estimação de *grasp*.

A implementação da deteção e segmentação foi testada em vários datasets para os dois objetos propostos ($90^{\underline{o}}$ tubo em cotovelo e suporte de parede triangular) utilizando um Photoneo PhoXi S 3D *scanner* (sensor RGB-D), e um computador composto por um processador Ryzen 5 5600x, placa gráfica Gtx 1060 6GB e memória 32GB DDR4, exibindo uma precisão de 80%, *recall* de 92.25%, Intersection Over Union (IOU) de 89% e pontuação F1 de 89%. Executando o processo de deteção e segmentação, este consome entre 0.52 segundos e 0.85 segundos. A implementação completa foi realizada num robô industrial com um manipulador UR10, uma plataforma móvel e o mesmo sensor RGB-D, e esta incluí estimação de posição, localização e de "grasping" para além da framework

explicada de deteção e segmentação.

A precisão observada e a capacidade de detetar objetos a granel em diversos ambientes desordenados possibilita a implementação do sistema em ambientes diversificados propícios para *bin picking*, ambientes quais são bastante observado em cenários de indústria, permitindo assim a deteção de diferentes objetos através de modelos específicos Mask-RCNN previamente treinados.

**Palavras-Chave**: *Bin picking*, *Deep learning*, Mask-RCNN, *Grasping*, ROS, Redes neuronais, Inteligência artificial.

# Abstract

Bin picking based on deep learning techniques is a technology that applies a relatively recent concept, utilizing neural networks, intending to enhance the standard bin picking approaches. These approaches are increasingly being applied in bin picking areas due to the potential of deep learning policies to further increase the intelligence and learning capabilities of robots.

This dissertation proposes a robust and accurate technique for random bin picking objects, employing an easy configuration to adjust the framework according to the intended object. The framework is implemented in Robot Operating System (ROS) and is divided into a detection and segmentation system, using Mask-RCNN instance segmentation neural network to handle the main detection system, identifying and segmenting several objects from a two dimensions (2D) grayscale image, and point cloud library (PCL) as a basis of the segmentation system, manipulating 3D point cloud data according to the detection results to select particular points of the original point cloud, generating a partial point cloud result. Furthermore to complete the bin picking system is employed a pose and grasping estimation approach based on matching algorithms, such as Iterative Closest Point (ICP).

The detection and segmentation implementation was tested in several datasets for two proposed objects ($90^{o}$ elbow tube and triangular wall support) using a Photoneo PhoXi S 3D scanner (RGB-D sensor), and a desktop computer composed by a Ryzen 5 5600x processor, Gtx 1060 6GB graphic card and 32GB DDR4 of memory, exhibiting high Average precision (Ap), Average recall (Ar), Intersection Over Union (IOU), and F1 score. Executing the full process consumes between 0.53 seconds to 0.85 seconds. The full implementation, including pose and grasping estimation, was tested in a real robot with a UR10 manipulator, a mobile platform and the same RGB-D sensor.

The accuracy and capability of detecting objects in several cluttered environments provides the possibility of implementing the system in distinct bin picking environments, as in industry 4.0, allowing to detect of different objects with spe-

cific Mask-RCNN trained models, requiring only a neural network training stage to generate these models.

The master thesis here presented was conducted under the European Union's Horizon 2020—The EU Framework Programme for Research and Innovation 2014–2020, under grant agreement No.101006798—Project Mari4_YARD.

**Keywords:** Bin picking, Deep learning, Mask-RCNN, Grasping, Neural networks, Artificial Intelligence, ROS.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| Abbreviations | Descrição | Página |
|---|---|---|
| ROS | Robot Operating System | 3 |
| AI | Artificial Intelligence | 5 |
| SVM | Support Vector Machine | 5 |
| RL | Reinforcement Learning | 5 |
| DL | Deep Learning | 5 |
| CNN | Convolutional Neural Networks | 8 |
| G-RCNN | Graph Recognition Convolutional Neural Network | 9 |
| RCNN | Region Based Convolutional Neural Networks | 9 |
| RNN | Recurrent Neural Networks | 9 |
| LSTM | Long Short Term Memory | 10 |
| RGB | Red, Green, Blue | 17 |
| 3D | Three-Dimensions | 18 |
| DCN | Deep Convolutional Networks | 19 |
| CAD | Computer Aided Design | 21 |
| ICP | Iterative Closest Point | 22 |
| RGB-D | Red, Green, Blue, Depth | 22 |
| LfD | Learning from Demonstration | 24 |
| KNN | k Nearest Neighbours | 25 |
| 6D | Six-Dimensions | 26 |
| YCB | Yale-CMU-Berkeley | 27 |
| GQ-CNN | Grasp Quality Convolutional Neural Network | 20 |
| 2D | Two Dimentional | 28 |
| Dex-Net | Dexterity Network | 20 |
| COCO | Common Objects in Context | 12 |
| AP | Average Precision | 12 |
| OOI | Objects-Of-Interest | 35 |
| CUDA | Compute Unified Device Architecture | 141 |
| API | Application Programming Interfaces | 141 |
| iid | independent and identically distributed | 45 |
| GPU | Graphics Processing Unit | 142 |
| CPU | Central Process Unit | 142 |

| Abbreviations | Descrição | Página |
|---|---|---|
| TPU | Tensor Processing Unit | 142 |
| MSE | Mean Squared Error | 142 |
| HBM | High Bandwidth Memory | 143 |
| MMU | Matrix Multiplication Unit | 143 |
| FLOPS | FLoating-point Operations Per Second | 143 |
| PCL | Point Cloud Library | 147 |
| GUI | Graphical User Interface | 54 |

# Chapter 1

# Introduction

Humanity is driven by a curiosity about the unknown and the desire of doing something new. One of the big advantages is that we do not settle with comfort or with rudimentary survival activities, and that is the basic reason why we are always evolving to be a better version of ourselves. The word ”better” has numerous meanings and contradictions, but in this dissertation is focused on technological areas.

In this day and age, humanity is evolving to be more automated and independent, turning basic day to day activities into full automated systems, that require barely non-supervision. These activities are specially prospering in industry and car fields, where there is a need to simplify or rather minimize human interactions, improving efficiency, precision and the working set of hours.

The industry currently has several automated areas, that are developed to automate simple human labour, for example, transporting or picking items from place to place. In the automation and robotics field this problem is many times referred as the bin-picking problem, where an object is picked from a bin to another specific location. This procedure is commonly presented in industry scenarios and is already mainly implemented in an automated form, presenting different methods and solution patterns.

This dissertation is aimed towards bin picking in clutter environment problems, where a robot manipulator can detect unique or different types of objects, and generate the necessary process to pick up the object from its original location to a different given position. In this case, the basic framework to detect an object is based on deep learning, which is a technique noticed by its extraordinary growth, as observed by the latest papers published in the bin picking area.

Deep learning is an area still new to the industry scene due to the lack of information. In the modern days, several state of art approaches and big industry

names are using this technology, making it interesting to study its possibilities.

## 1.1 Contextualization

This problem arose due to the need of developing a bin picking framework, capable of detecting and picking several objects with unknown positions and orientations in a cluttered environment, moving objects from a bin into another bin (place in the platform of the robot).

INESC TEC already has a full solution to solve this problem, integrated in Mari4_yard project, based on clusters manipulation and features detection, a robust framework capable of detecting objects in controlled environments [37] [31].

This framework has limitations, one of them being the detection phase, where the algorithm can have problems detecting and identifying different objects in highly cluttered environments. With the recent development and attention towards machine learning, it was decided to develop a framework based on deep learning techniques capable of generating 3D object detection. This was the initial idea of the project; in between were discussed many different ideas that could be interesting to be developed. These new ideas brought new problems and several interesting implementations, some of them implemented and described in this dissertation.

The implementation is also integrated on the European project (Mari4_yard), however it demonstrates another type of bin picking perception solution. Essentially, was a test to analyse the deep learning potential in an industry bin picking field. If, in the end, the proposed framework displayed promising results, it was considered a success, enabling a deeper study of this technology.

## 1.2 Motivation

This project is interesting due to several features, being one of the biggest reason deep learning. Deep learning is an interesting but quite unknown area to explore. First, this technology has a great potential because essentially is an artificial brain that learns with simple information, and in the future it can be easily trained and implemented by humans with low knowledge in this area or even by another machine, for example another neural network (already developed to efficiently train neural networks). Furthermore, it is simplifying and surpassing several complex methods, as observed in self-driving systems, displaying great improvements over the years. Unlike any other technique, it requires a self-learning stage, where the neural network will learn patterns and structures from data, processing results that occasionally are not expected, from a better and

worst point of view, this way producing new thought processes that are not fully static.

Additionally, this work is a industrial robotic problem, where it is possible to work with the hardware and software of industrial robots. To conclude, the operating system that the implementation was being applied was Robot Operating System (ROS), which is an attractive environment to work on.

## 1.3 Objectives

The initially proposed objective was to implement a detection framework for bin picking problems based on deep learning techniques. The first initial objective was to generate the initial dataset for a tubular model and train a neural network to detect these objects.

In succession, the next objective was to develop an algorithm (based on ROS action procedures) capable of sending pointcloud2 messages with the detected cloud.

An extra objective was trying to implement the framework on a working robot to pick and place objects from a bin to another bin, in a real scenario.

## 1.4 Schedule

The Gantt chart illustrated in Figure 1.1 summarizes the work planning developed for this dissertation.

| ID | Task Name | Start | Finish | Duration |
|---|---|---|---|---|
| 1 | State of Art study | 15/01/22 | 25/02/22 | 5.8w |
| 2 | Article development | 10/2/2022 | 4/4/2022 | 7.5w |
| 3 | Dataset generation | 15/02/22 | 5/5/2022 | 2.5w |
| 4 | Mask-RCNN training | 24/02/22 | 2/6/2022 | 4w |
| 5 | Implementation | 14/03/22 | 28/05/22 | 10.7w |
| 6 | Evaluation and results | 27/5/2022 | 14/06/22 | 2.5w |
| 7 | Extra work | 10/4/2022 | 22/04/22 | 1.7w |
| 8 | Thesis development | 1/2/2022 | 25/06/22 | 20.6w |

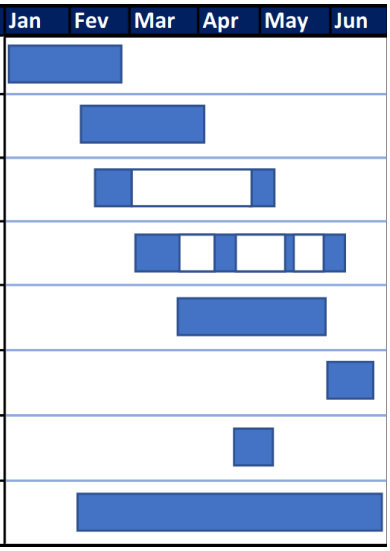



Figure 1.1: Gantt chart detailing the dissertation work plan

The blue horizontal rows represents the days elapsed developing the specific task. The white space between these rows represents an interval of pausing the task, where no further development was done.

## 1.5 Dissertation organization

In Chapter 2, are presented state of art approaches and methods utilized in bin picking based on deep learning techniques, and are also described and detailed different concepts to give a better understanding of this field.

In the next Chapter, 3, are detailed several software applications that were utilized during the dissertation development.

Chapter 4 begins detailing concepts important to enhance the development of the implementation. Furthermore, is described the main implementation, and all the methods and information necessary to implement the proposed object segmentation based on deep learning techniques framework.

In the succeeding Chapter, 5, are presented the proposed implementation evaluation with deep learning metrics, and are demonstrated several results acquired from the deep learning framework in different stages (detection, segmentation).

In Chapter 6, are detailed two extra works developed during the dissertation, allowing future work opportunities.

At last, Chapter 7 concludes the dissertation, describing the work developed, the fulfilled and unfulfilled objectives and future works proposals.

# Chapter 2

# Bin picking methodologies

Bin picking is a highly researched topic, due to the need for automated procedures in industrial environments. A general bin picking system requires a highly structured process, starting with data acquisition, and ending with pose estimation and grasping. A high number of bin picking problems are being presently solved, through deep learning networks, combined with distinct techniques.

## 2.1 Artificial Intelligence

Artificial Intelligence (AI) is the science and engineering field of making intelligent machines (therefore the term machine learning is a sub-field of AI), especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable [38]. In 1956, at the Dartmouth Artificial Intelligence Conference, the technology was described as "Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." [39].

Recent AI successes are largely attributed to new Machine Learning techniques that construct models in their internal representations. These include Support Vector Machine (SVM), random forests, probabilistic graphical models, Reinforcement Learning (RL), and Deep Learning (DL) neural networks [40]. So the basic difference between ML and DL is that a machine learning processes is designed to choose the best function from a set of implemented functions, to best explain the relationship between features, but in deep learning the entire process is automated by a deep neural network, and the patterns that deep learning algorithms extracts from datasets are functions that are represented as neural networks [41] (Figure 2.1).

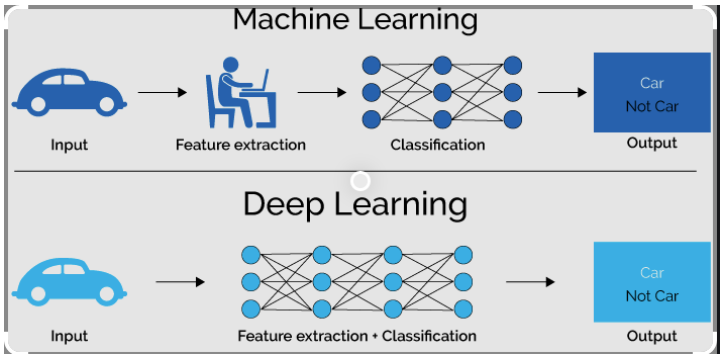



Figure 2.1: Comparison between Machine Learning and Deep Learning [1]

In 2015 the program AlphaGo based on machine learning was the first computer to defeat a professional human Go player, a Go world champion and arguably the strongest Go player in history. Go or Weichi is a traditional Chinese abstract strategy game, in which the aim is to surround more territory than the opponent. In 2014, DeepMind, the company that created the AI program was bought by Google. Nowadays this AI is considered one of the best in the world, not only executing predictions of Go problems, as well other world wide problems, such as predicting the weather, how to navigate without a map, self learning on how to walk, and many more.

### 2.1.1 Deep Learning

Deep learning has historical roots going back decades. LeCun *et al.* [42] or Krizhevsky *et al.* [43] generalized the back-propagation algorithm for training multilayer nets, which was the foundation of the Deep Learning revolution of the 2010s [44]. This AI subfield (Deep Learning) focuses on creating large neural network models that are capable of making accurate data-driven decisions, and is particularly suited to problems evolving complex data where there are large datasets available [41], extracting this way the full potential of this learning technique.

Conventional machine-learning techniques require careful engineering and considerable domain expertise to design a feature extractor that transforms the raw data into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input. Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification [2]. This dissertation is based on deep-learning methods based on representation-learning methods with multiple levels of representations.

Machine Learning is a large field with different learning forms, they are selected according to the context of the problem being solved. Among all the types of learning, like supervised, unsupervised, reinforcement, multi-task, among others, the supervised learning is the most common. In this learning technique during training, the machine is shown an image and produces an output in the form of a vector of scores, one for each category [2]. During training the machine modifies its internal adjustable parameters (weights) to reduce the error between the real and the measured score. To adjust the weight vector, the algorithm computes a gradient vector that, for each weight, indicates by what amount the error would increase or decrease if the weight was increased by a tiny amount. The weight is then adjusted in the opposite direction of the gradient vector. After training, the performance of the system is measured on a different set of examples called a test set. This serves to test the generalization ability of the machine, its ability to produce sensible answers on new inputs that it has never seen during training [2]. An example of a neural network used for computing the forward pass with two hidden layers and one output layer is presented in figure 2.2.

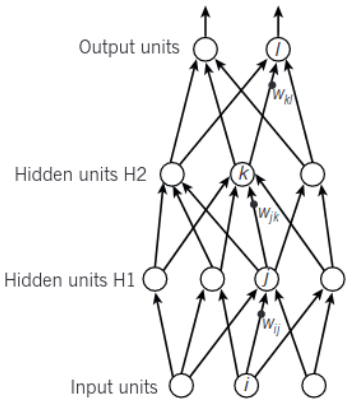



Figure 2.2: Equations used for computing the forward pass in a neural net with two hidden layers and one output layer [2]

Fundamentally, a supervised learning model is based on the operator providing input, output and feedback to build a model [45]. Unsupervised learning models establish that the conclusions and patterns are determined through unlabeled training data, instead the model learns from itself, discovering its own information about the solution. Reinforcement learning model is based on a system of rewards and punishments learned thought trial and error, seeking maximum reward, like the one developed in [16].

#### 2.1.1.1 Convolutional neural networks

Convolutional neural networks (CNN) are a specialized kind of neural network for processing data that has a known, grid-like topology. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Basically convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in, at least, one of their layers [3].

The architecture of a typical CNN is structured as a series of stages (typical of three stages), that are usually composed of two types of layers: convolutional and pooling layers. The purpose of a convolutional layer is to detect local conjunctions of features from the previous layer, and the role of the pooling layer is to merge semantically similar features into one [2]. The layer in the first stage performs several convolutions in parallel to produce a set of presynaptic activations. In the second stage, also called detector stage, each presynaptic activation is run through a nonlinear activation function, such as the rectified linear activation function. In the third stage is implemented a pooling function to modify the output of the further layer [3].

Pooling helps to make the representation become invariant to small translations of the input (Figure 2.3). The use of pooling can greatly improve the efficiency of a network, if the layer function learns to be invariant to small translations by adding an infinitely strong prior value [3], as seen in figure 2.3, the network detects a value, and produces a polling value based on a set with both variables, the value detected and the prior value defined.

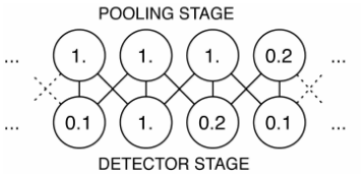



Figure 2.3: Max pooling [3].

Convolutional neural networks are designed to process data that come in the form of multiple arrays, for example a colour image composed of three 2D arrays [2]. There are many data modalities with different dimensions and number of channels, and some of the most common data types in convolutional networks are presented in Table 2.1.

In the early stages, CNN were more targeted to resolve speech recognition and document reading with time-delay neural networks, majorly developed by Microsoft. But over the years were also experimented for object detection and face

Table 2.1: Convolutional neural networks data modalities [2],[3].

| | **1D** | **2D** | **3D** |
|---|---|---|---|
| **Single Channel** | Audio waveform | Audio spectrograms | Volumetric data |
| **Multi Channel** | Skeleton animation data | Color image data | Color video data |

recognition. Since the 2000 CNN had a great success with detection, segmentation and recognition of objects and regions in images, which all processes had the particular factor of having abundant labelled data. Currently a major recent practical success is face recognition [2]. Most recently, in 2021, Cheng *et al.* [46], developed a Graph Recognition Convolutional Neural Network (G-RCNN), based on the detection algorithms created throughout the years, for example Region Based Convolutional Neural Networks (RCNN) (2014), Fast RCNN (2015), Mask R-CNN (2017). This method is able to recognizes graph structures from images [46].

Convolutional networks have played an important role in the history of deep learning, especially in bin picking. As informed in the following sections, traditional grasping methods have shown poor performance except in scenarios in which the parts and grasping contact points are well defined [44]. Therefore techniques were developed based in deep learning, in this case CNN.

#### 2.1.1.2 Recurrent neural networks

Recurrent neural networks (RNN) are the main tool for handling sequential data, involving variable length inputs or outputs [3]. These networks were mainly trained using the back-propagation method [2], where the procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired one. As a result of the weight adjustments, internal hidden units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods, such as the perceptron-convergence procedure [47].

In order to convert multi-layer networks to recurrent networks, it is essential to take advantage of parameters sharing procedures, an early idea found in machine learning, across different parts of a model. The idea of parameters sharing made possible the extension and application of the model to examples with different forms, like length, and generalize across them; essentially, the weights are re-used at each time step. This is particular important when input sequences can be

stretched non-linearly, such as vowels which may last longer in different samples. Compared to a multi-layer network, the weights in an RNN are shared across different instances of the artificial neurons, each associated with different time steps [3]. Figure 2.4 illustrates the unfolding in time of the computation involved in RNN forward computation, where the left circuit represents a hidden-to-hidden recurrence, and the same circuit is represented on the right as a time-unfolded flow graph, where each node is now associated with one particular time instance [3].

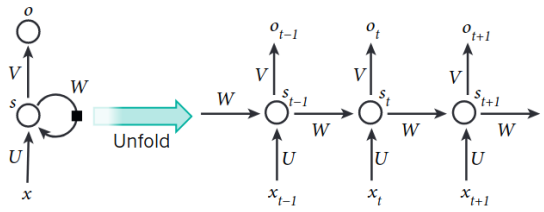



Figure 2.4: Recurrent network circuits [2].

Despite being a very powerful system, training RNN is problematic because the backpropagated gradients either grow or shrink at each time step, so over many time steps they typically explode or vanish [2]. Thanks to the advances of training and architecture, the RNN can be used for predicting the next character in the text, and more complex tasks. For instance, in 2015 Lopyrev [48] developed an application of an encoder-decoder RNN with long short term memory (LSTM) units to generate headlines from the text of new articles, which performed better than the more complex attention mechanism on a held out set of articles.

Recurrent neural networks have been generalized into recursive neural networks, in which the unfolded structure can be more general than a chain, bidirectional RNN and deep RNN.

#### 2.1.1.3 Deep learning segmentation

As described, CNN are networks used for several techniques in the area of deep learning. As seen in Figure 2.5, different object-level networks can provide different type of output information, so they have different objectives, like Object Recognition, Class Segmentation, Object Detection and Object Segmentation [4]. Presently, there are a large number of DNN with distinct objectives, apart from the ones described.

In the current section, is going to be reviewed deep learning segmentation, a core method in deep learning approaches. Image segmentation has been a fundamental problem in computer vision, with the objective of partitioning images,

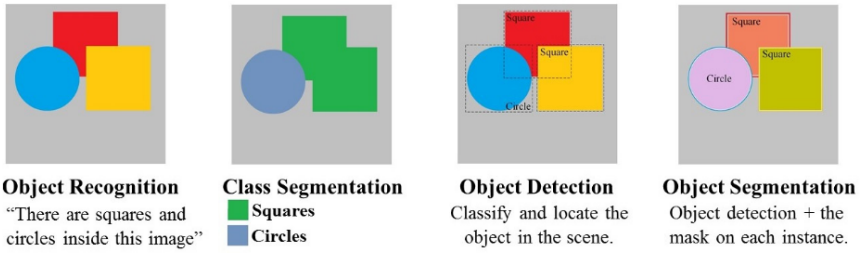



Figure 2.5: Different output information of object level scenarios [4].

video frames, or point clouds, into multiple segments and objects, and plays a central role in a broad range of applications [49]. Deep learning segmentation can be divided into three major topics of object level types, such as Instance segmentation and Semantic segmentation. The basic difference between these approaches can be seen in Figure 2.6.

### Instance segmentation

As observed, the main difference between these segmentation methods in a bin picking problem, is that one of these approaches, Instance segmentation, can differentiate the same type of object or a living being when they are overlaid or in a clutter, as seen for example with the cows in Figure 2.6. The singular identification produced by instance segmentation brings a great advantage relatively to semantic in cluttered environments with the same types of objects.

Instance segmentation generated a lot of interest in object-oriented bin picking, more precisely in analytical approaches, and is going to be the main focus of deep learning in this dissertation. Instance segmentation is aimed at finely delineating object instances in images, without explicit object models [35].

This type of segmentation differs from image classification, semantic segmentation and instance detection, where the first two presented methods aim to assign image-level and pixel-level object categories respectively, and instance detection consists in coarsely localizing instances by approximating them as rectangles [35]. Instance segmentation combines elements from the classical computer vision tasks for object detection, where the goal is to classify individual objects and localize each one using a bounding box, and semantic segmentation, where the goal is to classify each pixel into a fixed set of categories without differentiating object instances [25].

The segmentation method, called instance semantic segmentation, implements a multi model approach. This method focuses on detecting bounding box and object category as well as predicting the predefined category for each pixel in the

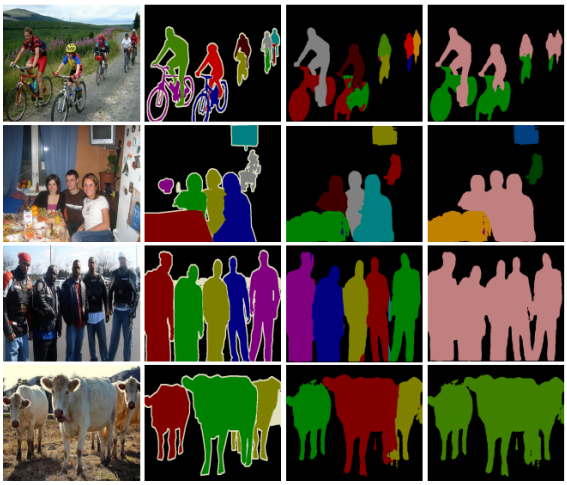


Figure 2.6: Different output information of networks (Left to Right): 1º Input images, 2º Ground truth, 3º Instance segmentation, 4º Semantic segmentation [5].

bounding box. The main idea of instance semantic segmentation is the combination of object detection and semantic segmentation using CNN [45]

In Table 2.2 are presented several frameworks based on Instance segmentation utilizing Common Objects in COntext (COCO) dataset [50]. The frameworks are classified from best to worst Average Precision (AP) with the same data and conditions. Between these presented, TensorMask and Mask R-CNN or frameworks based on RCNN are the most common ones, with a largest amount of instance segmentation articles written with their employment.

Table 2.2: Object instance segmentation frameworks based on COCO dataset [34]

| **Frameworks** | **Average Precision (AP)** | **Year** |
|---|---|---|
| Hybrid Task Cascade | 43.9% | 2019 |
| PANet | 42.0% | 2018 |
| SOLOv2 | 41.7% | 2020 |
| BlendMask | 41.3% | 2020 |
| SOLO | 40.4% | 2019 |
| CenterMask | 38.3% | 2019 |
| MaskLab+ | 38.1% | 2017 |
| TensorMask | 37.3% | 2019 |
| Mask R-CNN | 37.1% | 2017 |
| PolarMask | 32.9% | 2019 |

In 2018, He *et al.* [25] presented their method, a conceptually simple, flexible and general framework for object instance segmentation, as Mask R-CNN. Basically was an extension of Faster R-CNN [51] by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. Regarding bin picking, there are several articles based on this framework. For example, in 2019 Tuan-Tang Le and Lin [4] developed a method for detecting planar objects in a cluttered environment. This study applied an instance segmentation based on a deep learning approach (Mask R-CNN), using 2D image data for classifying and localizing the target object while generating a mask for each instance. This process is demonstrated in Figure 2.7 using universal serial bus (USB) packs with two different layers, front and back.


Figure 2.7: Results: (Left) Ground truth, (Right) Prediction [4]

Peichen Wu *et al.* [6] made use of a similar network, Faster R-CNN [51]. This framework is a further development of a previous RCNN, Fast R-CNN. In this implementation the Region Proposal Network (RPN) is merged with Fast R-CNN into a single network by sharing their convolutional features. Peichen work focused on a grasping plan and selection in a clutter scene. Firstly in the selection point of view, the objects were segmented from the input picture using the Faster R-CNN model as described. For planning the correct grasping was combined the geometry information of the point cloud and the color information.

It is possible to see a representation of Peichen work on Figure 2.8 and also different implementations (works) based on instance segmentation.

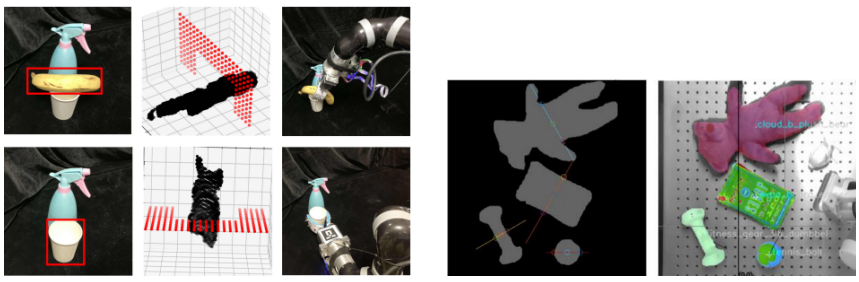


Figure 2.8: Instance segmentation approaches: (Left) Faster R-CNN[6], (Right) Mask R-CNN[7]

### Semantic segmentation

Semantic segmentation as previously detailed is another deep learning segmentation technique. Unlike instance segmentation this approach does not distinguish among objects of the same class, as displayed in Figure 2.6, for example, if it is presented an image with four cars, it is going to be generated four masks around the cars with the same instance, not identifying the difference between each car.

Chungang Zhuang developed a Part Mask RCNN based on the original Mask R-CNN, a framework proposed to predict category, bounding box, object mask and object-part in the RGB image [45].

The framework is based on semantic segmentation and pose estimation. Semantic segmentation is applied to find the object boundaries in the scene, estimating the segmentation of every object in the scene and the object classification derived from semantic labels. Afterwards is estimated the object 3D pose related to the semantic labelled scene point cloud [45].

## 2.2 Bin picking

The dominant labor force from 1950 encounters a lot of challenges, such as productivity demands, space utilization, labor availability, and as a result of these challenges, labor force is decreasing at a fast pace.

Alongside all the problems emerging from the advance of technology, bin picking is one of them. It made an appearance more than 50 years ago [52], and is still a a highly researched problem in the current time. Bin picking is a discipline which combines several investigation areas, such as scene analysis, object recognition, object localization, grasp planning, and path planning [52]. Some of the steps that where referred are going to be addressed in the following sections, identifying different approaches and selecting the ones that are meaningful for this study.

In the majority of cases, the objects are randomly arranged in the bin, that's why is also called random bin picking, and for this case the commercial products implemented in the modern industry can only solve one particular case. There is no single bin picking application, but rather a spectrum of specific situations based on any number of unique constraints [53]. This can be a peculiar object, environment and process. That's why, currently, the methods are growing in the direction of making a robust solution, that can work with almost every product of the industry, besides having speed, efficiency and accuracy [53].

Many methods were developed directed towards this problem, the most recent ones are based in deep learning techniques, like the ones that are going to be mentioned in this report.

### 2.2.1 Robot grasping

Grasping is widely explored in the robotic research area, but there are still many problems to be solved because of the variety of environments and the uncertainty of perception and execution [24]. Some of this main concerning problems are trying to be solved with state of art frameworks, similar to the one presented in this dissertation. The approach developed has the objective of developing a deep learning method that resolves at cluttered industrial scenarious, composed by one object variety in the bin, helping with the advance of bin picking in the industrial environment.

Grasping is the act of taking hold of some object by some kind of manipulator, in our case usually is the human hand. Robot grasping is the act of grasping by a robotic manipulator, which may have quite different designs [44]. There are mainly four types of robotic grippers namely, vacuum, pneumatic, hydraulic and servo-electric grippers. Each one of them can have diverse configurations (as depicted in Figure 2.9) , such as two fingers, five fingers (human hand), vacuum cup, and many others, influencing the motion planning of the robot given its objective. The operation of grasping relies on the type of grippers, in conclusion, with different end effectors, outcomes different grasping strategies.


Figure 2.9: Robot End Effectors [8] (Left to Right): (Top) Human hand and 4 Finger hand, (Middle) 3 finger hand and Parallel gripper, (Bottom) Vaccum gripper

### 2.2.2 Bin Picking approaches

Robotic grasping can be categorized along a set of multiple different criteria [54]. It can be divided into different domains, observed in Figure 2.10. Bin picking is divided into two main topics, that are Gripper Oriented and Object Oriented, each analysed in the next sections.

Gripper oriented techniques lack the notion of instance, meaning that the approach doesn't identify the different objects presented in the workspace, which is important for handling occlusions in dense piles of instances. It can be subdivided into unsupervised and supervised learning. Object oriented relies either on notion of pose or on generic instance segmentation techniques, being divided into model based and model free methods. Supervised and unsupervised sub-areas depend on the input given to the network. As the name suggests, supervised, is a method where the model is administered by an exterior entity, for example a labeled dataset to train algorithms to predict certain outputs. On the other hand, unsupervised models, work on their own, in order to discover information related to the objective, for example self-learning networks.

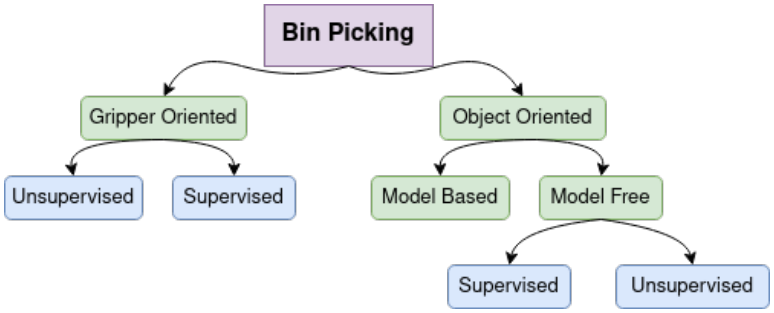



Figure 2.10: Bin Picking approaches.

Different approaches based on neural networks are typically designed for diverse purposes of bin picking, just like the ones described in this chapter. Some methods that developed one of the most known networks are identified in Table 2.3, with the respective bin picking approach and the year they were developed.

#### 2.2.2.1 Gripper Oriented

Gripper oriented methods consist in detecting grasp opportunities with respect to the robot end effector physics, like the few presented in Figure 2.9. Between the five grippers exhibited, the most used in bin picking, in the industry area, are still the Parallel and Vaccum grippers. This is due to their simple movement, caused by the small number of actuators, in this case only one or two actuators are needed for their control. Other factor is the wide study and utilization of these grippers in grasp detection methods [35].

Table 2.3: Bin picking methods based on Neural Networks for different objectives [35]

.

| | Gripper Oriented | Object oriented (Model based) | Object Oriented (Model free) | Year |
|---|---|---|---|---|
| **MaskRCNN [25]** | | | ✓ | 2017 |
| **FasterRCNN [51]** | | | ✓ | 2015 |
| **RCNN [55]** | | | ✓ | 2014 |
| **AmodalMask [56]** | | | ✓ | 2017 |
| **YOLO v1,v2,v3 [57, 58, 59]** | | | ✓ | 2016,2017,2018 |
| **MaskLab [60]** | | | ✓ | 2018 |
| **DeepMask [61]** | | | ✓ | 2015 |
| **U-Net [62]** | | | ✓ | 2015 |
| **DeepLabV3+ [63]** | | | ✓ | 2018 |
| **SegNet [64]** | | | ✓ | 2017 |
| **Line-MOD [65]** | | ✓ | | 2013 |
| **Lee *et al.* [13]** | | ✓ | | 2018 |
| **DexNet(1,2,3,4) [18, 11, 66, 67]** | ✓ | | | 2016,2017,2018 |
| **Jacquard [19]** | ✓ | | | 2018 |
| **Lenz *et al.* [68]** | ✓ | | | 2018 |

Vacuum grippers (Figure 2.2.1) are highly reliable in industry environments. When the gripper comes in full contact with the surface of an item, creates a vacuum area, by removing the air between the closed pad and the item surface, which generates negative pressure [69]. As a result, it allows to pick items with pinpoint accuracy, meaning that the implemented method can produce only one pick up point in the best possible position, and reduces the risk of picking several items simultaneously, though the types of picked items will be limited by the mesh that wraps the object.

Parallel grippers, or two-finger grippers (Figure 2.2.1), are able to pinch and pick items even if air passes through their surface. Although the gripper has the advantage of picking up many kinds of items, there is a catch [69]. In contrast to vacuum or suction grippers, the pose of the gripper must be determined while taking into consideration the collision with nearby items. As a result, the implementation requires higher accuracy and processing methods.

In the early approaches grasp detection frameworks employed unsupervised heuristic methods on red, green, blue (RGB) and depth images to detect the best locations for parallel grippers or best locally planar areas for vaccum cups. Some

heuristic methods are going to be described to explain the basis of distinct non machine learning methods.

### Heuristic methods

In real world grasping, adopting gripper oriented approaches can be difficult, due to the full three-dimensions (3D) shape of the object is hard to perceive. To solve this problem, in early stages Miller *et al.* [9] used heuristic rules to generate and evaluate grasps for three fingered hand, modeling an object as a set of shape primitives, such as spheres, cylinders, cones and boxes. To analyze and visualize the grasps of a variety of different hands and objects was created a grasping simulator called "GraspIt" this tool is an interactive simulator, which can import a wide variety of hands and robot designs, and a world populated with objects, within a 3D workspace. As a result of the progress provided by this simulator, it lead to an increase popularity in model free approaches.

Different from other approaches, like the ones proposed by Markenscoff [70] and Ponce [10], it is a generic method, basically works on different type of grips (Figure 2.11), on a workspace with distinct objects, and a large set of grasp positions measured on single primitives (Figure 2.12). In order to calculate the best grasp it is defined a metric function that analyses the grasp quality of the measurements.

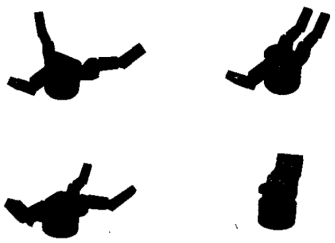


Figure 2.11: Grasp preshapes for the Barret hand spherical, cylindrical, precisian-tip, and hook grasps [9]

In case of a polygon object, Markenscoff and Papadimitriou presented an approach followed by the Baker ones, that achives a grasp planning of a polygon by a four finger gripper [70], instead of a three fingers one, as developed by Baker [71]. According to Baker, Fortune, and Grosse the smaller the forces the robot hand must exert on the object, the better the grip. Minimizing the necessary forces seems the natural criterion. Large forces would mean unnecessary stress and deformation of both the object and the robot hand and a higher deformation energy. Minimum force pretension seems to coincide with the intuitive idea of a firm or good grip of an object [70]. Essentially, in this work was developed an

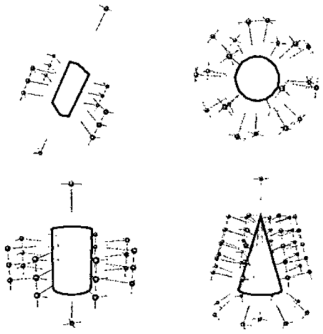


Figure 2.12: Grasp generation on single primitives [9]

optimization technique to the problem of optimizing a stable grip of any polygon object, determined using a mix of optimization and Euclidean geometry [70].

Similar to the process just discussed, Ponce and Sullivan, addressed the problem of characterizing the force-closure grasps of a three-dimensional object by a hand equipped with three or four hard fingers (Figure 2.13). As represented each finger is placed at an edge of the polyhedral objects, maximizing that way the efficiency of the grasp. In summary, they present an efficient, output-sensitive algorithm for computing the projection and an algorithm using linear programming for computing maximal grasp regions [10].

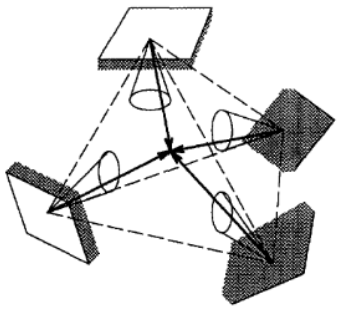


Figure 2.13: A 3D four-finger force-closure grasp [10]

The greatest advantage of the heuristic methods is that they provided absence of complication in terms of planning and the needless of knowing the object complete model, but on the other hand, the majority of these methods are particular for a specific form of objects, like polygons, rectangles, circles.

In the later stages, for more complex gripper oriented methods, ranking heuristics based grasp candidates was boosted by deep convolutional networks (DCN). Some of these methods based on DCN are going to be briefly described,

in order to enhance the several different implementations based on gripper oriented methods.

The majority of the methods with best results and with more depth were developed since 2017. One of the most popular articles is Dexterity Network (Dex-Net) 2.0 [11] by Mahler. A Grasp Quality Convolutional Neural Network (GQ-CNN) model resulted from this work, that rapidly predicts the sucess probability of grasps from depth images. Grasps are specified with a planar position, angle, and depth of a gripper relative to an RGB-D sensor [11]. The pipeline of this network can be identified in Figure 2.14.

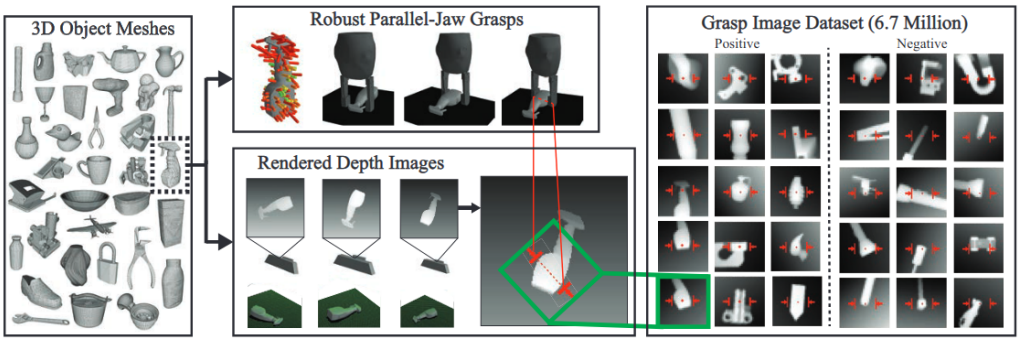



Figure 2.14: Dex-Net 2.0 pipeline for training dataset generation: (Left) The database contains 1,500 3D object mesh models. (Top) Parallel-jaw grasps to cover the surface and evaluate robust analytic grasp metrics using sampling. (Bottom) Rendered point clouds of each object in each stable pose. (Right) Each image is rotated, translated, cropped, and scaled to align the grasp pixel location [11].

Besides the developed neural network (GQ-CNN), that was just reviewed, in 2016 the same research team made an extensive synthetic dataset of 6.7 million point clouds, grasps, and analytic grasp metrics generated from thousands of 3D models for bin picking solutions in Dex-Net 1.0 [18], providing worldwide open source data.

#### 2.2.2.2 Object Oriented

Object-oriented bin-picking aims to locate affordable instances independently of the gripper model. In such a perspective, the disparity between object and gripper oriented approaches is that, object oriented is strongly related to the perception of occlusions, while gripper oriented approaches is more related to friction forces and torques of the gripper [35].

Object oriented approaches are divided into two categories: Analytic or Model based methods and Data-driven or Model Free methods. Analytic methods as-

sume an explicit model of the target, while Data-driven methods are mostly driven by image segmentation techniques, like image segmentation neural networks [35].

### Analytic (Model Based) approach

Analytic approaches (also called geometric) analyze the shape of a target object to identify a suitable grasp pose. Frequently are based on certain information about the models, like points of contact, Coulomb friction, and rigid body modeling. These characteristics are needed so it can be formulated a constrained optimization problem based on geometric, kinematic and dynamic formulations [72]. In conclusion, this approach is a model based method, that depends on the model, like a Computer Aided Design (CAD) model, given to solve the considered task. Many methods that apply this approach are based on traditional point cloud processing techniques, but can also be based on deep learning techniques. For instance neural networks such as Hinterstoisse *et al.* with LineMOD [65] and Lee *et al.* [13] implement this approach, also referred in Table 2.3.

Model-based robotic grasping can be considered as a three-stage process where first object poses are estimated, then a grasp pose is determined, and finally a collision-free and kinematically feasible path is planned towards the object to pick it [54].

As mentioned in the previous sections, when utilizing object-specific knowledge, approaches typically require an object-specific configuration, which means that a high amount of manual tuning and computational power is required until the system reaches a good performance accuracy ratio [54]. Consequently, this approach has the advantage of considering the geometric values of the objects but, in return, requires a high amount of time and computation power to solve the problem.

Several approaches with complex object models emerged using voting based learning strategies, generally based on feature clustering and depth based methods. Pochyly [73] developed a method based on a modified revolving vision system, using linear laser projectors and a SICK Ranger camera. In this approach a gripper was designed for a specific object, with the purpose of bin-picking testing for these particular objects. The software presents the bin model , the collision model of the end-effector and suitable candidates for grasping based on the model of the object, though this approach is quite time consuming and expensive.

Also with the same fundamental approach, Birdal formulated a method with a revised pipeline of the existing 3D object detection and pose estimation framework based on point pair feature matching [12]. This framework proposed to represent 3D target objects using self similar point pairs, and then matching the model to a 3D scene using efficient Hough like voting scheme operating on the reduced pose parameter space. Though it lead to great results, it had some

shortcomings related to the high dimension of the search space. Birdal addressed these drawbacks and proposed a new solution, coupling the object detection with a coarse-to-fine segmentation, applying weighted Hough voting during matching, and removing Iterative Closest Point (ICP) [12]. The pipeline is illustrated in Figure 2.15, where firstly an input CAD model is trained using ambient occlusion maps as weighting, then each scene is segmented into smaller regions and matched to the trained model, per each segment is retained hypothesis and verified using the rendered CAD model, finally hypothesis is ranked according to the scores.

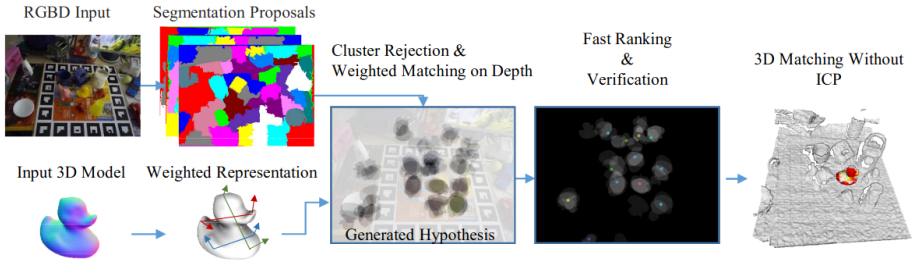



Figure 2.15: Proposed pipeline [12]

In the modern days, the vast majority of analytic methods incorporate trained deep convolutional networks for representation learning. In 2018 Lee proposed a method to estimate the 3D pose of an object using red, green, blue, depth (RGB-D) camera. The method consists in two modules: object detection by deep learning, and pose estimation using the ICP algorithm [13]. It's also proposed a depth based filtering method to improve the precision of object detection by preprocessing input data, and applied a plane fitting method in the second module to increase the accuracy of the estimated pose (Figure 2.16).

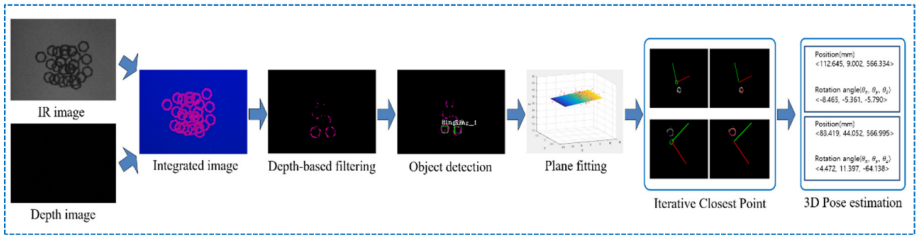



Figure 2.16: Lee *et al.* proposed approach [13]

Generally, analytical approaches face two limitations in bin picking. They require explicit object models that are not always available, and the notion of pose is only defined for rigid objects, because a rigid object can maintain the same shape when moved, making it possible to estimate its final pose [35].

#### Data-driven (Model Free) approach

Data-driven approaches (also called empirical) are based on machine learning [54], and gained popularity in recent years. These methods sample grasping candidates for unknown objects and rank them according to a certain metric [72]. Unlike analytic methods, they don't need a model, like a CAD model or a previously scanned model, being the only requirements labels collected by humans or labelling processes, physical trial and error, heuristic methods, or a process based on demonstration by humans or a developed robot, described in the next sections. Therefore, it is considered a model-free approach. Usually, it directly proposes grasp candidates and typically aims for a generalization to novel objects [54].

These approaches can be interesting in bin picking and robot grasping challenges, like the one organized by Amazon called Amazon Picking Challenge (APC). In 2017, the Massachusetts Institute of Technology (MIT) Princeton Team presented an approach to solve warehouse pick and place problems, concerning cluttered environments and self-occlusions. They developed a self supervised, data-driven system based on RGB-D data. Firstly the views of the scene were segmented by a fully convolution neural network, secondly a pre-scanned 3D object model was fit in the scene to then get a 6D object pose [14].

With the same context and very similar to the case just presented, Le [4] and Wu [6], developed in 2019 methods to adress bin picking [4] and predicting grasps [6], based on deep learning networks. Typically these approaches (data-driven) consists in firstly identifying the several objects in the scene, creating a region proposal or a segmentation, secondly obtaining the 3D scene, for example by a point cloud, and lastly estimating the object pose.

The three cases are represented in Figure 2.17, where it is possible to notice the high versatility and accuracy of data-driven approaches.

### 2.2.3 Learning methods

Bin picking based on Deep Learning techniques is a common topic presently explored, due to the advance of neural networks, mentioned in the previous sections. As referred, these networks require a previous training before their application on a bin picking problem, otherwise they are going to be completely lost with the intended objective. There are several learning methods, just like the ones mentioned in the sequel, and these will be differentiated based on the required learning time, objective and accuracy.

It's going to be briefly described the three typical learning methods, associating some works that implemented them. Despite all of them being regularly

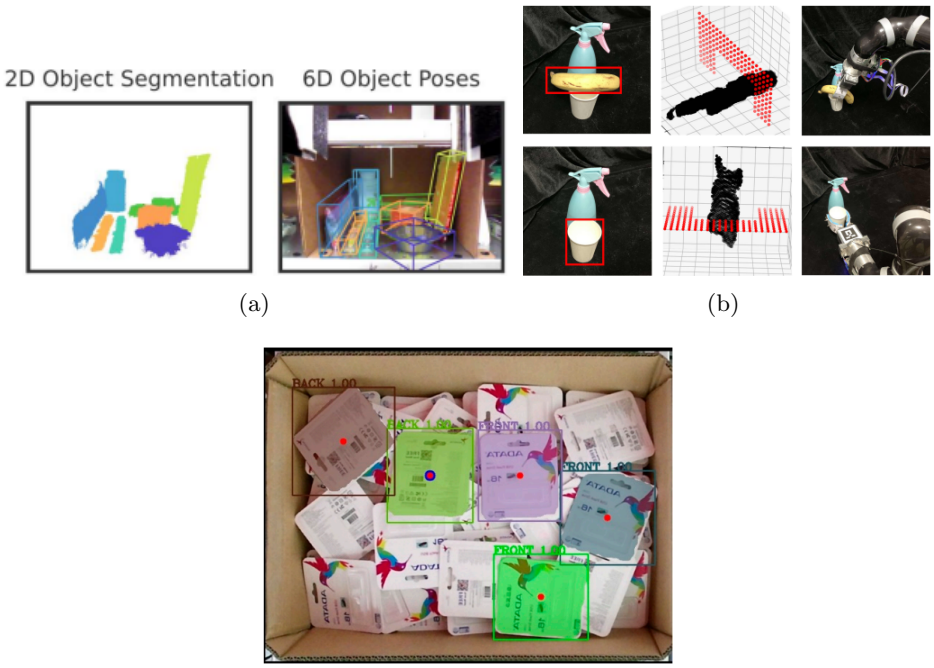



Figure 2.17: Data-driven approaches. (a) Warehouse pick and place[14] (b) Predicting Grasping Order[4] (c) Bin Picking for Planar Objects[6]

used in exceptional projects, labelling is mainly utilized due to the practicality and the results that provides.

#### 2.2.3.1 Demonstration process

One of the most naive methods from the user viewpoint, and hardest to implement from a programmer point of view, due to uncertainty in friction and push mechanics and the variety of objects encountered [15], is the demonstration process. As the name indicates, this process is based on demonstration by a human supervisor or an already developed robot for grasping objects [9]. Laskey trained a Deep Neural Network based on this concept in order to grasp a desired object in a cluttered situation, such as an Amazon warehouse. This network, combined with Online Learning from Demonstration (LfD) algorithms, such as DAgger and SHIV, empowers the robot to learn control policies for such tasks where the input is a camera image, and the system dynamics [15].

Apart from implementing the demonstration method, Laskey article's (Figure 2.18) was the first study of hierarchical supervisors for Learning from Demonstration. They deduce from physical experiments that with 160 expert demonstrations, the probability of a successful grasp can increase from 55% to 90% [15].

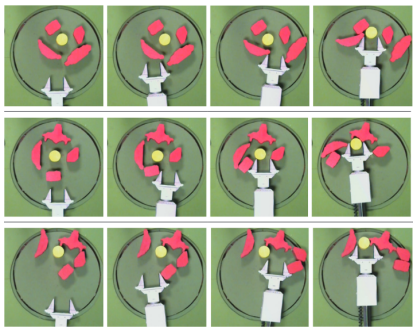


Figure 2.18: Laskey *et al.* demonstration process [15]

#### 2.2.3.2 Reinforcement learning (Trial and error)

Trial and error is an uncommon method to train neural networks, due to the low efficiency and high quantity of hours required to produce accurate decisions. It was developed to decrease the human labor involved in the labelling of the extremely lengthiest dataset. This technique usually prones the learner to overfit, in behalf of the length of the dataset. So, in 2015, Pinto [16], increased the training data to 40 times more than the previous work, leading to a dataset of 50000 data points collected over 700 hours of robot grasping attempts. To achieve this goal was used a Convolutional Neural Network (CNN), with pretrained layer from ImageNet Alexnet. The action of learning by trial and error is explained in the (Figure 2.19). In the end the results provided by this approach had a better result than linear Support Vector Machines (SVM)[74] and k Nearest Neighbours (KNN) algorithms [16].

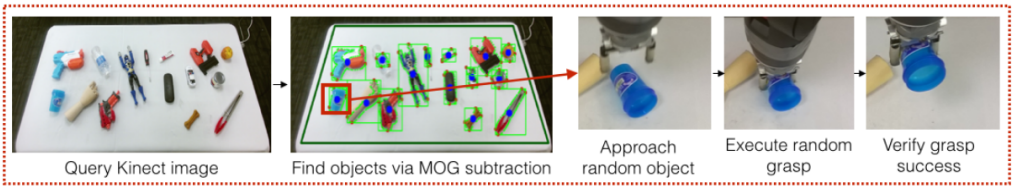



Figure 2.19: Overview of how random grasp actions are sampled and executed, so it can self learn from these samples [16]

In conclusion, trial and error its an easy to use method, because only needs the prior work of applying other heuristic methods or hand feeding grasp possibilities. Although deppending on the workspace and the hole sceene, it can take a considerably high amount of time.

#### 2.2.3.3 Labelling

Labelling learning is a method based on sucessful or unsucessful labels collected by humans or physical trials [72].

Generating the labels can be done with different techniques. The most common and simple one, is human labelling. This method can be implemented utilizing a third party software, like LabelMe [20], RecLabel [75], VGG Image ANootator [76], among many others, or can be manually implemented by indicating the characteristics of the label. However in order to generate grasp candidates with a high accuracy, are commonly used known datasets, like Cornell [77], Dex-Net Dataset [78], Google Grasp Dataset [79] and Jacquard [80].

Essentially, the annotation relies on the type of data that is being labeled. If, as referred, the approach is using known datasets, usually the annotations are based in depth and RGB images, in most cases taken by a RGB-D camera. For instance we have Kumra proposition [17] that adopted Jacquard [80] and Cornell [77] Datasets to tackle the problem of generating and performing antipodal robotic grasps for unknown objects (Figure 2.20).

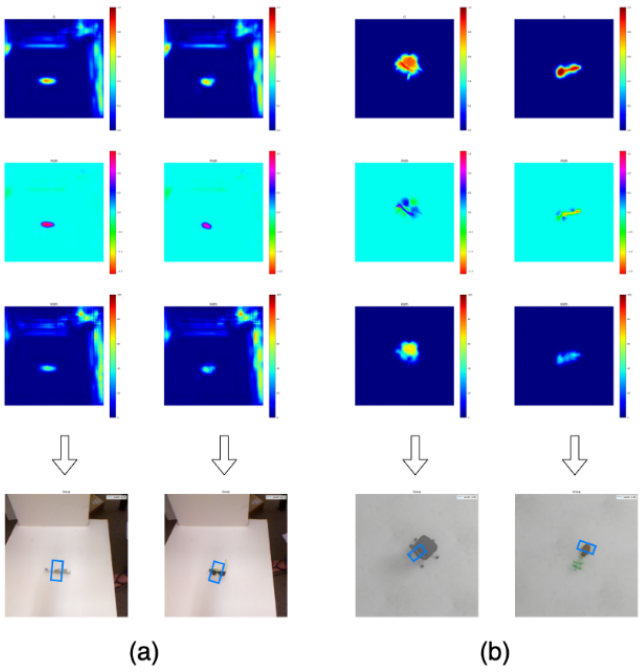



Figure 2.20: Qualitative results (Quality, angle, width and grasp by order): (a) Unseen objects from Cornell dataset (b) Unseen objects from Jacquard dataset [17]

Xiang [81] estimated a six-dimensions (6D) object pose in cluttered scenes

based on a large scale video dataset called Yale-CMU-Berkeley (YCB) [82]. Despite the employment of a model to define the final pose, the key ideia was developing a CNN capable of detecting and performing segmentation of the visualized objects. Shin [7] took advantage of the YCB dataset and developed a deep learning-based object recognition and robot manipulator for unkown objects, estimating the direction of the object, center point of the segment and the edge points by drawing straight lines from the center, as represented in (Figure 2.21).

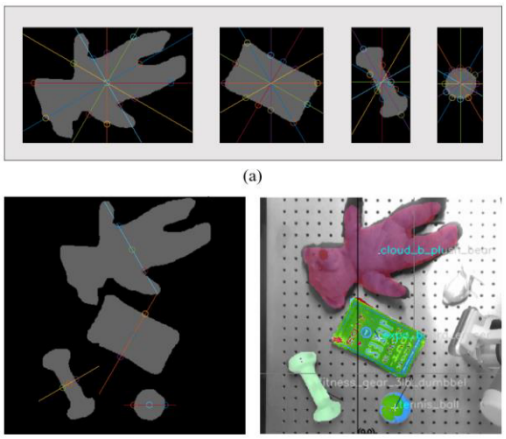



Figure 2.21: Estimation of direction of object to grasp [7]

In alternative of generating a dataset produced by real depth and rgb images and labelled by human hand or algorithms, or adopting developed datasets, that is high time consuming and is sensitive to human errors or mistakes, like the methods referred above. Is possible to create and label synthetic data, such as Dex-Net 1.0 dataset.

Dexterity Network 1.0 is a dataset of 3D object models based on a sampling-based planning algorithm to explore point clouds for robust grasp planing, where the grasp prediction is observed in Figure 2.22 it contains 6.7 million synthetic point clouds, grasps, and analytic grasp metrics generated from thousands of 3D models. Each grasp includes an estimate of the probability of force closure under uncertainty in object and gripper pose and friction [18].

Similar to the Dex-Net achievement, Depierre *et al.* created a large scale synthetic dataset with ground truth for Robotic Grasp Detection (Jacquard). Jacquard is built on a subset of ShapeNet (large CAD models dataset), and contains both RGB-D images and annotations of successful grasping positions based on grasp attempts performed in a simulated environment [19], providing practicable means of testing different algorithms, in most articles, due to the extensive dataset.

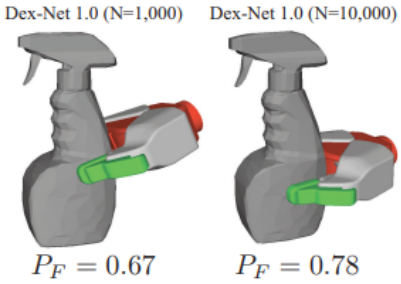



Figure 2.22: Grasp predictions: (Left) 100 prior objects on the spray bottle (Right) 10 000 prior objects [18].

Figure 2.23: Jacquard dataset: Objects with multiple labeled grasps on images[19]

In addition to all the methods discussed in this section remains the most simple ones. Instead of labelling depth and RGB image or point cloud data, the approach can be addressed on a planar field, by only requiring two dimensional (2D) data (RGB or grayscale image) (Figure 2.24). In this case the most common procedure is using a labeling software, as referred in the beginning of this section.

## 2.3 Data acquisition

Data acquisition is the acquiring of information of a real or simulated world from different sensors to generate data. Choosing these sensors is a delicate task that will affect the whole process in general. In this case the data acquired will most certain be a 2D image and a depth map.

Deep Learning action recognition techniques can be classified as RGB data, depth data, skeleton sequences and methods using a combination of these data modalities (multi-modal) [83].

Figure 2.24: Annotation of a 2D image by labelMe [20]

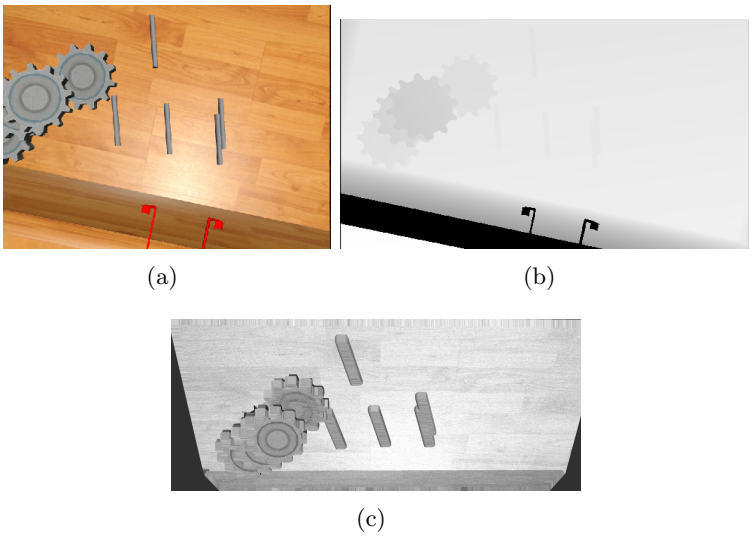
The popular RGB data, observed in Figure 2.25 (a), is a 2D array that defines red, green, and blue colour components for each pixel. Defined in the same way by a 2D array, grayscale data only represents the intensity information of light of each pixel. In contrast, a depth image, demonstrated in 2.25 (b), is a 2D array where each pixel is defined by the distance values between the image plane and the corresponding object measured by image techniques.

These sensors may also acquire point clouds, defined by a group of points manifested in the same 3D system ($x$,$y$,$z$). It is worth noting that a 2D image represented with a depth map(image), can produce a point cloud, basically a 3D data representation (multi-modal). This fact can be observed in Figure 2.25 where the data acquired by a RGB-D sensor, in a simulated world, produces a point cloud based on a RGB image and a depth map. Skeleton sequences aren't commonly applied in bin picking approaches because they are mainly used to recognize human actions using 3D skeleton joints recovered from 3D depth data [84].

Analysing the several techniques, there is not a better one for all the scenarios, if the scene needs high data acquisition frequency time-of-flights is probably the best sensor, otherwise if the scene is static for a few seconds, structured light is the better one, because of the high resolution it can produce. Concluding, it is all going to depend on the type of scenario and variables presented.

The majoraty of works focusing on deep learning approaches in a bin picking environment, take advantage of RGB-D cameras, but can also be used sensors like LIDAR, RGB cameras, hyperspectral cameras or more complex 3D sensors that provide a scene reconstruction [35], although they are not so commonly used.

There are several techniques to acquire 3D data using a RGB-D sensor. Table 2.4 summarizes these approaches based on different technologies. RGB-D data has two main methods of depth sensing: passive range sensing, where the sensors



(a) (b)

(c)

Figure 2.25: Simulated data acquisition. (a) RGB image (b) Depth map (c) Point Cloud

don't modify the scene to obtain the scene depth, and active range sensors, referring to the fact that the sensors modify the scene to simplify the reconstruction problem [36].

### 2.3.1 Structured light

Structured light is a technique that projects patterns upon a target, inside a specific range according to the sensor limitations. According to the shape presented, the displayed patterns will have distortions and displacements, enabling the sensor to estimate the size, shape or position of the object.

One of the earliest accounts of structured light was by Thomas Young in 1803, when he demonstrated one-dimensional (1D) intensity structured light. His observation of fringe structures demonstrated the wave nature of light, with important information encoded in both the density and direction of the fringes. These patterns are the basis for a technique which generates new waves with different patterns from two or more overlapped waves, that can be used to discover differences between the input waves, this science or technique is also named interferometry [85].

Structured light techniques are very similar to stereo vision, described in 2.3.2, although instead of two cameras, the system has one camera and one projector, as illustrated in figure 2.26. The projector emits light patterns upon the objects in the environment, eliminating the problem of relying on the texture variation in the object being scanned. This way the projector can project its own special textures, or patterns, onto the object [21].

Table 2.4: RGB-D Range sensing techniques [36]

| Techniques | Sensors | Description |
|---|---|---|
| Passive Range Sensing | Stereo or Binocular | Depth is estimated by a triangulation process. Pixels in the two images that observe the same 3D position in the scene are compared between the two camera views. |
| Active Range Sensing | Time of Flight | Measures the time of flight of an emitted light pulse. The emitted pulse than reflects back on the object and returns to the initial sensor. Depth=$\frac{Speed of Light\ x\ Travel Time}{2}$. Another type of sensors use a time-modulated light pulse and measures the phase shift between the emitted and returning pulse. |
| | Structured Light | Similar to stereo is based on triangulation, and the key idea is to replace one of the cameras in a stereo system with a projector. A unique structured pattern is projected into the scene to add artificial features. |

In order to establish the relation between the camera and the project, there are several structured pattern encoding that can be applied. These different forms of encoding an object are 2D Pseudorandom Codifications, Binary Structured Codifications, N-ary Codifications, Triangular Phase Codifications, Trapezoidal Phase Codifications, and Continuous Sinusoidal Phase Codifications, described in detail in [21].

### 2.3.2 Stereo vision

Stereo vision is a method of acquiring depth information, based on the relative position of the objects in the different sensors [86], it can be used for 3D reconstruction, scene analysis and other depth-related usage [22]. In the human case, these sensors are the eyes, that is why is possible to have depth perception, if the human body had only one eye, it can not precisely estimate depth because we do

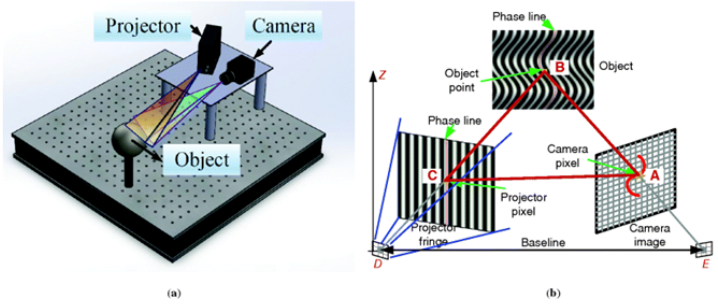



Figure 2.26: Structured light illustration [21]

not have the information and relation between the sensors.

A similar approach can be applied to cameras, mainly for static ones. Unlike our sensors, stereo vision information acquired with cameras can be classified into two categories: monocular camera with known scene information; and dual-camera system (standard procedure), which is based on techniques of stereo rectification and matching [22], demonstrated in figure 2.27. The basis of this technique is to find for each point in one image (left or right), the corresponding point in the other image [87].

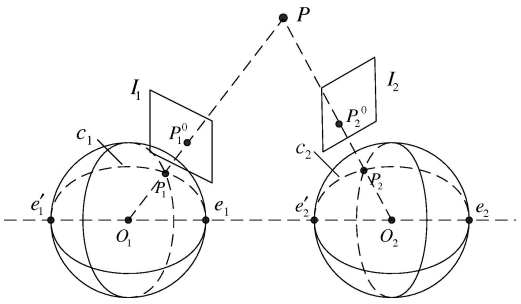



Figure 2.27: Spherical rectification [22]

### 2.3.3 Time-of-Flight

Time of flight cameras are 3D sensors that simultaneously provide intensity data and range information for every pixel. As described in table 2.4, its a principal based on the time consumed between the emission and the reception of a signal, which can be in pulsed or continuous-wave modulation [88], producing different results, mainly are used light signals, such as light-emitting diodes (LED) or laser. To estimate the depth between an object and a sensor based on discrete pulsed modulation, this technique measures the time of a near-infrared light (NIR) pulse,

via LED, based on lock-in measure phase differences between emitted and received signals [89].

Compared to other data acquisition technologies presented in this chapter, as seen in figure 2.28, where is illustrated the basis of the methods, this approach has certain advantages, such as registered dense depth and intensity images; complete image acquisition at a high frame rate; small, low weight, and compact design [89].

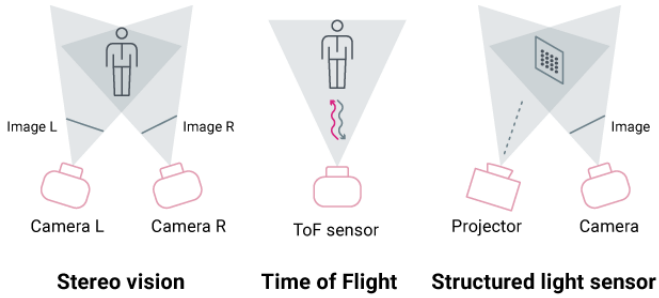



Figure 2.28: Different data acquisition technologies [23]

## 2.4 Pose estimation

In order to pick an object from the bin, it is essential to estimate a pose (position and orientation) of the object or a grasp solution. As referred this method can differentiate on account of the approach chosen, from model free or model based approaches.

Despite already being explained the major differences between model free and model based approaches in the Bin Picking Object Oriented chapter, deviates a little from this chapter. It's going to be briefly presented some algorithms and techniques to estimate a position and orientation of an object in a bin picking environment.

Pose estimation has numerous challenges. One example is dealing with a shiny object, where the challenge comes from the fact that the object appearance largely changes with its pose and illumination. Therefore, conventional 3D-2D correspondence search usually fails due to the inconsistency of feature descriptors. For textureless objects, feature matching isn't appropriate, because of the absence of texture features [90]. These adversities affect the complexity of pose estimation algorithms, making them hard to be implemented.

In contrast, there are approaches that do not implement pose estimation, such as the work written by Quanquan Shao *et al.* [24], represented in Figure 2.29. This method is simply based on self-supervised learning. Basically, it is implemented a special framework of CNN, that combines Resnet with U-net, to predict picking region without recognition and pose estimation. Essentially the robot learns to grasp cylindrical objects in a cluttered bin from the results of the previous grasp, which can be successful or failure.

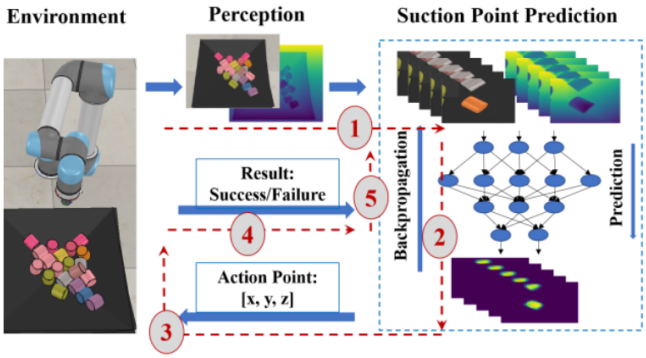



Figure 2.29: Structure of self-learning robotic picking system [24]

### 2.4.1 Model dependent

Model based methods, like the ones presented in previous chapters, require an object model. These approaches are hard to be implemented but can provide a high accuracy ratio. In order to estimate a pose of a visualized object based on a known model object its implemented algorithms that match the object with the pre-known model.

Feature based methods are commonly used in these scenarios. The objective is to find features (edges, corners, lines) and match them between images. Nevertheless, these methods are very dependent on the type of object mesh, as described in the last chapter, because different meshes can change the way a system visualizes an object [90].

Nowadays the most common technique for pose refinement is ICP, initially introduced in [91]. Basically is an algorithm employed to minimize the difference between two point clouds. In other words, aims to find the transformation between point clouds by minimizing the square errors. The inconvenience is that ICP requires a good initial pose estimation.

One of the most popular articles implementing these methods is written by Xian *et al.*, called PoseCNN [81]. Provides 6D poses of random objects, using only color images as input. PoseCNN estimates the 3D translation of an ob-

ject by localizing its center in the image and predicting its distance from the camera. The framework of the convolutional neural networks developed is divided into three different stages: feature extraction, embedding (embeds the high-dimensional feature maps into low-dimensional, taskspecific features) and classification/regression. In this case, ICP is used to refine the 6D pose.

Hinterstoisser *et al.* [65], introduced a method, called LineMOD, for detecting 3D objects using multi-modalities (2D image and depth map). This approach estimates an initial object pose and refines it by applying ICP.

### 2.4.2 Model independent

Pose estimation with model independent approaches are not entirely related to the model free methods previously referred to. These approaches can be based on different methods, such as deep learning methods, like the ones referred to in the gripper oriented chapter, feature based methods, and heuristic methods (described in the gripper oriented chapter).

The usual deep learning based methods normally require 2D images and a depth map. These information can be employed as an input for a neural network. Although several approaches only acquire and apply one type of input. For instance, Mahler *et al.* [11] presented a GQ-CNN, a model that predicts the probability of success of grasps from depth images. In such a manner, this framework estimated a grasp pose, by merely depth data, without any additional algorithm of pose estimate. In summary, GQ-CNN retrieves and explores the depth map indicated and predicts several poses of grasps, in particular, locations and angles (in the case of a parallel gripper) in order to pick up the object. The grasp with the most success rate is selected, and its calculated the depth of the object is to plan the gripper trajectory.

In some cases, can be used only RGB or grayscale images, accomplishing a plane exclusive system approach. This strategy is less common due to the diverse constraints, turning the approach into particular cases, and low accuracy estimates.

Turning away from gripper oriented approaches, Tuan-Tang le *et al.* [4] implemented the pose estimation stage with simple algorithms. The procedure is visualized in Figure 2.30, which is divided into two areas. Objects-Of-Interest (OOI) data Handling processes all the data acquired by the system, in this case, RGB images and depth images. This goes through a procedure of preprocessing the data, converting RGB image point to depth image point and selecting the best target object. The selection with the best target is collected by the second stage, Appropriate 3D Pose Estimation, where the information is queried. Finally, a 3D coordinate system is constructed on the expected target and is estimated and refined the final pose.

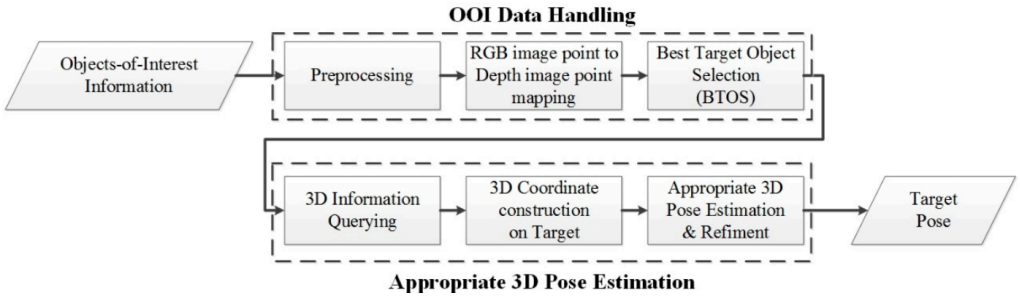



Figure 2.30: 3D pose estimation flowchart [4]

Shin *et al.* [7] also focused on simple algorithms to determine the localization and orientation of an object. After the process of segmentation, done by Mask R-CNN, its obtained the poses of objects. Firstly its calculated their centers of gravity, and secondly its applied an algorithms that selects straight lines from the center point to detect the edge points, as shown in Figure 2.31, this line is rotated by 30 degrees to find the line with the shortest distance between edge points.

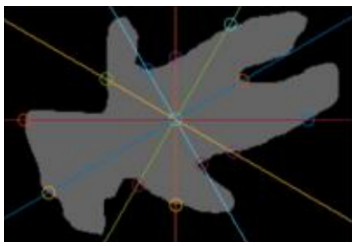


Figure 2.31: Pose estimation algorithm [7]

## 2.5 Deep learning based bin picking projects

In this section are presented several bin picking projects based on deep learning techniques, ordered by year, starting with Lenz work *et al.* in 2014 and ending with Mohammed work *et al.* in 2021. The developed frameworks are discriminated by bin picking domains, data modality and learning methodology, described in 2.2.2, 2.2.3, 2.3 respectively. In terms of neural network methods are mainly presented two different: instance and semantic segmentation.

To generate the Table 2.5 was reviewed several works about the main topic as explained, and were selected the most interesting articles relatable to the objectives of this thesis. There is no doubt that are numerous articles based on the same topic that deserves to be mentioned, although these are the ones that were exhib-

ited. The main objective of these reviews was to study the developed framework between detection, segmentation and pose estimation, and subsequently identify the methods utilized in some main areas of bin picking and deep learning. The identification system of bin picking domains that were used on each work was processed through the study of the architecture presented in each paper. At last, the variety of the applied data modalities is identified, between RGB image, depth image, point cloud data and multi-modal modalities, such as RGB-D. It is worth noting that the majority of point cloud data is produced by RGB-D modalities.

Table 2.5: Most relevant bin picking works based on deep learning.

| **Authors** | **Bin Picking Domain** | **Data modality** | **Learning** |
|---|---|---|---|
| Lenz *et al.* (2014) [68] | Gripper Oriented | RGB-D | Labelling |
| Mahler *et al.* (2017) [11] | Gripper Oriented | Depth | Labelling |
| Asif *et al.* (2018) [92] | Gripper Oriented | RGB-D | Labelling |
| Xiang *et al.* (2018) [81] | Object Oriented Model Based | RGB | Labelling |
| Lee *et al.* (2018) [13] | Object Oriented Model Based | IR Depth | Labelling |
| Shin *et al.* (2019) [7] | Object Oriented Model Free | RGB-D | Labelling |
| Le *et al.* (2019) [4] | Object Oriented Model Free | RGB-D | Labelling |
| Shao *et al.* (2019) [93] | Object Oriented Model Free | RGB-D | Self-supervised |
| Blank *et al.* (2019) [94] | Object Oriented Model Based | RGB-D | Labelling |
| Wu *et al.* (2019) [6] | Object Oriented Model Free | RGB-D | Labelling |
| Jiang *et al.* (2020) [95] | Gripper Oriented | Depth | Labelling |
| Tachikake *et al.* (2020) [96] | Gripper Oriented | RGB | Labelling |
| Sukhan Lee *et al.* (2020) [97] | Object Oriented Model Based | RGB-D | Labelling |
| Zhuang *et al.* (2021) [45] | Object Oriented Model Based | RGB | Labelling |
| Kumra *et al.* (2021) [17] | Gripper Oriented | RGB-D | Labelling |
| Iriondo *et al.* (2021) [98] | Gripper Oriented | RGB-D | Labelling |
| Jiang *et al.* (2021) [99] | Gripper Oriented | Point cloud | Labelling |
| Mohammed *et al.* (2021) [100] | Object Oriented-Model Free | RGB-D | Self-supervised |

## 2.6 Commercial products

Commercial products are frameworks developed mainly for companies, with the objective of providing a full service for, in this case, bin picking problems, supported with multiple different features. These frameworks are densely structured, composed of a high amount of metrics and algorithms, accomplishing user friendly and highly maneuverable architectures.

These products can be developed with a specific objective or for a particular company. Mainly presented in this section, are the ones defined by an objective. In most cases, these approaches are object oriented or oriented towards similar environments, because commercial products are directed towards industrial environments, where normally one robot solves only one bin picking problem with a specific environment and object.

Table 2.6: Commercial products

| **Software** | **Company** | **Features** |
|---|---|---|
| AccuPick 3D | SOLOMON | Collision-Free Motion Planning, custom GUI, Compatible Open Platform, Auto labelling, user-friendly |
| Smart Item Picker SMART | ROBOTICS | Custom GUI, modular system, user-friendly, up to 700 picking per hour |
| | FIZYR | Multiple ordered poses per object, Segmentation, 100 grasp propositions per second, Unstructured picking |
| | YASKAWA | 24/7 robot, 150 pick cycles per minute, 2 kg to 10 kg payload capacity |
| | SP Automation Robotics | 200 products per minute, 0 mm placement erros |

## 2.7 Summary

The bin picking methodologies chapter is developed to explain important bin picking and deep learning concepts and tasks divided into specific areas. This division was made according to specific articles to give a better structure, explaining the implicit problem and the solving technique.

The chapter serves as an introduction to the techniques that will be utilized in

this dissertation and describes several state of art approaches, object and gripper oriented, developed in recent years for the problem stated.

Additionally, are generated graphs and tables to illustrate important information given during the chapter. Further in-depth information about certain topics can be researched through the citations from the different articles.

# Chapter 3

# Bin Picking: Object Detection Solution

This chapter describes the proposed framework for object detection in a bin picking problem. The proposed framework was developed for an ongoing project to recognize an object in an industrial tray. The previous object recognition implemented [37] [31] was based on several features detection and description, such as surface normals, curvatures, keypoints, RANSAC features, among others, and clusters manipulation, recognizing different objects in the scene through clusters and comparing them to a reference model.

The idea of the proposed framework was to change the recognition framework and part of the segmentation approach with techniques based on deep learning methods explained in the next sections. The biggest reasons for this proposed adjustment were: deep learning models, when well trained can provide a high level of robustness to the system, meaning that, when encountering novel and difficult environments (highly cluttered environments), this system can accurately estimate results, and possibly outperform in some aspects the initial system. Another reason is the accessible modification to detect a novel object, which the framework was never trained to detect, only requiring a new neural network model with the knowledge to identify this new object, therefore not involving several complex steps.

The basis of the framework developed to detect objects in a clutter environment is depicted in Figure 3.1. As perceived in this image, there are three main areas, that will be described in the next sections of this chapter, such as 2D Segmentation in 3.6.2, 3D Segmentation in 3.6.3 and Pose Estimation in 3.6.4.

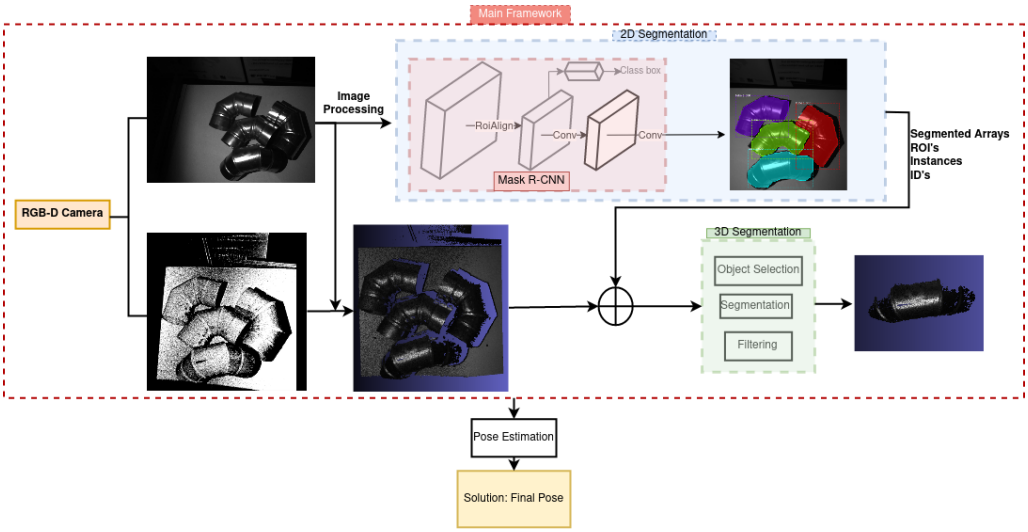



Figure 3.1: Proposed object detection framework

## 3.1 Neural Network (Mask R-CNN) framework

Deep neural networks are mainly known and selected from their architectures, presenting different features, such as bounding-box, and classifications, among many others, enabling them to reach the main objective with fewer difficulties, presenting better results. As for this work, it was intended to detect and segment instances of objects in a clutter environment. Mask R-CNN was chosen from all the possibilities considering that in the first place was the newest stable version of R-CNN, a neural network with good results and a lot of progress made in the segmentation field. Secondly, in several published papers, Mask R-CNN obtained results above average, surpassing several neural networks. Thirdly, as described in the main objective of this implementation, the objective was to find a specific known place of the model to grab the object, this way excluding most of the gripper oriented neural networks. Lastly, it is an open-source neural network with good community support, with exceptional results concerning the detection and segmentation of different objects with several occlusions.

Mask R-CNN consists of two stages. The first stage called a Region Proposal Network (RPN), proposes candidate object bounding boxes. The second stage, which is in essence Fast R-CNN [12], extracts features using RoIPool from each candidate box and performs classification, bounding-box regression, and outputs a parallel binary mask for each RoI [25].

Mask R-CNN has multiple architectures, with different convolutional backbone architectures and network heads. Evaluating different backbones, such as Resnet with 50 or 101 layers (Resnet-50 and Resnet-101), that extracted features

from the final convolutional layer of the fourth stage, which is called C4. A distinct backbone is Feature Pyramid Network (FPN), proposed by Lin *et al.* [101], that uses a top-down architecture with lateral connections to build an in-network feature pyramid from a single-scale input. It is a similar approach to Resnet, which can also extract ROI features from different levels of the pyramid, giving accuracy and efficiency gains [25]. The head architecture from these different backbones is essentially the same as the ones exhibited in 3.2 and 3.3 from Faster R-CNN (previous version), denoting the convolutions and deconvolutions with arrow, fully connected layers, and spatial resolution and channels with numbers.

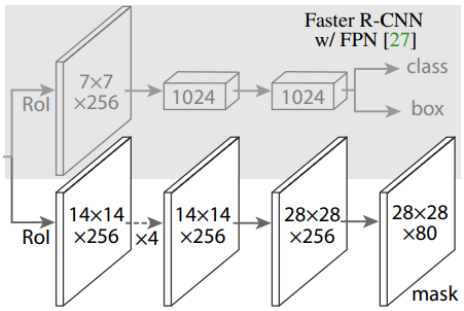



Figure 3.2: Faster R-CNN w/ FPN [25]

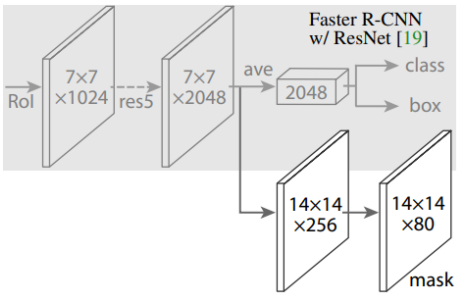



Figure 3.3: Faster R-CNN w/Resnet [25]

In this work, Mask R-CNN was implemented on Python3, Keras, and TensorFlow, with Resnet-101 and FPN as the backbone, similar to the approach proposed in [102]. The model generates bounding boxes and segmentation masks for each instance of an object in the image. The neural network backbone was selected because it has more precision than resnet-50, which in this case it is a important characteristic and the original Mask R-CNN was implemented with this backbone.

## 3.2 Deep Learning training stage methodology

As explained in the state-of-art section, neural network frameworks need to be trained beforehand, otherwise they are going to be completely lost in their objective. Several learning techniques can be used to train these neural networks, as described in 2.2.3, alternating according the each specific problem. Hence the importance of selecting the superior technique according to each problem.

Considering all the specifications given and required for this work, labelling was selected as the primary option. Given a wide range of possibilities concerning data label generation approaches, human labelling using third party labelling annotators was chosen. Human labelling can be basic because it does not require

substantial knowledge to execute the labelling task, although it is very time-consuming and can lead to several misleading data. To counter some of these problems is commonly used third party software (labelling annotators), described in section 3.4.3.

More advanced procedures can be described as semi-automated and automated methods, varying their framework according to the environment, data and objective defined. These procedures replace the high time-consuming manual labelling, producing label data in short periods and frequently with high precision labels.

In this case, the training and detection phase is completed with tensorflow. For training in this particular case was used tensorflow 2.4.2 with cuda version 11.0, allowing not only CPU but also GPU as a training device. The hardware used to train and evaluate models is detailed in Table 3.1.

Table 3.1: Experimental hardware specification

| **Hardware** | **Description** |
|---|---|
| Processor (CPU) | Ryzen 5 5600x-<br>6 cores, 12 threads |
| GPU | Gtx 1060 6 Gygabyte-<br>1408 cuda cores, 1830 Megahertz |
| RAM | 32 Gygabyte 3600 Megahertz DDR4 |

### 3.2.1 Training methods

The training phase of this project was implemented using several approaches, mainly divided into two areas: Full training from scratch; and Transfer learning. The two types of training were studied and then tested along with the implementation. Different results can emerge from these distinct methods, and they will specifically depend on the problem presented in each situation. By that means, the opinion formed in this section, about these two methods, is only related to this work implementation, and is not meant to be an overview opinion of these deep learning methods.

One of the biggest problems in deep learning is data dependency, causing deep learning approaches to depend on large amounts of training data, to satisfy neural network needs, such as data latent patterns, so it is possible to generate accurate results. The greater the complexity of the network the bigger amount needed for training data, creating almost a linear relationship between the two, meaning that in this implementation with a very complex neural network as Mask R-CNN, it is needed a high amount of data to train new network patterns to be able to detect and indentify the desired objects. Usually, pre-order layers in the

model can identify high-level features of training data, and the subsequent layers can identify the information needed to help make the final decision [27].

Aside from the training approaches, the training itself used different parameters, to diversify the neural network learning procedure, from mean grayscale values to images per GPU trained, referred to in section 3.2.3.

### Full training from scratch

Training from scratch is the standard training technique. The basis of this definition is to train all the stages divided by layers of a neural network from non-previous information given.

To apply this method is important to have a large amount of training data, that will depend on the neural network complexity, as explained earlier. This training consists of learning different features and characteristics from each stage, having the advantage of producing no false information due to not applying a pre-training stage, which can be implemented with weights or another strategy to the network official training. But on the other hand, if the information is insufficient so the neural network can learn the different features to detect an object, this training approach, no matter how much it trains, will never be enough to produce decent results.

Also, another disadvantage is that in addition to requiring large amounts of data, it also demands a large value of hours for the training itself.

With all the provided information, it is estimable that for this instance, is difficult to implement the full training from scratch approach, due to the large amounts of acquire labelled data and time required.

Contemplating the situation, researchers tried to developed solutions to reduce these values, simplifying and reducing the time needed to implement neural network applications, one of them being Transfer Learning.

### Transfer Learning

To solve the problem quoted earlier was developed a method called Transfer Learning illustrated in Figure 3.4. Transfer learning relaxes the hypothesis that the training data must be independent and identically distributed (iid) with the test data, solving the insufficient data problem required to fully train a complex neural network. Unlike training from scratch, models in the target domain are not needed to be trained from scratch, which can significantly reduce the demand for training data and training time in the target domain [27].

Tan *et al.* [27] in 2018 categorized deep learning transfer learning into 4 topics, exhibit in Table 3.2: Instance-based, Mapping-based, Network-based and

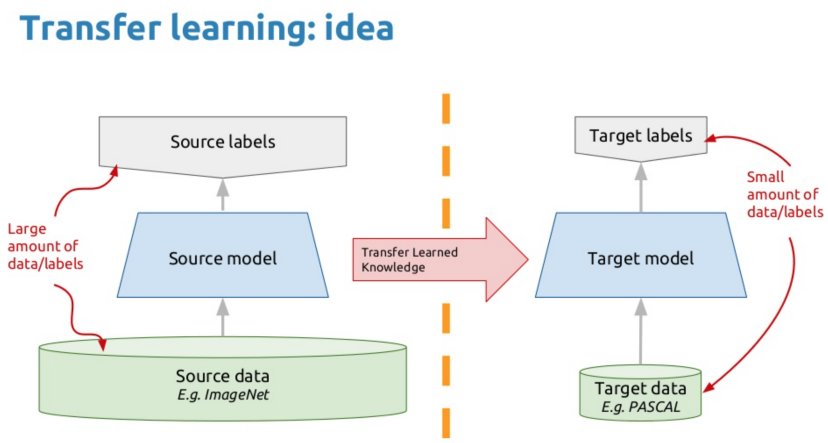



Figure 3.4: Transfer Learning [26]

Adversarial-based. Concerning this neural network and implementation, the transfer learning approach can be identified as instance-based.

The results obtained with transfer learning had much better accuracy than results generated from full training from scratch. In this implementation were only used COCO pre-trained weights, where Mask_RCNN was previously trained with the COCO segmented labelled dataset.

Table 3.2: Deep transfer learning categories [27]

| **Approach category** | **Description** |
|---|---|
| Instance-based | Utilize instances in source domain by appropriate weights |
| Mapping-based | Mapping instances from two domains into a new data space with better similarity |
| Network-based | Reuse the partial of network pre-trained in the source domain |
| Adversarial-based | Use adversarial technology to find transferable features that both suitable for two domains |

As described in Section 3.1, Mask_RCNN network architecture is divided into different layers, possible to train independently, such as heads, defined by RPN, classifier and mask heads features of the network, 3+ layers, Resnet stage 3 and up, 4+ layers, Resnet stage 4 and up, 5+ layers, Resnet stage 5 and up and all layers, where is possible to train all the layers of the neural network at the same time. Different training stages require more GPU memory to train and developed distinct results. Different trainings procedures diverged the neural network layers that were trained, given the fact that with transfer learning is not required to

train all the layers.

Working with only 6 Gigabytes of VRAM cause some constrains, for example, the stage that trained all the layers (all) from the neural network, was only possible to be applied with 640 by 480 resize resolution images, because of the memory consumption required.

#### Instance-based

Instance-based is a transfer learning strategy, based on weight adjustment. Essentially weight values are given to partial instances selected in the source domain at the training set. It is assumed that the weights selected in the source domain can be utilized in the target domain. This strategy is presented in Figure 3.5, where the source domain represents the training weights, and the target domain represents the target weights. The two circle reunion is contained by the weights taken into account, where is displayed both source and target weights [27].

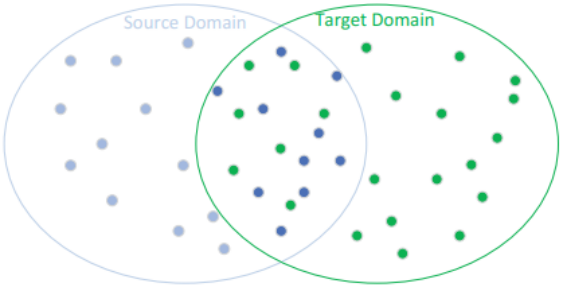



Figure 3.5: Instances-based deep transfer learning [27]

### 3.2.2 Experimental and training environment

Mask R-CNN was trained in the system depicted in Table 3.3. Unlike the original neural network from GitHub [102], this framework was processed with tensorflow version 2 (2.4.1) and ubuntu 20.04. This adjustment included a few changes to the neural network, assuring the compatibility between the Mask R-CNN framework, the libraries utilized, the CUDA toolkits and nvidia drivers.

More information about the compatibility of the different libraries and the system used are explicit in [102] and [103].

Most of the implemented Mask R-CNN training adjustments are presented in Mask R-CNN-TF2 GitHub [104]. Nearly all the adjustments were required due to the incompatibility from tensorflow functions, such as log_graph, set_intersection, sparse_tensor_to_dense, to_float, among others, that changed between tensorflow version 1.x.x and tensorflow version 2.x.x.

Table 3.3: Training and evaluation library versions

| Library | Version |
|---|---|
| Tensorflow | 2.4.1 |
| CUDA toolkit | 11.0 |
| Nvidia driver | 450.191.01 |
| cuDNN | 8.0 |
| Keras | 2.4.0 |
| Ubuntu | 20.04.4 LTS |

Furthermore, the dataset class in the main python training script was adapted to load different annotations formats (JSON and COCO format, demonstrated in the section 3.4.3) from the custom dataset generated, and different neural network parameters were adjusted, explained in section 3.2.3, and lastly, Mask R-CNN neural network layers (2+,3+, heads,+4,...) training procedure was adapted according to the learning results.

### 3.2.3 Parameter settings

Mask R-CNN neural network has several adjustable configuration variables, also called parameters, that can be can be adapted according to the neural network input, proposed objective, and training experience. These variables can control several neural network qualities, such as image resizing resolution, number of GPU, images per GPU (batch size), training steps per epochs, backbone (resnet50 or resnet101), layer size, pyramid size, anchor size, among many others.

According to the data acquired and the objective of the evaluation, some parameters need to have a specific value, otherwise, the training stage is going to fail or produce inadequate results, these parameters can be consulted and analysed in the implementation [102]. Such parameters are: steps per epoch, batch size, that require a specific value explained in equation 3.1; image resize resolution, which can not be higher than the image resolution; the number of GPU, in this case, depends on the GPU memory of the defined hardware. Further parameters are going to depend on the system and on the neural network objective, meaning that the choice made in this implementation is related to its features.

$$Steps = Dt - De/batch; \tag{3.1}$$

where:

| | |
|---|---|
| $Steps$ | Number of steps per epoch, |
| $Dt$ | Number of data to train, |
| $De$ | Number of data to validate, |
| $batch$ | Number of images per step for each GPU. |

Several training practices were applied in the implementation of the two models, to test the difference in the output and training functions results. Some of the last configuration values used for both model are illustrated in Table 3.4.

Table 3.4: Parameter values

| **Parameter** | **Value (Knee tube model)** | **Value (Triangular model)** |
|---|---|---|
| IMAGES_PER_GPU | 1 | 1 |
| NUM_CLASSES | 2 | 2 |
| STEPS_PER_EPOCH | 100 | 100 |
| VALIDATION_STEPS | 30 | 5 |
| DETECTION_MIN_CONFIDENCE | 0.90 | 0.84 |
| IMAGE_MAX_DIM | 1024 | 1024 |
| LEARNING_RATE | 0.001 | 0.001 |
| LEARNING_MOMENTUM | 0.9 | 0.9 |
| WEIGHT_DECAY | 0.0001 | 0.0001 |

### 3.2.4 Inference

As referred earlier, the deep learning training stage consists of acquiring information, for example, data patterns. The acquisition and learning can be executed with several methods, taking into account the objective required for the neural network.

After the training procedure, the model should recognize certain information, such as patterns, and create inferences from the available data. Allowing to implement a consecutive stage after training, called inference, where the trained model, with all the retained knowledge, is applied to the same or novel data, trying to figure out the different patterns, comparing them to the know models, detecting the instances according to the train realized.

Similar to a human procedure, inference requires a brain or multiple ones (CPU or GPU), to make choices about the occurring situation (data): for example when a traffic light is turning red, the correct reaction (output) can differ from the situation, between stopping or going through. This reaction is going to be different between an inexperienced user, meaning he does not know the pattern or information of the red light (model not trained), and from an experienced user, that knows the patterns and how to react to that situation (trained model), although that specific situation can be a first time for both of them.

## 3.3 Environment characterization

The bin picking implementation developed was applied to a mobile robotic platform, displayed in Figure 3.6. As observed, the robot manipulator is a UR10

Universal Robots, placed on top of a mobile platform, with an RGB-D sensor attached to the end part of the manipulator, and a ROBOTIQ parallel gripper attached to the end of the manipulator to pick both tubes and triangles, as well as several other objects.


Figure 3.6: Robot platform

In addition to the external tools, that are visible in the image, the platform also has two computers to process all the necessary data, one with CPU and integrated GPU and another with an intel CPU and a nvidia GPU. Located on the bottom are placed sick s3000 laser sensors, to garantee the robot autonomy and navigation system, considering object collisions.

The navigation system is displayed in Figure 3.7, developed by INESC TEC. As displayed in the image, the reference map is defined. The robot starts in an known position, reads the beacons placed at the warehouse, represented by the blue dots, mostly placed in the edges. After this reading, the robot estimates the 3 DoF position and starts the proposed trajectory flagged with arrows and lines.

Reaching the goal position, the movement goal is completed, and the robot switches to a bin picking action, trying to pick up an object from a bin placed at the side of the robot, exhibited in Figure 3.8, to the bin on the platform. After concluding the bin picking objective, the robot moves back to the starting point.

The bin containing the pickup objects has a darker background with grey edges. The illumination in the bin is non-uniform, frequently with low values, as demonstrated in most of the results in section 4.3.

From the environment described, the essential hardware for this bin picking implementation is the RGB-D sensor, described in section 3.4.2, the robot

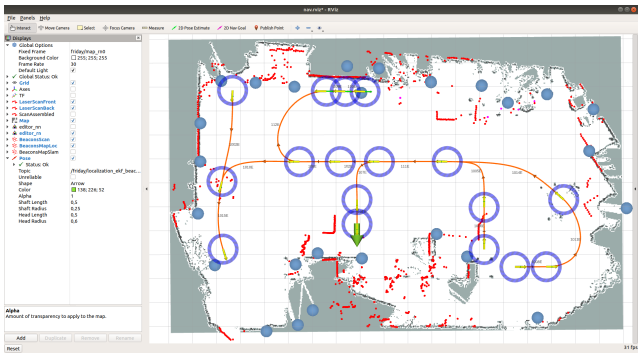


Figure 3.7: Platform trajectory depicted on rviz

Figure 3.8: Bin overview

manipulator and the computer.

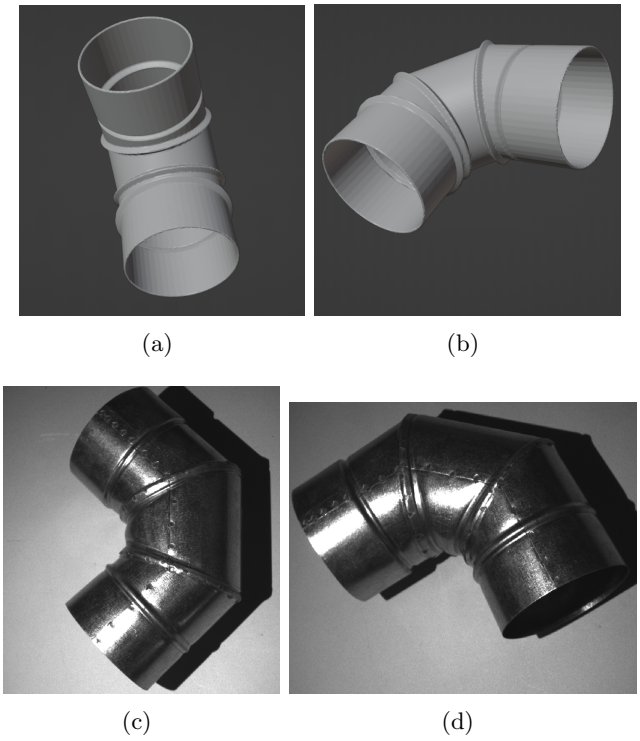
## 3.4 Dataset generation

The dataset generation in the early phase was done with real data, acquired by an RGB-D sensor as described in section 3.4.2, and labelled by a human operator, using third party annotators, which are going to represent a mask, a gripper pose ,and a bounding box, among others. This is a predominant step because it will affect the most important stage, designated by the training stage, affecting all the implementation detection.

In other words, if inaccurate labels are presented or generated, the model will get lost, considering that, the labels are untrue to the real scenario, and if on the other hand is presented with bad images, meaning the images are not appropriated to the specific training, the deep learning model will get a high lost value, therefore it will estimate imprecise results.

### 3.4.1 Models utilized

The framework was developed with the intention of being applied to any object presented in a bin. Although for initial tests the algorithm was only applied to two models, a $90^{o}$ elbow tube model, demonstrated in Figure 3.9, as Model A, and a triangle wall support model represented in Figure 3.10, as Model B.



(a) (b)

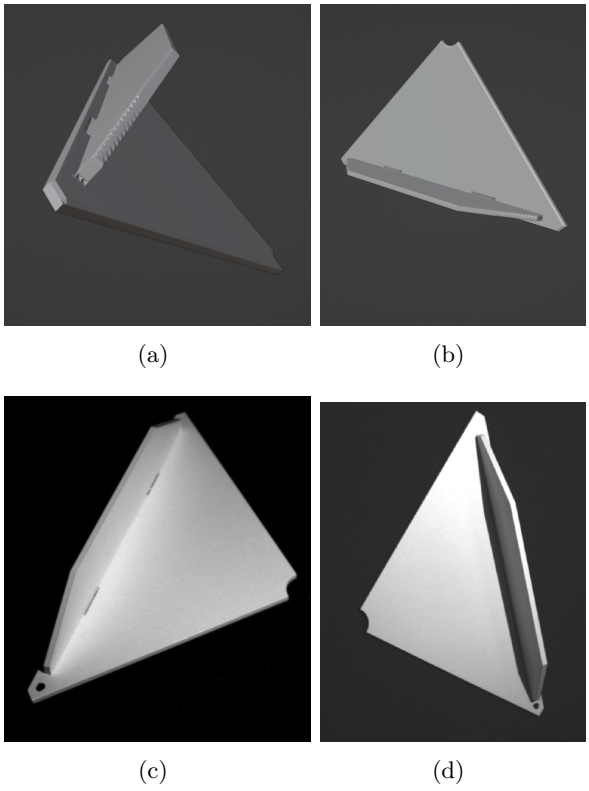
(c) (d)

Figure 3.9: $90^{o}$ elbow tube model (Model A). (Top) CAD Model, (Bottom) Real Model

The decision of these models was defined by the main grasping objective, and due to the difficult and particular presented textures. The $90^{o}$ elbow tube model exhibits a great texture to extract features, but with a highly reflective material, having inconstant patterns according to light changes. On the other hand, the triangular wall support model has a basic, dull texture, worst for acquiring features, and an absorptive material.

### 3.4.2 Data acquisition

The data acquisition is defined in Section 2.3. The data was collected by an RGB-D sensor, identified as Photoneo PhoXi 3D Scanner, capable of acquiring depth and grayscale data, allowing different types of deep learning training inputs, not exclusively traditional RGB images.



(a) (b)

(c) (d)

Figure 3.10: Triangular wall support model (Model B). (Top) CAD Model, (Bottom) Real Model

In particular, the training was executed with 2D grayscale data, although in later stages of the process was necessary point cloud data, in order to implement pose estimation and segmentation methods. The 3D point cloud data is automatically generated by the sensor through multi-modal approaches from the initial data acquisition (RGB or grayscale and depth map), illustrated in Figure 3.1.

PhoXi 3D scanner, exhibited in Figure 3.11, was utilized to acquire 3D views of the scene in both training and testing of the different algorithms. The sensor produces high-resolution data and has high accuracy for static scenes, which can be a disadvantage in certain situations where the workspace or the robot end effector is moving; that is why acquiring data with this sensor is required to be static.

Figure 3.11: PhoXi 3D Scanner [28]

This Photoneo scanner can provide RGB-D data, meaning that is possible to generate point cloud data, acquired from structured light approaches, reviewed in Section 2.3. The robustness of the scanner ensures quality scans and performance in the harshest environments, which is necessary for cluttered environments, like the ones presented in this implementation. Finally, and probably the biggest advantage is that this scanner has one of the best resolution/speed ratios, enabling a high performance and efficient framework.

The PhoXi 3D Scanner has different models, from XS to XL, diverging several features, such as the dimension of the scanner, scanning time, and scanning range. The robot manipulator in this work was equipped with a PhoXi 3D scanner model S.

The scanner, as detailed in Table 3.5, has up to 3.2 million 3D points of resolution with a throughput of 16 million points per second, and from 384 to 520 mm of scanning range, represented in Figure 3.12. The resolution of the 2D images taken is up to 2060x1544, including grayscale and depth images. It can be controlled by a software, called PhoXi Control, via ethernet, which includes a Graphical User Interface (GUI) and an Application Programming Interface (API).

Table 3.5: Photoneo PhoXi 3D Scanner S technical table

| | |
|---|---|
| **Scanning Technique** | Structured light |
| **Scanning range** | 384 – 520 mm |
| **Depth sensor resolution** | 3.2 M |
| **Point cloud type** | Grayscale |
| **ROS Driver** | *http : //wiki.ros.org/phoxi_camera* |
| **Dimensions** | 77 x 68 x 296 mm |

This sensor only generates grayscale data, implying that it can not detect object colours, but in this scenario was not necessary, this way taking advantage of high-resolution grayscale data. More in-depth features can be found at [28].

The training stage was divided into the two models presented in the previous section. Although different training data was tested to analyze the relation between the different approaches and the accuracy of the detection and training efficiency (time and memory consumed), in this first stage was used not the most efficient but the most casual training method.

The generated dataset for model A was divided into 4 batches, depicted in Figure 3.13. The first batch was introduced with different 3D views of the scenario, including only one tube or model. Similar to the first, the other batches, from second to fourth, presented a similar scenario with different views, differing in the model quantity. The second batch was generated with two models, the

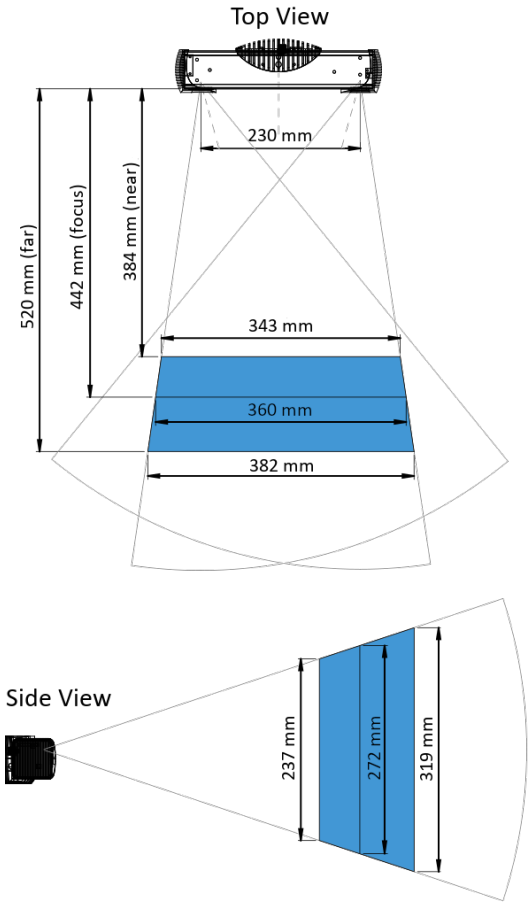
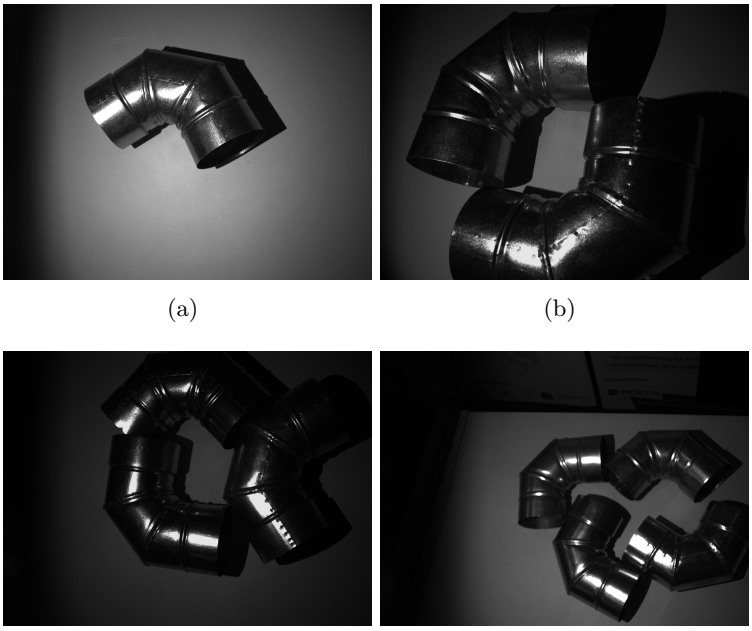



Figure 3.12: Photoneo PhoXi 3D Scanner S scanning range [28]

third with three and so on. The higher the number of batches, the more complex was the environment.

For model B, was firstly applied an algorithm based on automated labelling, described in Section 5.1, implemented with OpenCV, to generate a dataset with several 3D views of the scenario with only one object in the scene. The second resorted approach was very similar to model A: the dataset was divided into three different batches, differing in the number of objects presented. To add robustness to the model was implemented augmentation process, adding noise, brightness, and rotations, among other processes.

Table 3.6 depicts information about the dataset generation of the two models (A and B). Model A dataset is larger than model B, and with more complex objects to segment, requiring on average 21 seconds to annotate a single object,



(a) (b)

(c) (d)

Figure 3.13: Model A dataset. (a)1º Batch (b)2º Batch (c)3º Batch (d)4º Batch

taking into account that some image have 4 objects, demanding on average 84 seconds to complete an image annotation with four objects. Model B dataset, originally is much smaller, including 59 images with 155 annotations. However are applied more augmentation steps, generating 179 images from the original 59.

Both datasets include an augmentation step while training. This process can diverge from rotating, cropping, flipping, saturating, and adding noise or blue, among many other processes to enhance the training, increasing the quantity and the robustness of the training data.

Revising all the data described in this section is possible to conclude that human labelling is a tedious process, requiring a lot of time and precision between each annotation. Combining the results, the labelling of both datasets was completed in 17257.95 seconds, equivalent to 4 hours 47 minutes and 37 seconds, excluding the label identification, combining only the segmentation.

Table 3.6: Dataset information

| **Dataset** | **Nr of images** | **Nr of annotations** | **Time per annotation (s)** |
|---|---|---|---|
| Model A | 488 | 737 | 21 |
| Model B | 59 | 155 | 11.49 |
| Model A + Augmentation | 732 | - | - |
| Model B + Augmentation | 238 | - | - |

### 3.4.3 Dataset annotator tool

As described in the section 2.2.3, labelling is a learning methodology based on successful or unsuccessful labels collected by humans or physical trials. There are diverse methods of generating these labels for the acquired training data, in this case, 2D data, because there were only used grayscale images. Between the diverse methods, in this first approach, the labels were generated by human hand with the help of a labelling software.

Dataset annotators are third party software, mainly designed to simplify the human labour for generating labels and to decrease the time necessary to complete all the labels. In Table 3.7 are exhibited some of the open-source image annotators that can be used for deep learning.

Table 3.7: Labelling annotators tools (open-source)

| **Labelling annotators** | **Annotation format supported** | **Features** |
|---|---|---|
| VGG image annotator (VIA) | 2D, Video, Audio | Draw bounding box, polygon, circle, elipse, point, polyline, segmentation. |
| COCO Annotator | 2D | Segmentation, draw bounding box, key points, api endpoints, datasets already annotated, simultaneously labelling, advanced tools. |
| Rectlabel | 2D | Draw bounding box, polygon, cubic bezier, line, and point, keypoints, brush, superpixel tools, segmentation, Core ML models. |
| LabelMe | 2D Video | Polygon, draw bounding box, circle, line, point, flag for classification and clearing, customized GUI. |
| LabeIimg | 2D | Draw bounding box. |
| CVAT | 2D, Video | Draw bounding box, object's grouping, object's mapping, polygons, polylines, points. |
| VoTT | 2D, Video | Cloud storage providers, draw bounding box. |
| ImgLab | 2D | Auto suggestion, draw bounding box, circle, polygon, point. |
| Fiji | 2D | Draw bounding box, circle, polygon, polyline, point. |

The annotators are distinguished by their interface and specific characteristics such as: different outputs, in this case, it can be diverse formats, such as JSON, and COCO, among others; different region shapes, in order to create the label shapes, which can be a rectangle, a polygon with unlimited vertices, key-points,

circles, ellipses, among others. Some of these annotators can be used not only for images but for video and audio.

The annotator used in the first stage of this work was the VGG annotator and afterwards was the COCO-annotator. The two work similarly, VGG image annotator (VIA) is a simple manual annotation software for images, audio and video, is open source and can be run via HTML [76], so it is not necessary to install anything and it is user friendly. The segmentation label format for the JSON file is explicit in Table 3.8 and for the COCO file in Table 3.9.

Table 3.8: JSON segmentation format

| ID | Categories | Sub-categories | Sub-sub-categories | Value(examples) |
|---|---|---|---|---|
| id | filename | | | file_name.png |
| | size | | | (w*h) |
| | regions | shape_attributes | name | polygon |
| | | | all_points_x | [771,776,773,...] |
| | | | all_points_y | [959,851,832,...] |
| | | region_attributes | label | label1 |
| | file_attributes | | | |

COCO annotator is a web-based image annotation tool designed to create training data for image localization and object detection. It provides many features including image segmentation, tracks object instances, labelling objects and the annotations are exported in COCO format [105]. It was used in docker mode, following the Getting Started instructions in [105].

Labelling generation is an essential process because it can interfere with all the results obtained in basically every stage of the detection. An example is demonstrated in Figure 3.14, presenting two labels of the same image, both generated in the same software. The difference between them is the number of instances presented: the left figure with six instances and the right with four instances.

The distinct information given to the network can create different outputs in the 2D Segmentation algorithm, causing misleading information from this point on.

The labelling process, at the first look, may not seem very important but affects all the work in a deep learning framework; that is why human labelling demands high accurate labels, and semi-automated and automated labelling software's are so difficult to develop.

### 3.4.4 3D Point cloud data structure

This section has the intention of clarifying the point cloud data structure that was obtained by the RGB-D sensor referred to in section 3.4.2.

Table 3.9: COCO segmentation format

| Categories | Sub-categories | Value(examples) |
|---|---|---|
| **images** | id | 1 |
| | dataset_id | 1 |
| | category_ids | [2,4,...] |
| | path | "..." |
| | width | w |
| | height | h |
| | file_name | "file_name.(png,jpg)" |
| | annotated | (false,true) |
| | annotating | [...] |
| | num_annotations | (0,1,...) |
| | metadata | ... |
| | deleted | (true,false) |
| | milliseconds | 2 |
| | events | [] |
| **regenerate_thumbnail** | id | 1378 |
| | image_id | 1 |
| | category_id | 2 |
| | segmentation | [[453.0,475.0,...]] |
| | area | 417916 |
| | bbox | [453.0,247.0,...] |
| | iscrowd | (true,false) |
| | isbbox | (true,false) |
| | color | #35aaef |

Figure 3.14: Labelling difference

One of the biggest advantages was that the point cloud was ordered (Figure 3.15), meaning that it can be manipulated as an image, with three dimensions $x$,$y$,$z$, where $x$ and $y$ are respectively represented by the width and the height of the captured image. In this case, the width had 2064 pixels and height had 1544 pixels, and the last dimension ($z$), is represented through the depth map.

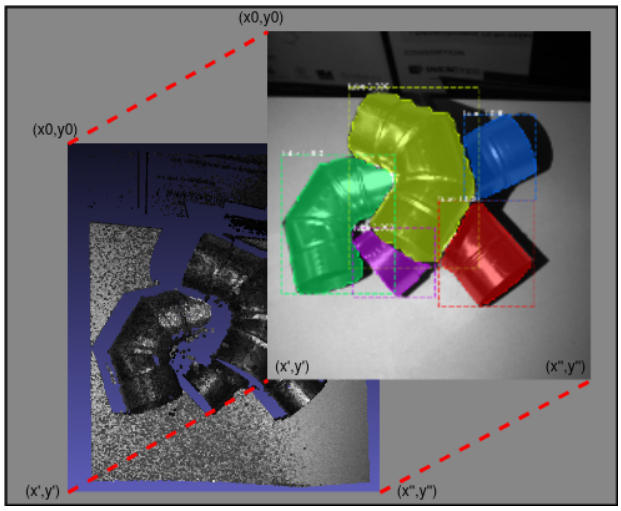



Figure 3.15: Ordered point cloud

In such a manner, these features enable the 3D Point Cloud data segmentation derived from 2D segmentation. In addition, these data structures had the regular properties, such as normals, curvature, and RGB channels, among others.

## 3.5 Filtering

Filtering and down sampling was implemented at this stage to reduce the density and the level of detail of point cloud segmentation, allowing to increase the overall algorithm performance and accuracy. As described earlier, this method was developed with PCL filters and segmentation techniques.

In this Section are briefly described several PCL filters, including the ones implemented, as well as tests that were performed with each filter implementation with and without segmentation. These tests include: the number of points reduced from both models, depicted in Table 3.10; and several point cloud results, demonstrated in subsequent figures.

PCL filters are designed for 3D point cloud data applications, such as data acquired by a depth sensor, like in this case. This sensor can produce measurement errors, which can influence the accuracy of the final results. Filters have different mechanisms, such as noise removal, down sample and outliers detection.

### 3.5.1 Pass through

Pass Through filter, as the name suggests, iterates the entire point cloud, and selects or removes the data inside or outside a specific limit (min and max), diverging from range, colour or intensity [106]. The range can be defined through

Table 3.10: PCL filters point cloud reduction

| **Data retrieved** | **PCL Filters** | **Model A (Quantity of Points)** | **Model B (Quantity of Points)** |
|---|---|---|---|
| Without Segmentation | Initial Point CLoud | 3,186,816 | 2,113,650 |
| | Passthrough Filter | 1,051,514 | 2,113,650 |
| | VoxelGrid Filter | 292,750 | 170,344 |
| | StatOutRemov Filter | 284,913 | 148,284 |
| With Segmentation | Initial Point CLoud | 3,186,816 | 2,113,650 |
| | Passthrough Filter | 147,214 | 213,608 |
| | VoxelGrid Filter | 34,434 | 11,529 |
| | StatOutRemov Filter | 33,630 | 9,620 |

the different axis $x,y,z$, creating a band on one axis or a 2D or 3D bounding box, represented in Figure 3.16.

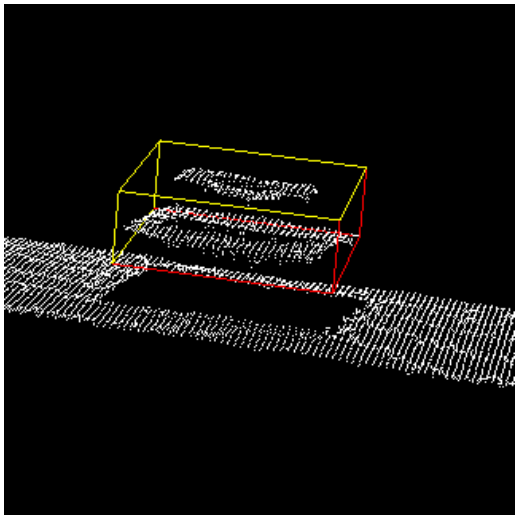
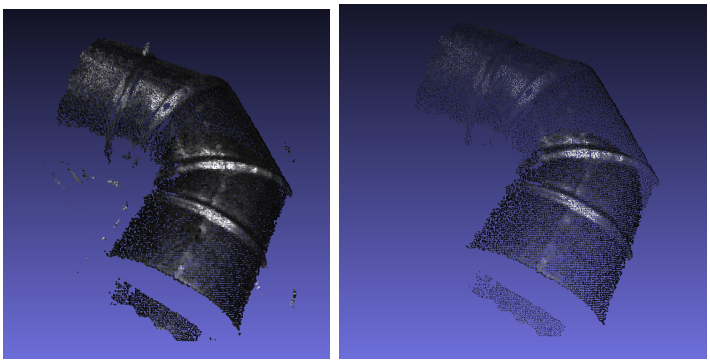


Figure 3.16: Pass Through filter example [29]

### 3.5.2 Voxel grid

Voxel grid is a down sampling and filtering technique, mainly used to scale down the level of detail of a point cloud by reducing the point quantity, replacing voxel clutters with a specific leaf size with a single point (example in Figure 3.17) [106].

There are different processes to implement this method, namely approximate, centroid and occlusion estimation. Approximate voxel grid filters return the centroid of the voxel itself, instead of the centroid of all the points in the voxel, minimizing the time consumption. Centroid voxel grid filters compute an approximation of the centroid of all the points to each voxel cluster, representing the underlying surface more accurately. Occlusion estimation filters, as the name suggests, estimates occluded space in the scene [107].



(a) (b)

Figure 3.17: Voxel grid + Statistical Outliers Removal filter example. (a) Segmented point cloud (b) Voxel grid filtering

### 3.5.3 Outlier removal

Outlier removal is a technique that removes undesired points (outlier data) caused by corrupted measurements, shadows, light and specific textures. This can be applied using different types of filters.

Statistical Outlier Removal filter uses point neighbourhood statistics to filter data, computing the average distance that each point has to its nearest neighbours. The distance threshold is the sum of the mean with the standard deviation times the multiplier. The number of nearest neighbours and the multiplier can be adapted (example in Figures 3.18 and 3.18).

Radius Outlier Removal filter selects points in a point cloud based on the neighbours set by a certain radius. If the point has fewer neighbours in the selected region than the chosen value, the point will be considered an outlier [106].

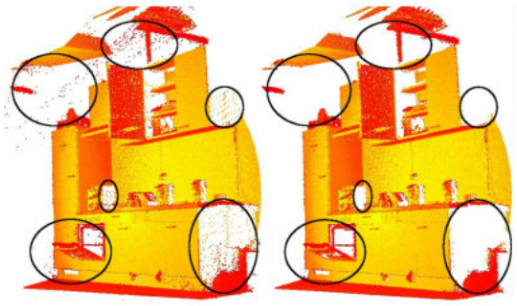


Figure 3.18: Statistical Outlier Removal [30]

### 3.5.4 Median

Median filter implements a depth filter technique ($z$ axis), reducing the noise of 3D organized point cloud data, with the advantage of being efficient and robust to outliers points. This filter applies a moving window of fixed size, replacing the pixel in the center with the median inside the window. The maximum distance allowed to move during filtering is set to a specific value that one point can move along the $z$ axis [106].

### 3.5.5 Filter evaluation

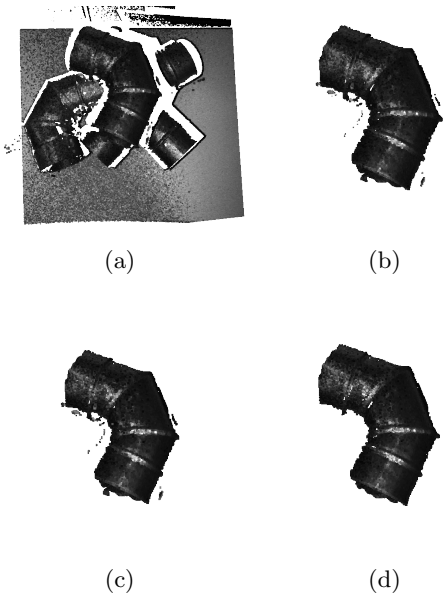
The evaluation was developed for the two models: $90^{o}$ elbow tube (Model A) and triangular wall support (Model B). The results of the first test are presented on Table (3.10), which demonstrates the reduction of points retrieved from different point clouds, generated thought specific PCL filters, such as pass-through filter, Voxel Grid based on centroids and Statistical Outlier Removal, that were applied in the main object detection framework.

As observed in Figure 3.19, these filters can reduce a significant amount of points from the original point cloud, most of them identified as noised or not relevant clusters.

The filters applied in this stage take into account the relation between the point cloud density and the displayed information, considering that the final objective is to match the partial point cloud with a pre-known model. Considering this fact, filters can be adjusted to generate less dense or denser outputs, according to the matching phase demand.

## 3.6 Object detection and segmentation framework

The problem presented at this stage is an industry bin picking problem, characterized by a number of equal objects placed simultaneously in a bin, creating a



(a) (b)

(c) (d)

Figure 3.19: Model A Filtering

high cluttered environment.

Relating this problem, the developed framework main objective was to detect identical types of objects, some of them with slight variations in certain characteristics, such as size, particular edges and features, that were unknown in the early stages (2D detection stage). The detection, specified as instance segmentation or 2D segmentation, has the intention to detect singular objects in a cluttered environment and to disclose their 2D shape, in most cases detecting segmentations with polygon shapes due to the several occlusions presented in the environment.

In between the first and second stage (3D segmentation stage), all the masks are agglomerated to a single image; since detection is performed in 2D, mask overlapping will not be possible. Each mask is identified with different values, starting by ID=1 and ending with ID= total number of objects detected. Furthermore, the center of ROI, image size, and number of instances (objects) are added to the end of the image array.

In the second stage, detailed in section 3.6.3, is loaded the point cloud taken at the same time of the image. It is estimated the height of each instance received from the image, and the object closer to the sensor is selected. In the end, the partial point cloud acquired from the selected object mask is sent as PointCloud2 message. An overview of the entire framework is displayed in Figure 3.20.

For a straightforward explanation, Algorithm 1 presented the framework basis as a pseudo-code.

**Algorithm 1** Object detection and segmentation

**Require:** $im = image, pt = pointcloud, m = trained\_model$
  **if** $Goal$ = load_triangle **then**
    Load Mask R-CNN triangular trained model
  **else if** $Goal$ = load_tube **then**
    Load Mask R-CNN knee tube trained model
  **else if** $Goal$ = process **then**
    **if** Goal.loaded **then**
      $mask\_im \leftarrow Object\_Detection(im, m)$
    **else if** Goal.notloaded **then**
      Load Mask R-CNN knee tube trained model //standard
      $mask\_im \leftarrow Object\_Detection(im, m)$
    **end if**
  **end if**
  $filtered\_pt \leftarrow PointCloud\_Segmentation(mask\_im)$
  **if** $filtered\_pt \leq 1000$ data points **then**
    Result aborted
  **else if** $filtered\_pt \geq 1000$ data points **then**
    Result successful
    $Result = filtered\_pt$
  **end if**

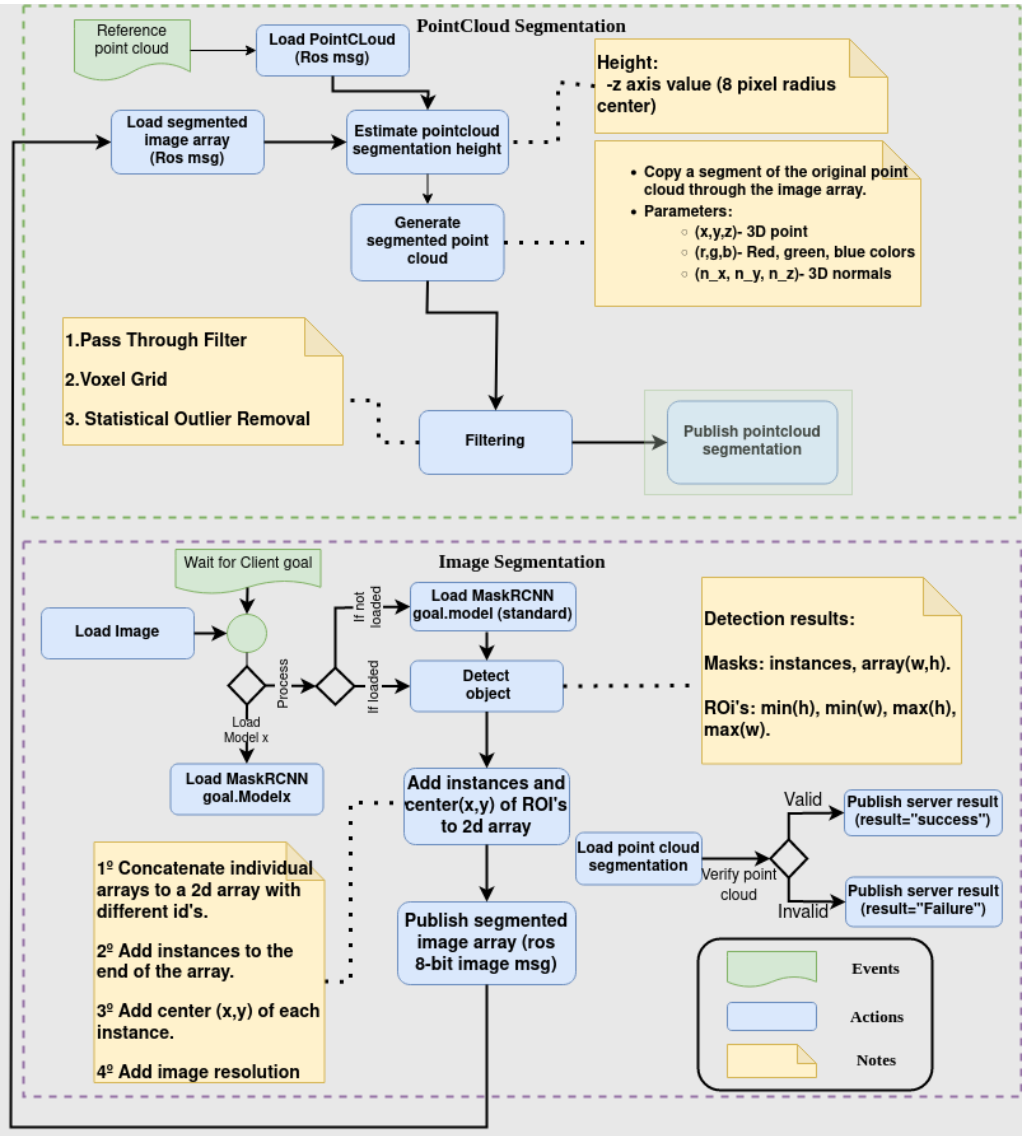



Figure 3.20: Object detection overview

### 3.6.1 ROS action server interface

The object detection server (ROS action) exchanges information between the ROS client and the ROS server. The server receives a goal, that can be defined as load_model, diverging from the type of model requested to detect (example: triangle, tube), and the type of procedure (example: process, load).

If the goal is load_model, the server is going to load the explicit model, received as goal.model, in other words, this is a pre-stage before the execution of the detection system, that can be called while the robot is setting up to pick the objects in the bin, to speed up the loading process that, in average, can take 4 seconds. Mask R-CNN models are h5 files (HDF) with an average of 247 Kilobyte size, delaying the entire procedure because of the high time-consuming loading system, therefore implementing this method of loading while the robot manipulator is setting up.

If the goal is process, it verifies the model status. If the model is already loaded (process described in the last paragraph), the server is going to proceed to the 2D detection stage, otherwise, the server will run the loading procedure, to load the specific model subsequently before executing the 2D detection algorithm.

The server can also have a feedback parameter, that receives the progress, in percentage, of the algorithm.

To conclude, the Object Detection Server presents the result as successful or unsuccessful, based on the analysis of the published point cloud message. The sensor_msgs/PointCloud2 point cloud message includes the highest segmentation region derived from the partial mask, with all the necessary data.

### 3.6.2 2D object detection

The detection is generated in a 2D space, as defined in Section 3.6, resorting to an object instance segmentation framework (Mask R-CNN), which can identify several segments, with a mask and bounding box, of the same or different type of objects.

In this implementation, the objective was to identify and segment the same type of objects (in each detection) exposed in a bin, with the higher accuracy and precision achievable.

The detection algorithm starts by loading the specified model, as explained in section 3.6.1. After the loading is completed and is called the process command from the client, the algorithm waits for an image message publication, consequently reading the image data (lines 3 and 4 from Algorithm 2).

After reading the input image, image processing techniques are performed to enhance the object's visualization. Firstly is implemented a denoise method, with

an OpenCV function named fastNlMeansDenoising (explained in D.4), consecutively is applied a Contrast Limited Adaptive Histogram Equalization (CLAHE) (explained in D.4). The functions were applied in this order to first remove noise from the image, because CLAHE causes noise in near-constant regions as the background, and to improve the contrast in the image, improving several detections with darker images. In terms of the functions parameters, for fastNlMeansDenoising was applied the standard parameters except filter strength that was selected according to tests executed in several images, and for CLAHE was applied a contrast limiter related to the mean of the image and a grid size 3 by 3 so the affected tiles are smaller.

Mask R-CNN detection, explicit in Algorithm 2 line 9, extracts the number of objects, mask (segmentation) of each object and bounding boxes. Afterwards is generated a 2D array with all the masks discriminated by a different id for each instance, the number of objects, the center $x,y$ of each bounding box, and the image resolution, and is published as an 8-bit image topic to be received by the 3D point cloud segmentation algorithm.

In the end is published a result to the client: successful if is collected an acceptable point cloud from the 3d point cloud algorithm, and unsuccessful if the point cloud is not acceptable or is not received.

**Algorithm 2** $mask\_im \leftarrow Object\_Detection(im, m)$

**Input:** $im = image, m = trained\_model$

1: $Device$ = "cpu"
2: $Config$ = InferenceConfig
3: Subscribe to im topic
4: wait for im msg
5: $im = fastNlMeansDenoising(im)$
6: $limit = exponential\_function(image\_meanvalue)$
7: $CLAHE = createCLAHE(cliplimit = limit, tileGridSize = (3, 3))$
8: $im = CLAHE.apply(im)$
9: $detection = m.detect(im)[0]$
10: $masks \leftarrow detection['masks']$
11: $rois \leftarrow detection['rois']$
12: $Instances \leftarrow masks.shape[-1]$
13: $info\_array = instances, im.size$
14: **for** i in range(0, instances) **do**
15: $roi\_x = (rois[i, 1] + rois[i, 3])/2$
16: $roi\_y = (rois[i, 0] + rois[i, 2])/2$
17: $info\_array = info\_array, roi\_x, roi\_y$
18: $binary\_mask = masks[:, :, i].astype(np.uint8) * (i + 1)$
19: $id\_mask = (binary\_mask == (i + 1))$
20: $final\_mask[id\_mask] = binary\_mask[id\_mask]$
21: **end for**
22: $mask\_im = append(final\_mask, info\_array)$
23: Publish $mask\_im$

### 3.6.3 3D point cloud segmentation

The 3d point cloud segmentation algorithm starts at the same time as the 2D object detection algorithm. It expects to receive an image array containing all the detection information provided from Object_Detection, and the original point cloud acquired at the same time as the grayscale image. If it does not receive the image array, the point cloud is read and the segmentation algorithm is not executed. On the other hand, if the point cloud is not received, the segmentation algorithm enters a condition (lines 4-7 Algorithm 3) to wait for the point cloud message publication.

When the point cloud is received (lines 1-2 Algorithm 3), the point cloud message is transformed from a ROS point cloud message (sensor_msgs::PointCloud2) to a PCL pointcloud (pcl::PointCloud< T >).

As described in the previous section, the image array information containing the detected masks with different ID, the center(x,y) of ROI of each mask, the image resolution, and the number of instances are organized in specific variables

as demonstrated in Lines 9-14 from Algorithm 3, so it is easier to manipulate the acquired data.

Succeeding the data management, the original point cloud is resized, allowing to treat this point cloud as a 2D array with the same shape as the original grayscale image acquired, and is measured the mean 8x8 center region z value of each segment (instance), demonstrated in lines 16-26 (Algorithm 3), to predict the object closest to the sensor in line 27(Algorithm 3).

After estimating the highest point cloud segment, the segment is extracted from the original point cloud throughout the ID (line 28 to 34 of Algorithm 3) and are applied several filters (line 35 of Algorithm 3), described in Section 3.5, that essentially were used to remove noise and information not necessary.

This algorithm structure improves the total time consumed because, instead of selecting and filtering every point cloud segment detected, it only selects and filters the highest segment points, which is the essential instance to pick up.

To conclude the inverse transformation is implemented (line 36 of Algorithm 3), to publish the point cloud segment as a pointcloud2 message.

**Algorithm 3** *PointCloud_Segmentation(mask_im)*

**Input:** $pt = pointcloud$

Subscriber to pt_msg
$pcl :: fromROSMsg(pt\_msg, cloud\_received)$
Subscriber to mask_im
**while** $!pt\_received\_flag$ **do**
    $ros :: Duration(0.01).sleep();$
    $ros :: spinOnce();$
**end while**
Shutdown pt_msg subscriber
$width = im\_width$
$height = im\_height$
**for** int i=0; i<Instances; i++ **do**
    $cloud\_center\_x[i] = info\_array.roi\_x$
    $cloud\_center\_y[i] = info\_array.roi\_y$
**end for**
Resize $cloud\_received(height * width)$
**for** $l = 0; l < Instance, l++$ **do**
    **for** $i = cloud\_center\_x[l] - 8; i \leq cloud\_center\_x[l] + 8, l++$ **do**
        **for** $j = cloud\_center\_y[l] - 8; j \leq cloud\_center\_y[l] + 8, l++$ **do**
            **if** $cloud\_received \rightarrow at(i, j).z > 0$ **then**
                cnt[l]++
                $h[l] = h[l] + cloud\_received \rightarrow at(i, j).z$
            **end if**
        **end for**
    **end for**
    h[i]= h[i] / cnt[i]
**end for**
$higher\_cloud = std :: distance(h, std :: min\_element(h, h + Instances))$
**for** int i=0; i<width; i++ **do**
    **for** int j=0; j<height; j++ **do**
        **if** id_cloud(i,j)==higher_cloud+1 **then**
            $partial\_cloud = push\_back(cloud\_received- > at(i, j))$
        **end if**
    **end for**
**end for**
$filtered\_pt = Filters(partial\_cloud)$
$pcl :: toROSMsg(*filtered\_pt, pointcloud\_result)$
Publish *pointcloud_result*

### 3.6.4 Pose estimation and grasping

The pose estimation algorithm is implemented after the point cloud segmentation, and is based on cloud matching, between the reference point cloud (displayed in Figure 3.21) and the acquired point cloud [37].

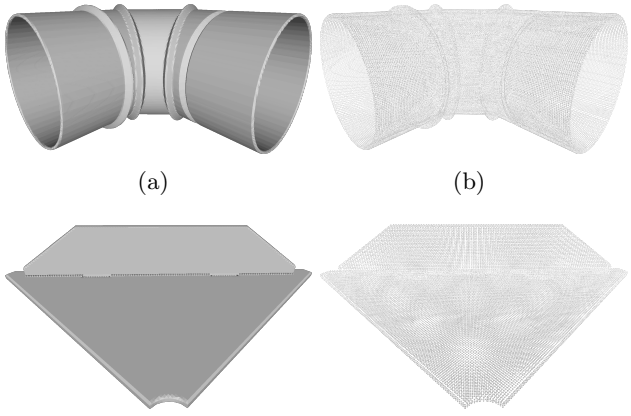



Figure 3.21: Reference point cloud. (a,c) CAD model (b,d) Generated point clouds

Essentially, the segmented point cloud is matched with a reference point cloud provided through the CAD model of the specific part that is being picked; subsequently is generated a grasp solution from the grasp evaluation system, detecting the best possible grasp (from all the possibilities) to pick the object in the current position.

The algorithm begins loading the pipeline configurations and the reference point cloud, acquired either from the $90^{o}$ elbow tube CAD model or the triangular wall support CAD model, displayed in Figure 3.21. Subsequently, the point cloud is filtered and are estimated the surface normals and curvatures. Concluding the first step, the reference point cloud is stored, and is applied and stored keypoint detections and descriptions.

The initial pose is estimated through RANSAC, and an initial alignment of the two point clouds is refined, applying ICP functions, as detailed in Figure 3.22, applying a transformation from the reference point cloud, displayed in the bottom left image of Figure 3.22 to the detected point cloud in the top right image of Figure 3.22.

Concluding the pose estimation algorithm, the final refinement is implemented through another ICP, and the 6 DoF pose is estimated.

Posterior to the pose estimation, object collision is detected to evaluate the

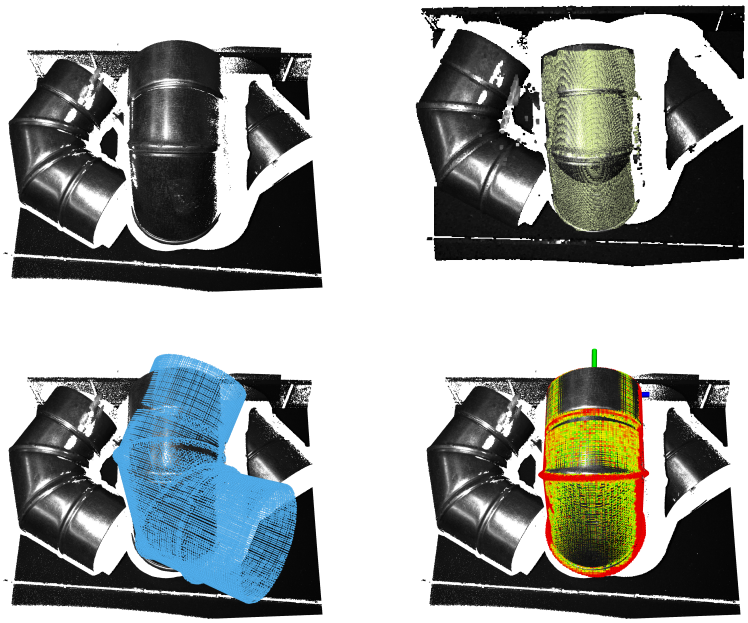


Figure 3.22: Pose estimation overview

best grasp candidate among all the previously taught grasp poses exhibited in Figure 3.23. This decision is formed through scores that are given based on heuristics and on the type of gripper (parallel gripper). In the end, the grasp with the lower cost is chosen [31].

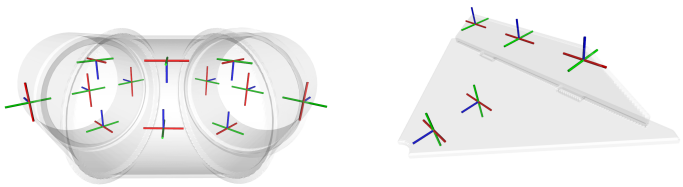


Figure 3.23: Grasping candidates [31]

After the full procedure (detection, segmentation, pose estimation and grasp estimation) the robot manipulator starts the bin picking action, displayed in Figure 3.24. As observed, after moving to the scanning point depicted in (a), the manipulator captures the input data and starts the bin picking framework. After estimating the grabbing pose, it moves to the target, grabbing the object by the reference candidate, explained in Figure 3.23, and observed in (c). Subsequently,

it changes the position to the object placement pose, to place the object in the bin, shown in (d).

The same process is repeated for any other object, for example the triangular wall support model, displayed in the bottom images of Figure 3.23, modifying only some features of the main object detection framework.

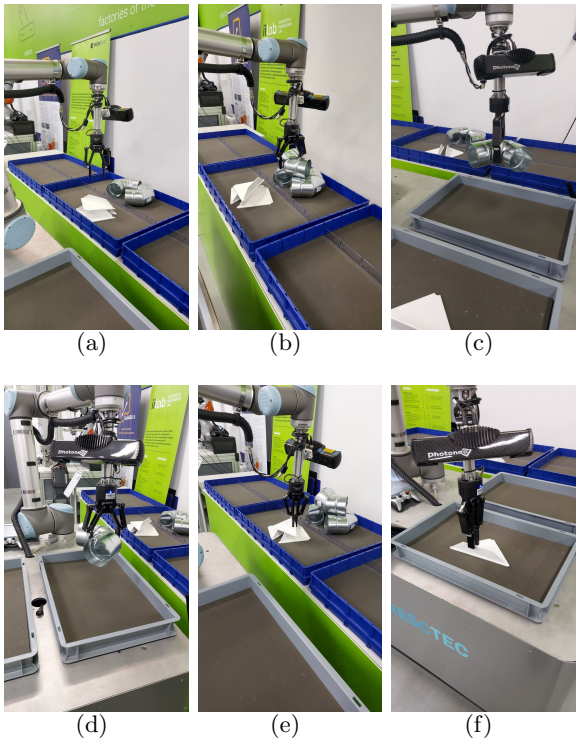



Figure 3.24: Bin picking action sequence

## 3.7 Summary

This chapter details the full implementation development. In each section are described the different stages of the implementation, from training the neural network, generating and annotating two datasets (Model A and B), point cloud filtering algorithm, 2D object detection and 3D point cloud segmentation system, and to conclude the pose estimation and grasping system.

The main algorithms, in particular detection and segmentation, are explained through flowcharts and pseudo-code, detailing all the features and functions utilized in each algorithm. The full implementation explanation is supported by images and tables across all the sections, providing a better learning experience.

Concluding the chapter, are demonstrated the 6 DoF pose estimation and grasp evaluation through the developed detection and segmentation processes, as well exhibited an example of the full implementation on a real robot.

# Chapter 4

# System tests and evaluation

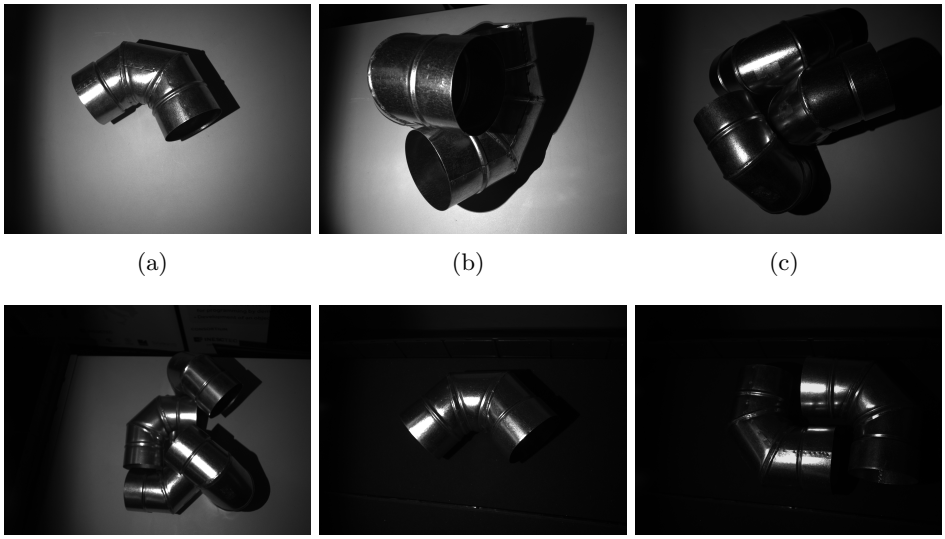
The object detection and segmentation framework was tested in eleven types of scenarios (datasets), divided into six types of datasets for model A and five types of datasets for Model B. The computer utilized for this evaluation is detailed in Section 3.1, however, the GPU was not required for any tests, including the time consumed by the algorithms evaluation test.

The evaluation datasets, exhibited in Figures 4.1 and 4.2, are divided according to the number of objects presented in the bin, both starting with one object; and the type of environment, varying between a known environment, where all the images are detailed in the training dataset and a novel environment, with less illumination, new positions, viewpoints and backgrounds, creating an ardours scenario to apply and test the developed framework.

The results that will be demonstrated in this chapter are based on the evaluation metrics detailed in Section E. The values displayed in Section 4.2 are an average based on all the measurements acquired from the eleven dataset batches, detailing more information in Table 4.1. The evaluation metric values for each individual sample, and several results for each batch, are displayed in Appendix A and Appendix B, detailing all the information acquired from every sample of the dataset, represented by metric values as well as images, providing the necessary data to conclude the precision and accuracy of the system.

Table 4.1: Evaluation dataset

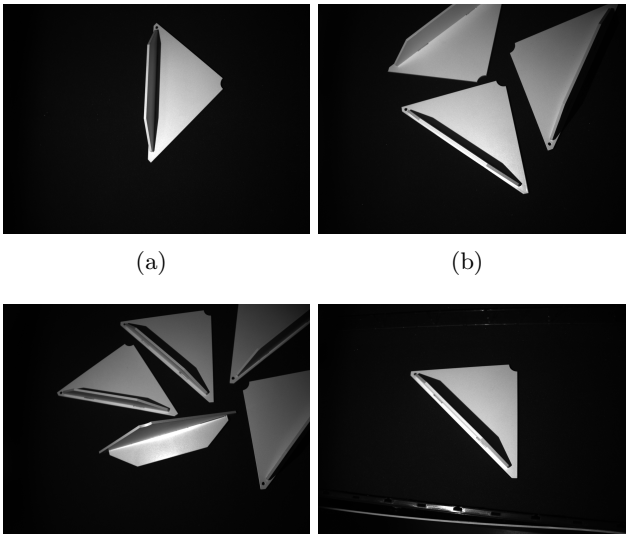
| **Evaluation dataset** | **Number of images** | **Number of annotations** |
|---|---|---|
| tubes_eval | 118 | 269 |
| triangles_eval | 54 | 102 |



(a) (b) (c)

(d) (e) (f)

Figure 4.1: Evaluation Datasets Model A (knee tube)



(a) (b)

(c) (d)

Figure 4.2: Evaluation Datasets Model B (triangle)

## 4.1 Training evaluation

The Mask-RCNN training can be evaluated with the tensorboard library through specific logs created when the training is started. These logs have important information, such as epoch loss, bounding box loss, class loss and mask loss.

Epoch loss defines the average neural network loss acquired in each epoch, in

other words, it represents the loss when updating the internal model parameters, to verify if the neural network is actually learning from each epoch.

Bounding box loss details the loss of the bounding box detection, represented by the rectangular region that detects each object. This value is calculated from how distant the detected bounding box is from the ground data.

Class loss in these two training stages has a lower impact in analytic terms, due to the few classes presented in each dataset. Each dataset is defined with only two classes, one for the objects (ex: triangle, tube) and another for the background.

Mask loss is probably the more important information of the training stage, along with the epoch loss, representing the loss between the detected mask of each object and the ground truth, allowing to estimate the average object segmentation results based on the value exhibited.

Unlike many other neural networks, the information displayed will vary according to particular Mask-RCNN layers. As seen in Figure 4.3, but it is not observed a constant loss decrease. This occurs due to the change in Mask-RCNN layers, as displayed in Figure 4.3 (a), from 1.4k steps to 1.5k (k meaning a thousand), noticing an epoch loss increase.

The larger graph peak values displayed in every loss graph, for example in epoch 2.3k in Figure 4.3 (b), are generated due to the training breaks, restarting again from the same point. These breaks were needed because the training consumes a lot of time and processing power, incapacitating the computer from executing any other task.

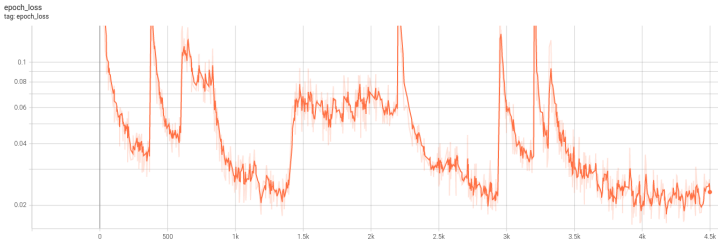
### Model A training evaluation

Analysing the information displayed in Figure 4.3 it is observed that all the graphs start to stagnate (not making progress) at the end, for almost 600 steps, oscillating between the same range of loss values, giving no reason to progress with the training.

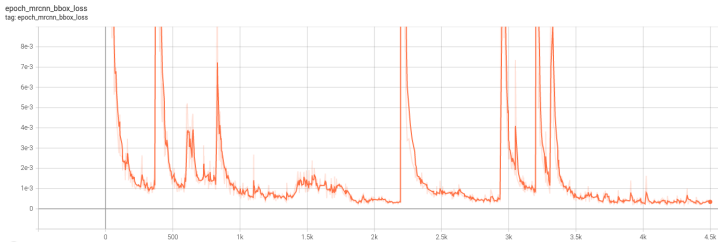
Epoch loss values oscillate between 3 and 4%, bbox loss values are approximately 0, displaying 0.1% loss value, class loss values are in the same category, oscillating between 0.3 and 0.2% and mask loss values display a range between 2.5 and 2%.

The results depicted in Figure 4.3 do not detail the full training process for model A, because an initial training was executed and the logs were lost. So, although the logs in these figures start at epoch 0, in reality, it is probably epoch 3000.

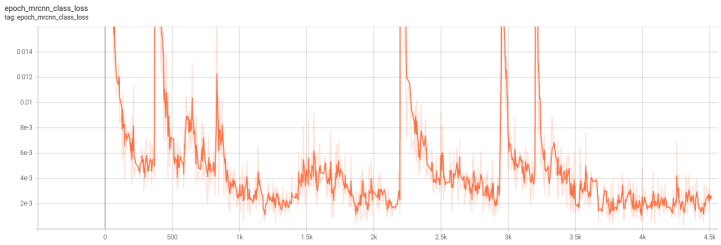
Concluding, with the analysis of the training results provided by tensorboard, it is assumable that the train was successful, displaying good loss values in all categories.



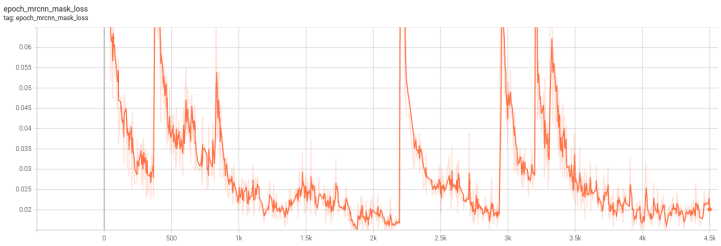

(a)




(b)




(c)




(d)

Figure 4.3: Tensorboard loss Scalars (Model A). (a) Epoch loss (b) Bbox loss (c) Class loss (d) Mask loss

## Model B training evaluation

Analysing the information displayed in Figure 4.4 it is observed, just like in the previous training, that all the graphs start to stagnate at the end, for almost

1000 steps, oscillating between the same range of loss values, giving no reason to progress with the training.

The class loss graph exhibits strange range oscillating patterns. The reason for this occurrence could be because the loss value is displayed on a small scale between 0.28% and 0.04%, and so, in reality, the value is almost static.

Epoch loss values oscillate between 1.8 and 1%, bbox loss values are approximately 0, displaying between 0.2 and 0.1% loss value, class loss values oscillate between 0.18 and 0.04% and mask loss values display a range between 1.4 and 1%.

Concluding, with the analysis of the training results provided by tensorboard, one can assume that the training was successful, displaying good results in all categories. Compared to the previous training, are observed some improvements in loss values.

## 4.2 Experimental results and analysis

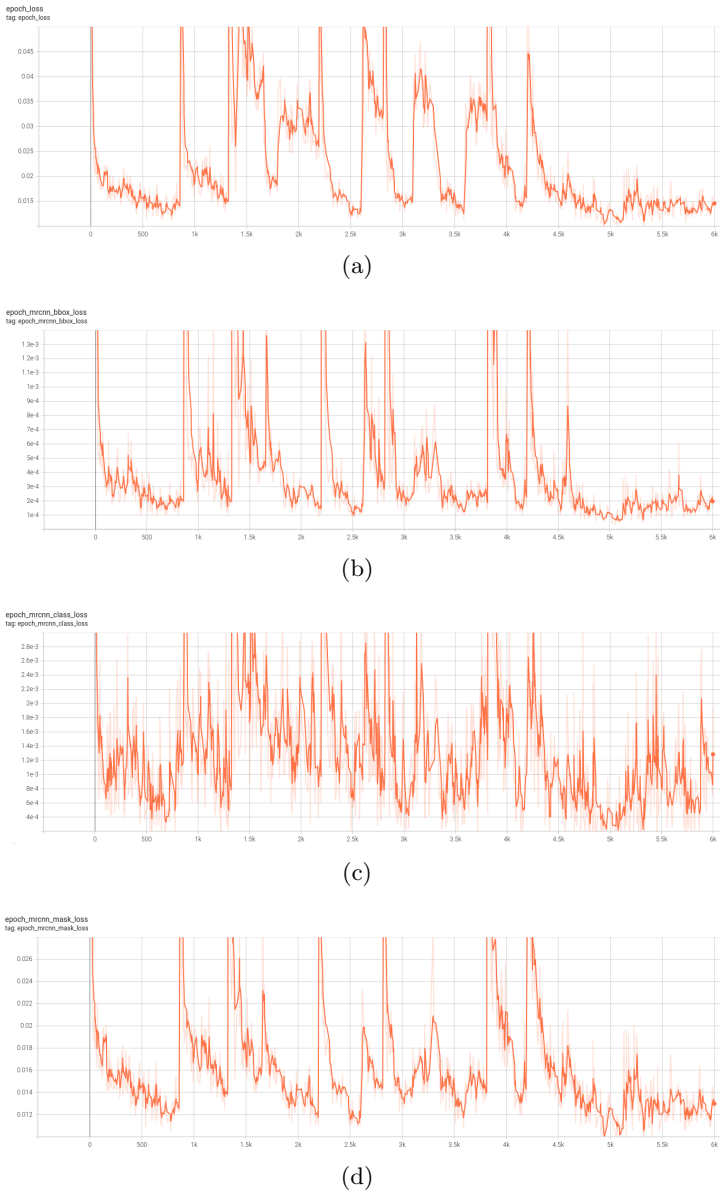
The object segmentation framework precision results are illustrated in Table 4.2, depicting all the metrics defined in the Appendix E. As observed, the mean average precision of Model A, from all the six datasets, is 0.829, almost 83% average precision on 118 images and 269 objects. The average precision for 50% IOU is 0.971 and 0.937 for 75% IOU. The mean IOU of all the instances detected is 0.905 with an average recall for 50% IOU of 0.955 and 0.904 F1 score.

Table 4.2: Metrics precision

| **Overall** | **Ap** | **Ap$_{50}$** | **Ap$_{75}$** | **mIOU** | **Ar$_{50}$** | **F1$_{50}$** |
|---|---|---|---|---|---|---|
| **Model A** | 0.829 | 0.971 | 0.937 | 0.905 | 0.955 | 0.904 |
| **Model B** | 0.751 | 0.893 | 0.819 | 0.873 | 0.890 | 0.885 |

The results of all the samples of the distinct datasets are displayed in Appendix A and B, demonstrating the metric values defined in the previous chapters in tables.

Compared to model A, model B results display a lower average score on every metric, however still exhibiting high values, around 85% on every metric. These results are expected because model B had a lower quality training compared to model A, with less training time, fewer images, just one camera viewpoint and fewer batches with a distinct number of objects. It is important to notice that the dataset to train model B was generated with only one camera viewpoint, causing difficulties in the detection with very distinct viewpoints, such as batch 5th of the evaluation dataset for model B.




(a)

(b)

(c)

(d)

Figure 4.4: Tensorboard loss Scalars (Model B). (a) Epoch loss (b) Bbox loss (c) Class loss (d) Mask loss

Comparing these results to the results of the original Mask-RCNN [25], it is observed that the results acquired with this implementation are better. The improvement is understandable due to the type of objects displayed in an image (classes), the simpler background due to the application in an industrial environment and the training phase of the neural network.

The training phase for both models was individually processed, detecting only two classes, the object and the background, and producing good results to similar

scenarios as seen in Table 4.2.

The execution time of the object detection and segmentation framework, illustrated in Table 4.3, is divided into the detection stage, segmentation and segmentation with filtering. The values presented are an average measurement from all the sample of the datasets, being the full execution time results displayed in Appendix C.

Table 4.3: Execution time (CPU)

| Overall | Instances number | Det(s) | Seg(s) | Seg + Filters(s) |
|---|---|---|---|---|
| Model A | 1 | 0.475 | 0.048 | 0.452 |
| | 2 | 0.481 | 0.070 | 0.431 |
| | 3 | 0.519 | 0.045 | 0.417 |
| | 4 | 0.518 | 0.038 | 0.483 |
| Model B | 1 | 0.464 | 0.033 | 0.299 |
| | 3 | 0.508 | 0.037 | 0.278 |
| | 5 | 0.504 | 0.028 | 0.193 |

The detection stage, on average needs around 0.49 seconds to generate the necessary masks of the input image so it can transfer them to the segmentation algorithm. The segmentation without filters requires an average of 0.04 seconds, generating the segmented point cloud of the input mask closest to the sensor, however exhibiting all the noise and outliers produced by the depth sensor in the original point cloud. The segmentation with filtering requires around 0.36 seconds, applying pass-through, voxel grid and remove outlier filters.

The average system total time is 0.53 seconds without filtering and 0.85 seconds with filtering. Therefore, the application of filters requires on average 0.32 seconds, increasing or decreasing the time required according to the number of points given to the filters. In such a manner, for now, this stage consumes a substantial amount of time, more than expected, crucial to a bin picking problem, so it will be adapted in the future to require less time.

Analysing these results, one big advantage is that the time elapsed is not dependent on the instance number, due to the segmentation algorithm structure. As displayed in Table 4.3, the only noticeable difference is in the segmentation and filtering stage between model A and model B, diverging from 0.44 seconds on average to 0.25 seconds, respectively. This difference is from the original point cloud generation; as shown later in the results, the model B point cloud has much less noise and outliers than the model A point cloud.

These average time consumed values can fluctuate according to the model resolution, which was the resolution that the images were resized when training the model. In this dissertation, the results are tested with 1024x1024 resolution,

enabling to decrease the necessary value to detect the objects by decreasing this resolution.

## 4.3 Object detection and segmentation evaluation

To conclude the evaluation, are going to be presented detection and segmentation image results, and some errors that appeared during the implementation evaluation. The results for the detection and segmentation system are presented in Sub-section 4.3.1 for model A and Sub-section 4.3.2 for model B. It is only chosen a random sample of each batch from the two datasets to illustrate the distinct outputs generated given different environments.

In addition, several results are exhibited in Appendix A and B, for model A and model B respectively, displaying the complete set of precision results in tables and between two and five images detection and segmentation.

These tests were acquired only for two models, tubular model (model A) and triangular (model B), because the two Mask-RCNN models were only trained for these specific objects, due to the solution objective. An advantage of this framework is that to be applied to a different model, the only extra requirement is neural network training before the application. Meaning that if the Mask-RCNN model accurately detects distinct instances, the solution algorithm can generate a solution for just about every object.

### 4.3.1 Detection and segmentation results (Model A)

The following images, displayed in Figure 4.5, are results from the detection and segmentation system from one random sample in each dataset of model A. As observed, the system receives the original image (left column of Figure 4.5) and detects the distinct instances in the scene, identified by colour segmentation masks for easy visualization (Middle-Left column). Analysing the results, even in novel and arduous environments, the system can detect and segment with high accuracy each object, as observed in row 5 of Figure 4.5.

The segmentation system, despite noticing a high value of noise in the point cloud data (Middle-Right column Figure 4.5), does a great job of segmenting the partial instance and reducing this noise to improve the efficiency of the succeeding systems (Right column).

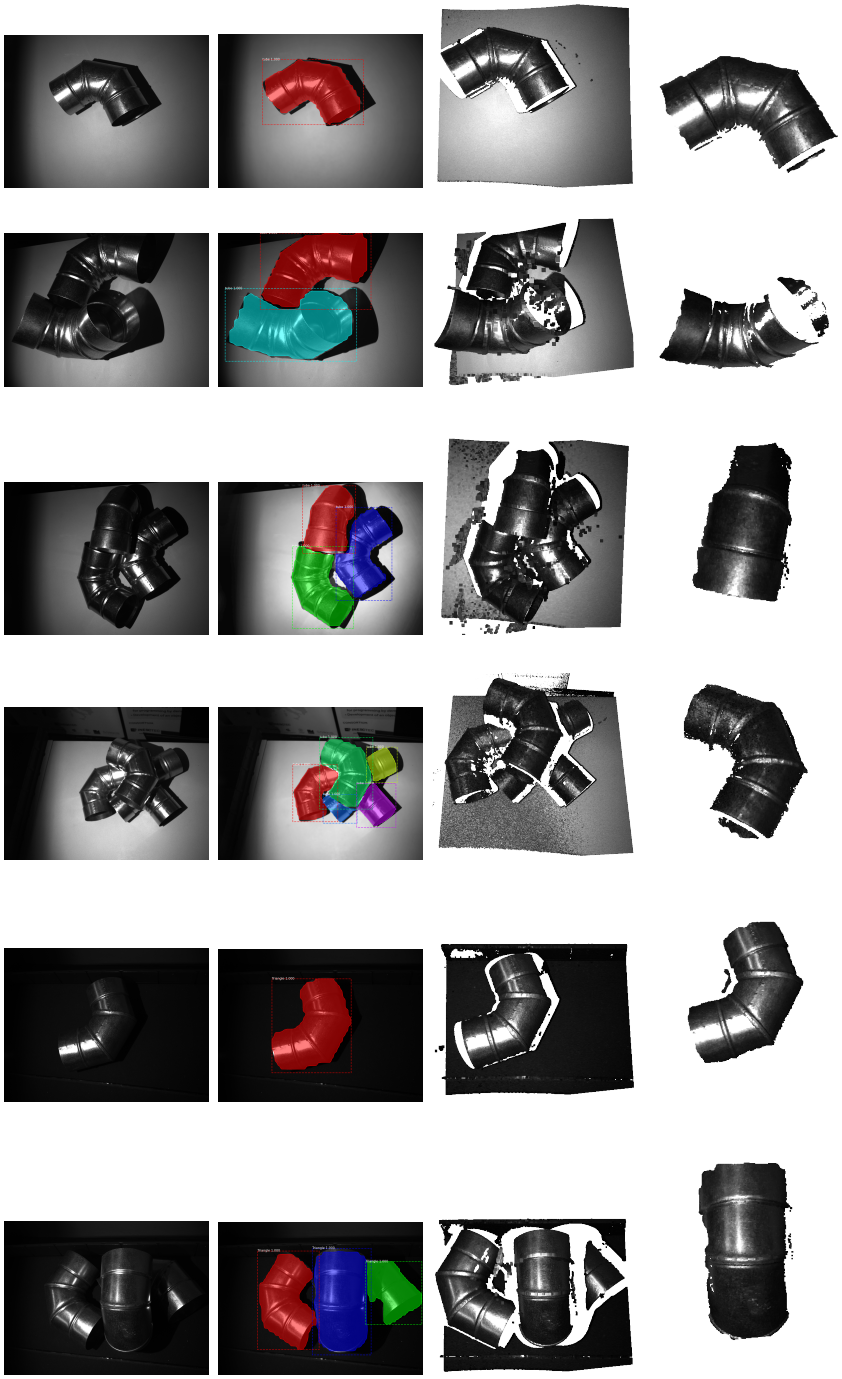


Figure 4.5: Results from evaluation dataset Model A. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

### 4.3.2 Detection and segmentation results (Model B)

The following images, displayed in Figure 4.6, are results from the detection and segmentation system from one random sample in each dataset of model B. It has the same structure as the information detailed in the previous sub-section. The results exhibited are similar to model A results, however displaying an inferior detection precision, with some flaws in dense piles of objects with occluded parts, as shown in row 2 of Figure 4.6, and a lower precision value due to the 2D mask segmentation edges.

The segmentation system with this model has an easier task due to the lack of noise and outlier in the original point clouds. Nevertheless, analysing the results, the segmentation algorithm in these five tests has a 100% ratio of identifying the top objects to pick up and does not remove important data from the partial segmented point cloud.

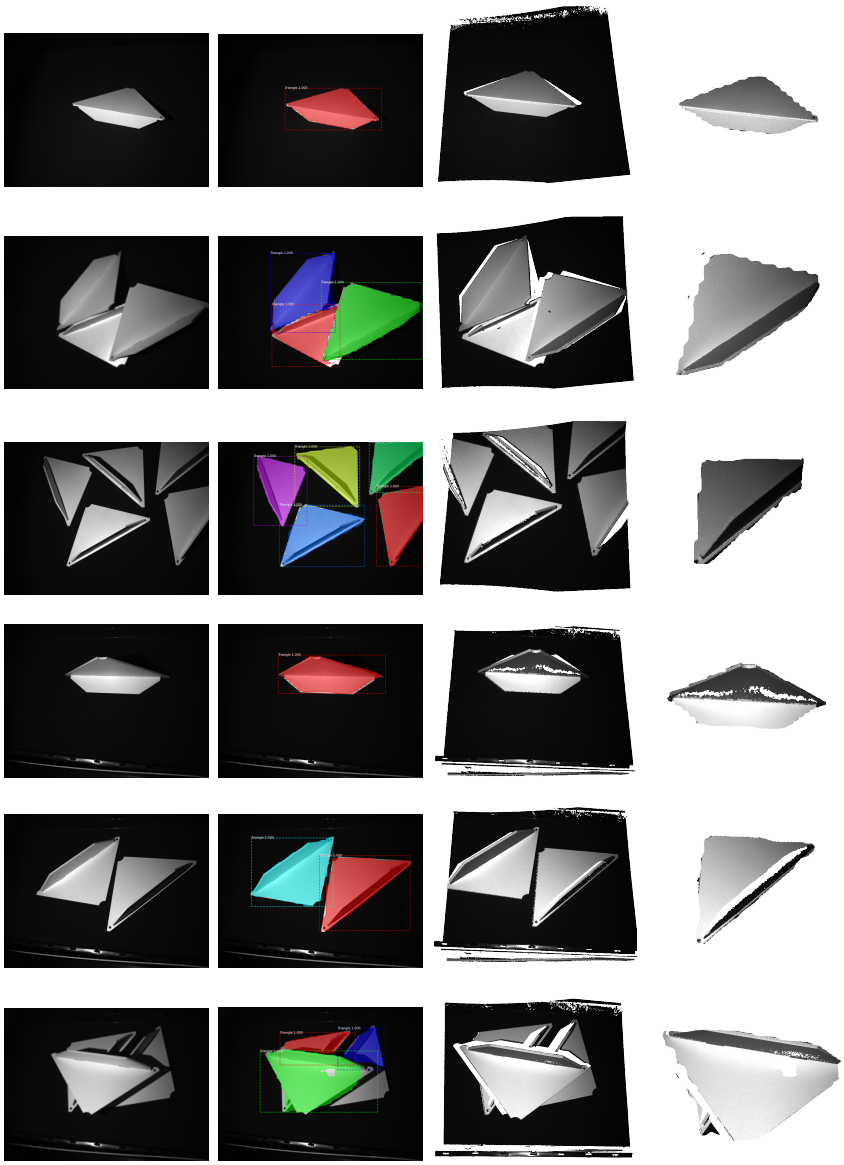
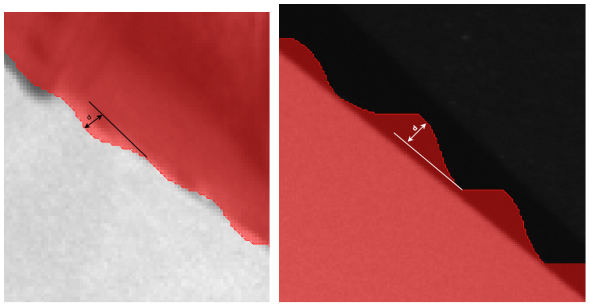


Figure 4.6: Results from evaluation dataset Model B. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

### 4.3.3 Detection and segmentation curling errors

One of the biggest misunderstandings about the detection was the curling presented in every instance, as seen in Figure 4.7. Some objects were presented with a curly mask on one side and accurate on the other. The first thought for this occurrence was the resolution of the input images given to the network model in the inference stage, by reason of, in the training stage these images were resized to 1024 pixels, and at this stage were given images with 2064 by 1544 pixels.



(a) (b)

Figure 4.7: Curling error. (a) Model A (b) Model B

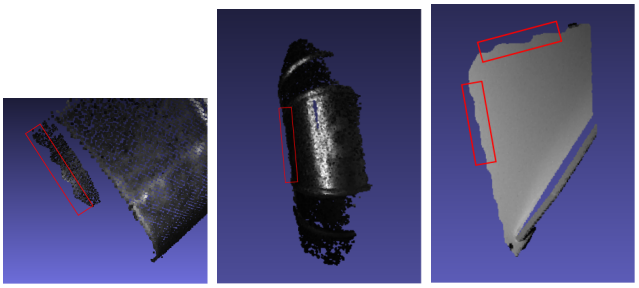
After some measurements, was concluded that the resolution was not the reason causing the incidence, but probably the lack of training, the reason being that the model with more training (tubes), displayed a lower mean curly error than the triangular model.

In the tubular model, the curly value averages 3 to 5 pixels of amplitude to the peak value, and in the triangular model, the curly value averages 15 to 17 pixels of amplitude. Considering the training difference and the similar scenarios applied to both objects, it was assumed that the difference in training was the reason for the curly circumstance.

This situation was tested in both detection and segmentation, as seen with Figures 4.7 and 4.8, because it could be only a detection and visualization error, but as demonstrated, it is possible to identify the curling in the point cloud segmentation in both models.

This error is not meaningful to the implementation, due to the pose estimation algorithm framework, as result of the matching between the CAD model and the point cloud segmentation. Furthermore, the curling peak value can be approximated to zero the better and bulkier the training stage of the Mask-RCNN model is.

The range distance is calculated in millimeters in equation 4.1. Given the 0.174 mm distance from point to point at the optimal scanning distance, where



(a) (b) (c)

Figure 4.8: Point Cloud curling. (a,b) Curling in tubular model (c) Curling in triangular model

most of the data were acquired (around 450 mm of height), specified in the sensor datasheet [28], and the image resolution (2064x1544) it is possible to estimate the scanning range (in mm) where the objects are placed, as demonstrated in Figure 3.12.

Knowing the average range in each image, and the detection errors in pixels, equation 4.2 estimates the average detection error in mm for Model A considering an error average value of 4 pixels and equation 4.3 estimates the error for model B considering 16 pixels of curling error.

Comparing the difference in values between an inference detection from a model (Model A) which had more training and a model (model B) which had less training, it is possible to declare that the training process has a significant impact on these results.

$$\begin{gathered} range\_mm = 0.174 \cdot 2064 \Leftrightarrow \\ \Leftrightarrow range\_mm \approx 360 \end{gathered} \tag{4.1}$$

$$\begin{gathered} error\_mm = (4 \cdot range\_mm)/2064 \Leftrightarrow \\ \Leftrightarrow error\_mm \approx 0.70mm \end{gathered} \tag{4.2}$$

$$\begin{gathered} error\_mm = (16 \cdot range\_mm)/2064 \Leftrightarrow \\ \Leftrightarrow error\_mm \approx 2.79mm \end{gathered} \tag{4.3}$$

### 4.3.4 Full framework results

The full framework implementation is displayed in Figures 4.9 and 4.10 to detail each step of the system. Beginning with the detection system in the top-middle figure, the segmentation system at 2nd row-left, the reference point cloud generated through CAD model (2nd row-middle), the pose estimation system (2nd

row-right), the grasping estimation system (3rd row) and the picking action (bottom).

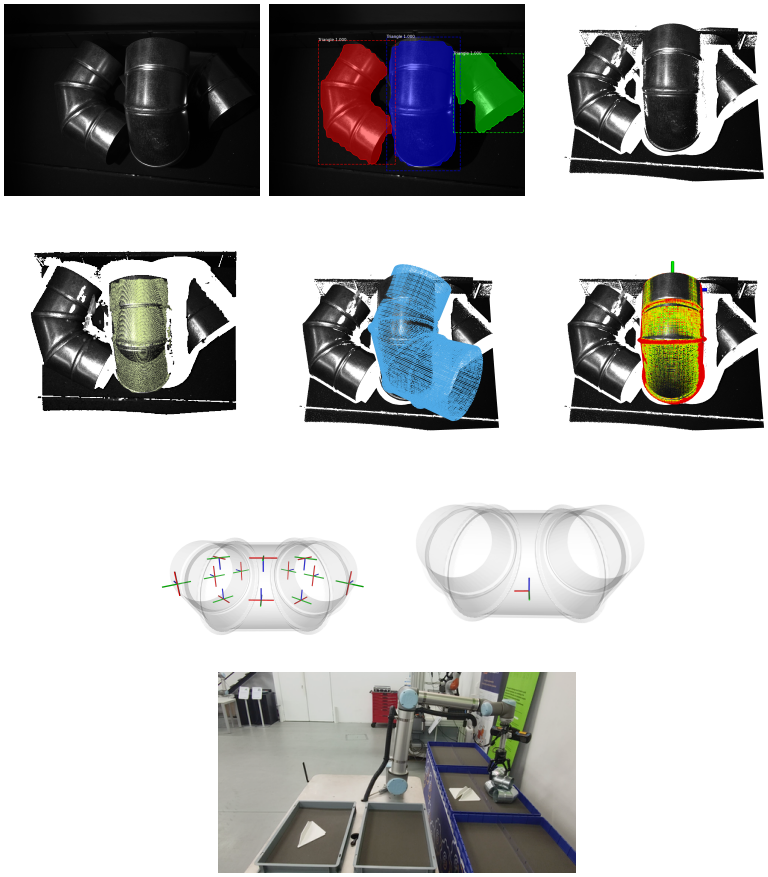


Figure 4.9: Object detection overview ($90^{o}$ elbow tube)

## 4.4 Summary

This chapter is divided into important implementation stages, such as the training stage, detection, and segmentation, crucial to the final output of the system. First, it establishes the evaluation datasets utilized to estimate the precision of the implemented system, divided into two models. Secondly, are presented important results, as images and tables across the different stages, specifying the precision of the system in distinct environments. Thirdly, are presented error occurring during the evaluation.

To conclude, is exhibited one example of the different results captured across particular stages, visualizing the full work developed by the implemented system, starting with the initial input, until the bin picking of the specified object.

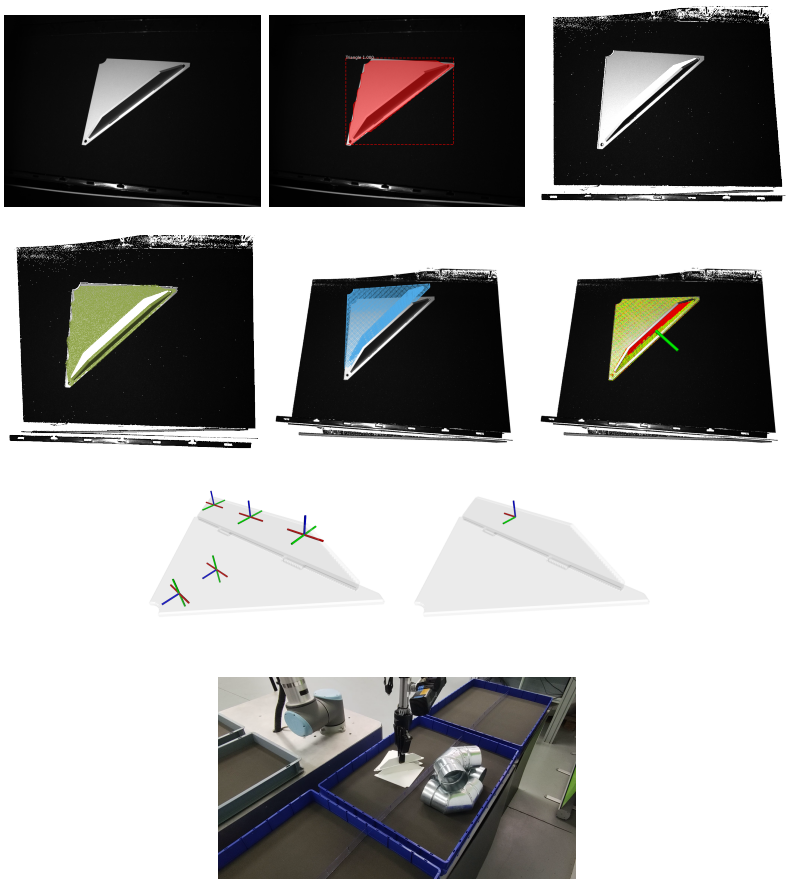


Figure 4.10: Object detection overview (Triangular wall support)

# Chapter 5

# Additional work developed

This chapter describes partial implementations that were developed alongside the dissertation. Some of this work has the objective of assisting main processes or are early approaches of an interesting work that can be further developed in future work.

The two works presents in this chapter have similar objectives: to generate instance segmentation labels from a singular 2D image, mainly diverging from a real environment to a simulated environment.

In the next section will be briefly detailed the two implementations and some of the results obtained until this date, to give an idea of the possibilities this implementation is capable of.

## 5.1 Automated dataset generation technique for one object

Automated labelling can be one of the most powerful tools in deep learning, because it can improve the efficiency of a deep learning technique development, lowering the time consumed from dataset labelling.

Most of the automated labelling techniques are implemented in a simulated environment, as the implementation detailed in the next section, considering that in this type of environment almost every variable can be controlled and known, such as the position, orientation, and number of the objects. This implementation had the initial objective of labelling a real image containing only one object, due to the high difficulty of labelling and identifying several objects in an image.

The implementation is a technique based on a two-step process. The first stage generates the object binary mask through different methods, and these methods

can vary according to the given environment. In the case of the triangular dataset, demonstrated in Figure 5.1, were applied morphological transformations, a denoise function and canny edge detection to generate the binary mask. Essentially, after the edges detection, all of the pixels between the first and last edge are given a value of 1 and the remaining as 0. With this approach was possible to detect even darker regions, as seen in Figure 5.1 (e) and (f).

The second stage converts the binary mask to a COCO JSON format annotation. This method was based on [108]. The output is a standard COCO annotation with all the information, including category names, segmentation information, original image name, total area, image ID, and bounding box. The segmentation is acquired with the generation of polygons (in this case triangles) finding contours in the mask.

Although this approach produced a good result, it can be very complex to attempt to implement a technique to label images with more than one object. In this sense the main objective was changed to automated labelling with simulated data, detailed in the following section.

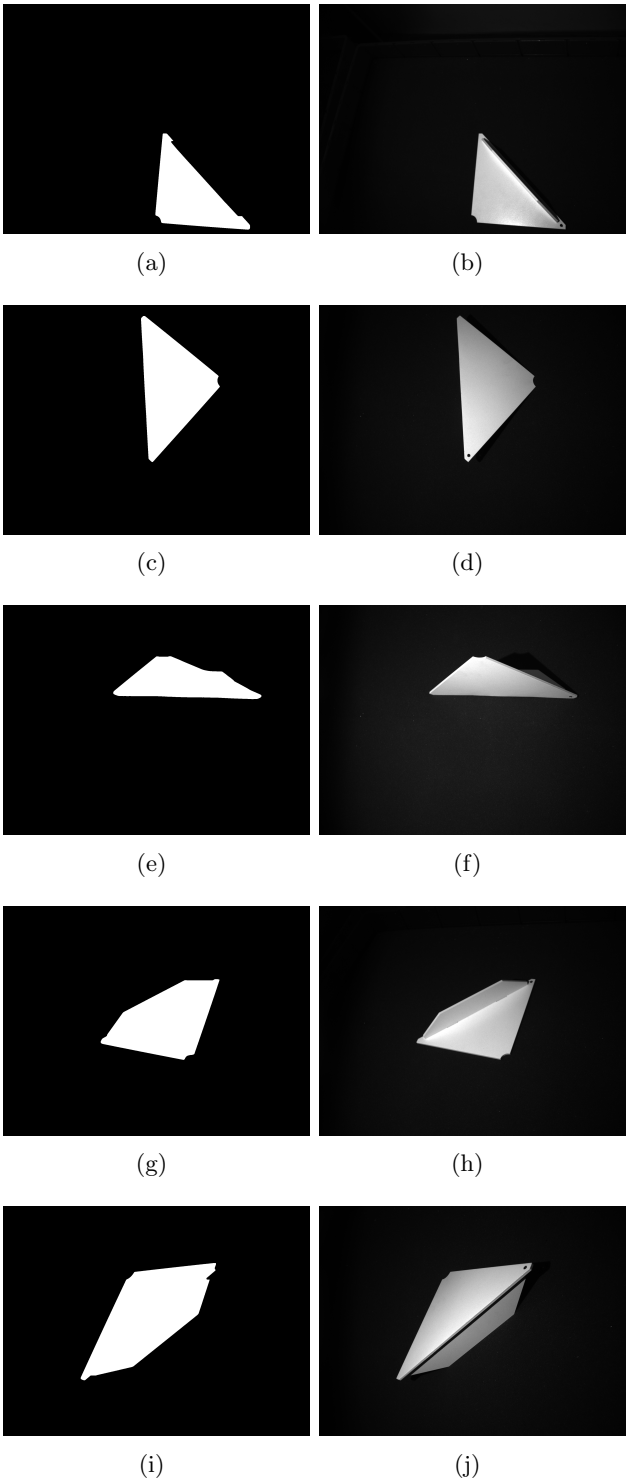
## 5.2 Simulated dataset generation

Given the difficulties identified in the previous section, of not being able to label and identify different objects in an image and the background constraints, a new solution was employed, based on blender and simulated data.

This early stage solution was developed only to test the capabilities and possibilities of blender as an automated system to generate deep learning datasets. The main objectives are to generate several simulated 2D images (grayscale or RGB) with a realistic bin picking background, alternating object poses, camera viewpoints, illumination positions within a specific area and generate an instance segmentation label of each image. For this purpose the solution needs to verify and correct all the physical impossibilities, such as an object inside a solid object, identify each object displayed in the image, execute a segmentation procedure and acquire all the information. To conclude, all the information acquired is converted to an annotation format (COCO or JSON).

The last stage of the solution is already solved because, after having the segmentation rightly identified, it can be used in a similar process to the one detailed in the previous section.

At this stage was developed a python script, a feature provided by blender, to change object positions and orientations within a certain range, verify object collisions, change camera viewpoints, and to conclude capture images of all camera viewpoints to a file, as illustrated in Figure 5.2.



(a) (b)

(c) (d)

(e) (f)

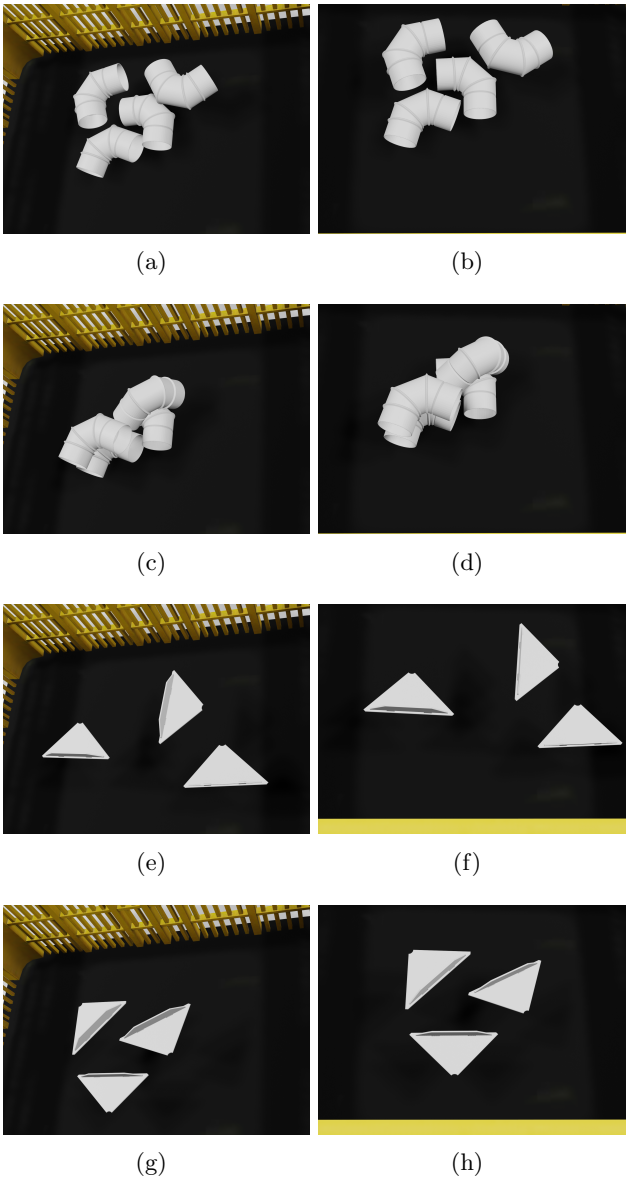
(g) (h)

(i) (j)

Figure 5.1: Automated labelling with one object results

The object labelling method was not implemented, leaving it for future work. To solve this problem two approaches were formed. In the first approach, the position and orientation of each object are acquired and transformed into a world reference and subsequently transformed to a camera reference; to conclude is applied a transformation from 3D to 2D.

In the second approach, another image is taken where the colour of each object is replaced with a unique colour and after that is applied an algorithm based on filters identifying and segmenting each object in the image.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 5.2: Simulated data

## 5.3 Summary

This chapter details the work developed in addition to the original objective, providing some context of the origins of some ideas generated in this dissertation. These works are not very detailed as to how were implemented because their only purpose is to contribute to possible future development ideas. Beyond explaining the main ideas of the systems created, are demonstrated several results obtained with these projects.

# Chapter 6

# Conclusion

This dissertation addresses bin picking problems in clutter environments using a standard robot manipulator with a parallel gripper, implementing a system based on deep learning techniques. Current bin picking approaches diverge from gripper oriented and object oriented (model free and model based) methods reviewed in Chapter 2, however very few present a deep learning system in an industrial type of environment capable of grasping several types of objects with high accuracy, i.e. two bins with objects in clutter, each with only one type of objects.

After analysing different frameworks, it was proposed in Chapter 3 an approach to detect and segment instances based on instance segmentation techniques and point cloud procedures using grayscale image and point cloud data generated with an RGB-D sensor. This implementation estimates the different instance segments in a grayscale image by applying image processing techniques and Mask-RCNN neural network. Subsequently, it is received point cloud data to estimate and extract the object (partial point cloud) closest to the sensor to be sent to the pose and grasp estimation systems, and finally the robot manipulator grasps the object through the desired position.

The proposed system is capable of segmenting several types of objects in clutter environments. However, in this case, it was only trained to detect two models, a $90^{o}$ elbow tube model (model A) and a triangular wall support model (model B), resulting in accuracy values presented in Chapter 4. For model A, the system is able to segment on average 90% of the total object area with 83% of precision, presenting 97% precision with 50% IOU and 94% precision for 75% IOU. Concerning model B, the system segments on average 87% of the total object area with 75% of precision, 89% precision for 50% IOU and 82% precision for 75% IOU. The average execution time is 0.53 seconds without the filtering process and 0.85 seconds with it. In the end, the framework sends a partial point

cloud of the object closest to the sensor, i.e the propitious object to be picked up, to the pose and grasping estimation systems, explicit at the end of Chapter 3.

The accuracy and execution time achieved by the proposed system allows to detect and segment instances with several occlusions in hard environments, being able to estimate the closer object to the sensor, which will probably be the easiest to pick up, with acceptable execution times for an industrial solution.

## 6.1 Main contribution

The main contributions of this dissertation are:

- Instance segmentation datasets:
  - Two datasets to train deep neural networks with COCO and JSON annotations
  - Two evaluation datasets with ground truth data for different instances and environments
  - One dataset with instance segmentations and keypoints
- Object segmentation system capable of:
  - Generating accurate point cloud segments to easily match with CAD models
  - Estimating the object closest to the sensor
  - Possibility of applying the system to several object types
  - Detect different objects in cluttered environments
  - Generate an accurate, efficiency and robust solution
- Mask-RCNN training with tensorflow 2.0 for datasets with COCO and JSON annotation
- Ideas to create an automated labelling system with simulated data
- One state-of-the-art article presented at ICARSC 2022 [109]
- Currently writing an implementation article

## 6.2 Future work

This work unveils several possibilities for future research.

The current implemented system can be further improved by estimating an initial pose of the output cloud, given the fact that the partial point cloud has the

same ($x$, $y$) position as the original one, improving the efficiency of the matching algorithm in the pose estimation system, and decreasing the iterations required to match the two point clouds. In addition to initial pose estimation, the system can be implemented with a multiple channel neural network input, modifying Mask-RCNN to predict the output based on, for example, two types of data (such as grayscale and depth images), instead of segmenting instances based only on a 2D image. The last future work for this system is to enhance system performance with Mask-RCNN models with less resolution and similar precision values.

Considering that the neural network training stage is very time-consuming and exhausting to create instance segment labels for all the original images, this stage can be improved with an automated image generation and labelling system, for example, using 3D creation software, such as Blender, as implemented in Chapter 5.

# References


[1] Lawtomated, "A.I. Technical: Machine vs Deep Learning." `https://lawtomated.com/a-i-technical-machine-vs-deep-learning/`, Last access: January 19 2022. [cited on p. v, 6]

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015. [cited on p. v, ix, 6, 7, 8, 9, 10]

[3] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*, vol. 1. MIT press Massachusetts, USA:, 2017. [cited on p. v, ix, 8, 9, 10]

[4] T.-T. Le and C.-Y. Lin, "Bin-picking for planar objects based on a deep learning network: A case study of USB packs," *Sensors*, vol. 19, p. 3602, Aug. 2019. [cited on p. v, vi, 10, 11, 13, 23, 24, 35, 36, 38]

[5] Z. Wu, C. Shen, and A. v. d. Hengel, "Bridging category-level and instance-level semantic image segmentation," 2016. [cited on p. v, 12]

[6] P. Wu, W. Chen, H. Liu, Y. Duan, N. Lin, and X. Chen, "Predicting grasping order in clutter environment by using both color image and points cloud," in *2019 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, IEEE, Aug. 2019. [cited on p. v, vi, 13, 23, 24, 38]

[7] H. Shin, H. Hwang, H. Yoon, and S. Lee, "Integration of deep learning-based object recognition and robot manipulator for grasping objects," in *2019 16th International Conference on Ubiquitous Robots (UR)*, IEEE, June 2019. [cited on p. v, vi, 13, 27, 36, 38]

[8] H. Park and D. Kim, "An open-source anthropomorphic robot hand system: HRI hand," *HardwareX*, vol. 7, p. e00100, Apr. 2020. [cited on p. v, 15]

[9] A. Miller, S. Knoop, H. Christensen, and P. Allen, "Automatic grasp planning using shape primitives," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, IEEE. [cited on p. v, 18, 19, 24]

[10] J. Ponce, S. Sullivan, J.-D. Boissonnat, and J.-P. Merlet, "On characterizing and computing three- and four-finger force-closure grasps of polyhedral objects," in *[1993] Proceedings IEEE International Conference on Robotics and Automation*, IEEE Comput. Soc. Press. [cited on p. v, 18, 19]

[11] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. Aparicio, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," in *Robotics: Science and Systems XIII*, Robotics: Science and Systems Foundation, July 2017. [cited on p. v, 17, 20, 35, 38]

[12] T. Birdal and S. Ilic, "Point pair features based object detection and pose estimation revisited," in *2015 International Conference on 3D Vision*, IEEE, Oct. 2015. [cited on p. v, 21, 22]

[13] J. Lee, S. Kang, and S.-Y. Park, "3d pose estimation of bin picking object using deep learning and 3d matching," in *ICINCO*, 2018. [cited on p. v, 17, 21, 22, 38]

[14] A. Zeng, K.-T. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge," 2016. [cited on p. vi, 23, 24]

[15] M. Laskey, J. Lee, C. Chuck, D. Gealy, W. Hsieh, F. T. Pokorny, A. D. Dragan, and K. Goldberg, "Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, IEEE, Aug. 2016. [cited on p. vi, 24, 25]

[16] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2016. [cited on p. vi, 7, 25]

[17] S. Kumra, S. Joshi, and F. Sahin, "Antipodal robotic grasping using generative residual convolutional neural network," 2019. [cited on p. vi, 26, 38]

[18] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kroger, J. Kuffner, and K. Goldberg, "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2016. [cited on p. vi, 17, 20, 27, 28]

[19] A. Depierre, E. Dellandréa, and L. Chen, "Jacquard: A large scale dataset for robotic grasp detection," 2018. [cited on p. vi, 17, 27, 28]

[20] "Welcome to labelme, the open annotation tool.." Accessed: 2022-04-17. [cited on p. vi, 26, 29]

[21] T. Bell, B. Li, and S. Zhang, "Structured light techniques and applications," Feb. 2016. [cited on p. vi, 30, 31, 32]

[22] D. Wan and J. Zhou, "Stereo vision using two PTZ cameras," *Computer Vision and Image Understanding*, vol. 112, pp. 184–194, Nov. 2008. [cited on p. vi, 31, 32]

[23] "Time-of-flight: what you need to know about these new means of computer vision." `https://www.avsystem.com/blog/time-of-flight/`. Accessed: 2022-05-05. [cited on p. vi, 33]

[24] Q. Shao, J. Hu, W. Wang, Y. Fang, W. Liu, J. Qi, and J. Ma, "Suction grasp region prediction using self-supervised learning for object picking in dense clutter," 2019. [cited on p. vi, 15, 34]

[25] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2017. [cited on p. vi, 11, 13, 17, 42, 43, 82]

[26] Akashdeep-47, "Facial-recognition." `https://github.com/Akashdeep-47/Facial-Recognition.git`, 2020. [cited on p. vi, 46]

[27] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Artificial Neural Networks and Machine Learning – ICANN 2018*, pp. 270–279, Springer International Publishing, 2018. [cited on p. vi, ix, 45, 46, 47]

[28] "Photoneo phoxi 3d scanner." `https://www.photoneo.com/phoxi-3d-scanner/`. Accessed: 2022-04-20. [cited on p. vi, 53, 54, 55, 89]

[29] "Fill empty space with pcl." `https://stackoverflow.com/questions/49422982/fill-empty-space-with-pcl`. Accessed: 2022-05-18. [cited on p. vi, 61]

[30] "Removing outliers using a statisticaloutlierremoval filter." `https://pcl.readthedocs.io/projects/tutorials/en/latest/statistical_outlier.html`. Accessed: 2022-05-19. [cited on p. vii, 63]

[31] J. Carvalho de Souza, C. Costa, L. Rocha, R. Arrais, A. Moreira, E. Pires, and J. Cunha, "Reconfigurable grasp planning pipeline with grasp synthesis and selection applied to picking operations in aerospace factories," *Robotics and Computer-Integrated Manufacturing*, vol. 67, 07 2020. [cited on p. vii, 2, 41, 73]

[32] F. P. Mahdi, K. Motoki, and S. Kobashi, "Optimization technique combined with deep learning method for teeth recognition in dental panoramic radiographs," *Scientific Reports*, vol. 10, Nov. 2020. [cited on p. viii, 152]

[33] O. L. F. de Carvalho, O. A. de Carvalho Júnior, A. O. de Albuquerque, P. P. de Bem, C. R. Silva, P. H. G. Ferreira, R. dos Santos de Moura, R. A. T. Gomes, R. F. Guimarães, and D. L. Borges, "Instance segmentation for large, multi-channel remote sensing imagery using mask-RCNN and a mosaicking approach," *Remote Sensing*, vol. 13, p. 39, Dec. 2020. [cited on p. viii, 152]

[34] A. M. Hafiz and G. M. Bhat, "A survey on instance segmentation: state of the art," *International Journal of Multimedia Information Retrieval*, vol. 9, pp. 171–189, July 2020. [cited on p. ix, 12]

[35] M. Grard, *Generic instance segmentation for object-oriented bin-picking.* PhD thesis, 05 2019. [cited on p. ix, 11, 16, 17, 20, 21, 22, 29]

[36] M. Zollhöfer, "Commodity RGB-d sensors: Data acquisition," in *RGB-D Image Analysis and Processing*, pp. 3–13, Springer International Publishing, 2019. [cited on p. ix, 30, 31]

[37] C. Costa, H. Sobreira, A. Sousa, and G. Veiga, "Robust 3/6 dof self-localization system with selective map update for mobile robot platforms," *Robotics and Autonomous Systems*, vol. 76, pp. 113–140, 02 2016. [cited on p. 2, 41, 72]

[38] J. McCarthy, "What is artificial intelligence?," 1998. [cited on p. 5]

[39] H. Reese, "Understanding the differences between ai, machine learning, and deep learning," *URL: https://www. techrepublic. com/article/understandingthedifferencesbetweenaimachine learningand-deeplearning*, 2017. [cited on p. 5]

[40] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, "XAI—explainable artificial intelligence," *Science Robotics*, vol. 4, Dec. 2019. [cited on p. 5]

[41] J. Kelleher, *Deep Learning.* The MIT Press Essential Knowledge series, MIT Press, 2019. [cited on p. 5, 6]

[42] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [cited on p. 6]

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84–90, May 2017. [cited on p. 6]

[44] M. Alonso, A. Izaguirre, and M. Graña, "Current research trends in robot grasping and bin picking," in *Advances in Intelligent Systems and Computing*, pp. 367–376, Springer International Publishing, June 2018. [cited on p. 6, 9, 15]

[45] C. Zhuang, Z. Wang, H. Zhao, and H. Ding, "Semantic part segmentation method based 3d object pose estimation with RGB-d images for bin-picking," *Robotics and Computer-Integrated Manufacturing*, vol. 68, p. 102086, Apr. 2021. [cited on p. 7, 12, 14, 38]

[46] L. Cheng and Z. Yang, "Grcnn: Graph recognition convolutional neural network for synthesizing programs from flow charts," 2020. [cited on p. 9]

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986. [cited on p. 9]

[48] K. Lopyrev, "Generating news headlines with recurrent neural networks," 2015. [cited on p. 10]

[49] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," 2020. [cited on p. 11]

[50] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2014. [cited on p. 12]

[51] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015. [cited on p. 13, 17]

[52] D. Buchholz, *Bin-Picking - New Approaches for a Classical Problem*. PhD thesis, Jul 2015. [cited on p. 14]

[53] J. A. Marvel, K. Saidi, R. Eastman, T. Hong, G. Cheok, and E. Messina, "Technology readiness levels for randomized bin picking," in *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, PerMIS '12, (New York, NY, USA), p. 109–113, Association for Computing Machinery, 2012. [cited on p. 14]

[54] K. Kleeberger, R. Bormann, W. Kraus, and M. F. Huber, "A survey on learning-based robotic grasping," pp. 239–249, Current Robotics Reports, 2020. [cited on p. 16, 21, 23]

[55] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013. [cited on p. 17]

[56] Y. Zhu, Y. Tian, D. Mexatas, and P. Dollár, "Semantic amodal segmentation," 2015. [cited on p. 17]

[57] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015. [cited on p. 17]

[58] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 2016. [cited on p. 17]

[59] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018. [cited on p. 17]

[60] L.-C. Chen, A. Hermans, G. Papandreou, F. Schroff, P. Wang, and H. Adam, "Masklab: Instance segmentation by refining object detection with semantic and direction features," 2017. [cited on p. 17]

[61] P. O. Pinheiro, R. Collobert, and P. Dollar, "Learning to segment object candidates," 2015. [cited on p. 17]

[62] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. [cited on p. 17]

[63] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Computer Vision – ECCV 2018*, pp. 833–851, Springer International Publishing, 2018. [cited on p. 17]

[64] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 2481–2495, Dec. 2017. [cited on p. 17]

[65] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *2011 International Conference on Computer Vision*, IEEE, Nov. 2011. [cited on p. 17, 21, 35]

[66] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg, "Dex-net 3.0: Computing robust robot vacuum suction grasp targets in point clouds using a new analytic model and deep learning," 2017. [cited on p. 17]

[67] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, "Learning ambidextrous robot grasping policies," *Science Robotics*, vol. 4, Jan. 2019. [cited on p. 17]

[68] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," 2013. [cited on p. 17, 38]

[69] M. Fujita, Y. Domae, R. Kawanishi, G. A. G. Ricardez, K. Kato, K. Shiratsuchi, R. Haraguchi, R. Araki, H. Fujiyoshi, S. Akizuki, M. Hashimoto, A. Causo, A. Noda, H. Okuda, and T. Ogasawara, "Bin-picking robot using a multi-gripper switching strategy based on object sparseness," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, Aug. 2019. [cited on p. 17]

[70] X. Markenscoff and C. H. Papadimitriou, "Optimum grip of a polygon," *The International Journal of Robotics Research*, vol. 8, pp. 17–29, Apr. 1989. [cited on p. 18, 19]

[71] B. S. Baker, S. Fortune, and E. Grosse, "Stable prehension with three fingers," in *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85*, ACM Press, 1985. [cited on p. 18]

[72] Y. hao Li, Q. jiang Lei, C. Cheng, G. Zhang, W. Wang, and Z. Xu, "A review: machine learning on robotic grasping," in *Eleventh International Conference on Machine Vision (ICMV 2018)* (D. P. Nikolaev, P. Radeva, A. Verikas, and J. Zhou, eds.), SPIE, Mar. 2019. [cited on p. 21, 23, 26]

[73] A. Pochyly, T. Kubela, V. Singule, and P. Cihak, "Robotic bin-picking system based on a revolving vision system," in *2017 19th International Conference on Electrical Drives and Power Electronics (EDPE)*, IEEE, Oct. 2017. [cited on p. 21]

[74] A. Depeursinge, "Multiscale and multidirectional biomedical texture analysis," in *Biomedical Texture Analysis*, pp. 29–53, Elsevier, 2017. [cited on p. 25]

[75] "Labeling images for bounding box object detection and segmentation." Accessed: 2022-04-18. [cited on p. 26]

[76] "Vgg image annotator (via)." Accessed: 2022-04-20. [cited on p. 26, 58]

[77] Cornell University, "Robot Learning Lab: Learning to Grasp." `https://www.kaggle.com/oneoneliu/cornell-grasp`, Last access: January 16, 2022. [cited on p. 26]

[78] University of California, Berkeley, "GQ-CNN Training Datasets." `https://berkeley.app.box.com/s/6mnb2bzi5zfa7qpwyn7uq5atb7vbztng`, Last access: January 16, 2022. [cited on p. 26]

[79] Sergey Levine, Peter Pastor, Alex Krizhevsky, Deirdre Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection." `https://sites.google.com/site/brainrobotdata/home/grasping-dataset`, Last access: January 16, 2022. [cited on p. 26]

[80] A. Depierre, E. Dellandrea, and L. Chen, "A Large-Scale Dataset for Robotic Grasp Detection." `https://jacquard.liris.cnrs.fr/database.php`, Last access: January 16, 2022. [cited on p. 26]

[81] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," 2017. [cited on p. 26, 34, 38]

[82] "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," Jan 2018. [cited on p. 27]

[83] M. B. Shaikh and D. Chai, "RGB-d data-based action recognition: A review," *Sensors*, vol. 21, p. 4246, June 2021. [cited on p. 28]

[84] X. Yang and Y. Tian, "Effective 3d action recognition using EigenJoints," *Journal of Visual Communication and Image Representation*, vol. 25, pp. 2–11, Jan. 2014. [cited on p. 29]

[85] A. Forbes, M. de Oliveira, and M. R. Dennis, "Structured light," *Nat. Photonics*, vol. 15, pp. 253–262, Apr. 2021. [cited on p. 30]

[86] J. C. A. Read, "Stereo vision and strabismus," *Eye*, vol. 29, pp. 214–224, Dec. 2014. [cited on p. 31]

[87] N. Lazaros, G. C. Sirakoulis, and A. Gasteratos, "Review of stereo vision algorithms: From software to hardware," *International Journal of Optomechatronics*, vol. 2, pp. 435–462, Nov. 2008. [cited on p. 32]

[88] L. Li, "Time-of-flight camera – an introduction," Jan. 2014. [cited on p. 32]

[89] S. Foix, G. Alenya, and C. Torras, "Lock-in time-of-flight (ToF) cameras: A survey," *IEEE Sensors Journal*, vol. 11, pp. 1917–1926, Sept. 2011. [cited on p. 33]

[90] J. J. M. Rodrigues, "3d pose estimation for bin-picking: A data-driven approach using multi-light images," 2018. [cited on p. 33, 34]

[91] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, Feb. 1992. [cited on p. 34]

[92] U. Asif, J. Tang, and S. Harrer, "GraspNet: An efficient convolutional neural network for real-time grasp detection for low-powered devices," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, International Joint Conferences on Artificial Intelligence Organization, July 2018. [cited on p. 38]

[93] Q. Shao, J. Hu, W. Wang, Y. Fang, W. Liu, J. Qi, and J. Ma, "Suction grasp region prediction using self-supervised learning for object picking in dense clutter," 2019. [cited on p. 38]

[94] A. Blank, M. Hiller, S. Zhang, A. Leser, M. Metzner, M. Lieret, J. Thielecke, and J. Franke, "6dof pose-estimation pipeline for texture-less industrial components in bin picking applications," in *2019 European Conference on Mobile Robots (ECMR)*, IEEE, Sept. 2019. [cited on p. 38]

[95] P. Jiang, Y. Ishihara, N. Sugiyama, J. Oaki, S. Tokura, A. Sugahara, and A. Ogawa, "Depth image–based deep learning of grasp planning for textureless planar-faced objects in vision-guided robotic bin-picking," *Sensors*, vol. 20, p. 706, Jan. 2020. [cited on p. 38]

[96] H. Tachikake and W. Watanabe, "A learning-based robotic bin-picking with flexibly customizable grasping conditions," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. 2020. [cited on p. 38]

[97] S. Lee and Y. Lee, "Real-time industrial bin-picking with a hybrid deep learning-engineering approach," in *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, IEEE, Feb. 2020. [cited on p. 38]

[98] A. Iriondo, E. Lazkano, and A. Ansuategi, "Affordance-based grasping point detection using graph convolutional networks for industrial bin-picking applications," *Sensors*, vol. 21, p. 816, Jan. 2021. [cited on p. 38]

[99] P. Jiang, J. Oaki, Y. Ishihara, J. Ooga, H. Han, A. Sugahara, S. Tokura, H. Eto, K. Komoda, and A. Ogawa, "Learning suction graspability considering grasp quality and robot reachability for bin-picking," 2021. [cited on p. 38]

[100] M. Q. Mohammed, L. C. Kwek, S. C. Chua, A. S. Aljaloud, A. Al-Dhaqm, Z. G. Al-Mekhlafi, and B. A. Mohammed, "Deep reinforcement learning-based robotic grasping in clutter and occlusion," *Sustainability*, vol. 13, p. 13686, Dec. 2021. [cited on p. 38]

[101] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2016. [cited on p. 43]

[102] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow." https://github.com/matterport/Mask_RCNN, 2017. [cited on p. 43, 47, 48]

[103] "Tested build configurations." https://www.tensorflow.org/install/source#gpu. Accessed: 2022-03-03. [cited on p. 47]

[104] "Mask r-cnn for object detection and segmentation using tensorflow 2.0." https://github.com/ahmedfgad/Mask-RCNN-TF2. Accessed: 2022-03-13. [cited on p. 47]

[105] J. Brooks, "COCO Annotator." https://github.com/jsbroks/coco-annotator/, 2019. [cited on p. 58]

[106] "Overview." [cited on p. 60, 62, 63, 147]

[107] "voxel_grid_filter." https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/voxel-grid-filter-design.html. Accessed: 2022-05-18. [cited on p. 62]

[108] "Convert segmentation rgb mask images to coco json format." https://github.com/chrise96/image-to-coco-json-converter. Accessed: 2022-05-04. [cited on p. 94]

[109] A. Cordeiro, L. F. Rocha, C. Costa, P. Costa, and M. F. Silva, "Bin picking approaches based on deep learning techniques: A state-of-the-art survey," in *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, IEEE, Apr. 2022. [cited on p. 100]

[110] B. Pang, E. Nijkamp, and Y. N. Wu, "Deep learning with TensorFlow: A review," *Journal of Educational and Behavioral Statistics*, vol. 45, pp. 227–248, Sept. 2019. [cited on p. 141, 142]

[111] P. Goldsborough, "A tour of tensorflow," 2016. [cited on p. 142]

[112] "Google colaboratory." https://research.google.com/colaboratory/faq.html. Accessed: 2022-05-02. [cited on p. 143]

[113] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [cited on p. 144]

[114] A. Koubaa, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Springer Publishing Company, Incorporated, 1st ed., 2016. [cited on p. 144]

[115] "Nodes." http://wiki.ros.org/Nodes. Accessed: 2022-04-15. [cited on p. 145]

[116] "Messages." `http://wiki.ros.org/Messages`. Accessed: 2022-04-15. [cited on p. 145]

[117] "Services." `http://wiki.ros.org/Services`. Accessed: 2022-04-15. [cited on p. 146]

[118] "Topics." `http://wiki.ros.org/Topics`. Accessed: 2022-04-15. [cited on p. 146]

[119] "actionlib." `http://wiki.ros.org/actionlib`. Accessed: 2022-04-15. [cited on p. 146]

[120] "Parameter server." `http://wiki.ros.org/Parameter%20Server`. Accessed: 2022-04-15. [cited on p. 146]

[121] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, May 2011. [cited on p. 147]

[122] "Pcl overview." `http://wiki.ros.org/pcl/Overview`. Accessed: 2022-05-04. [cited on p. 147]

[123] "Getting start about blender- introduction." `https://docs.blender.org/manual/en/latest/getting_started/about/introduction.html`. Accessed: 2022-05-28. [cited on p. 148]

[124] D. P. Rohe, "An optical test simulator based on the open-source blender software.," 10 2019. [cited on p. 148]

[125] "Denoising computational photography." `https://docs.opencv.org/5.x/d1/d79/group__photo__denoise.html#ga4c6b0031f56ea3f98f768881279ffe93`. Accessed: 2022-05-18. [cited on p. 148]

[126] "Histograms - 2: Histogram equalization." `https://docs.opencv.org/5.x/d5/daf/tutorial_py_histogram_equalization.html`. Accessed: 2022-05-18. [cited on p. 148]

# Appendix A

# Detection and segmentation results (Model A)

This appendix presents the results obtained from the detection and segmentation system applied to the different batches of the model A evaluation dataset. This dataset, as explain in Chapter 4, has 6 batches. The first four are images used to train the model and are divided into instance numbers. The last two batches are composed by novel images, in a arduous environment, with low illumination, providing a difficult environment to detect and identify objects. There are two types of results exhibited, distinguished by figures and tables.

Each figure is divided through batches, demonstrating some results of the several images utilized to test these systems. As observed in the next figures, the layout is constantly the same, the rows divide the different samples in the batch and the columns separate the different data acquired in each system. The detection mask is the result of the detection system according the specific input image, identified in the second column, the masks are displayed in different colors for a visualization purpose. The partial point cloud originates from the original point cloud based on the segmentation system and is exhibited in the last column.

Each table is exactly organized as the figures, however they display specific metric results about the developed framework, such as, average precision with 50% IOU(AP_50), average precision with 75% IOU (AP_75), average recall with 50% IOU (AR), average precision (AP) (measured based on an interval between 50% and 95% IOU), F1 score and IOU.

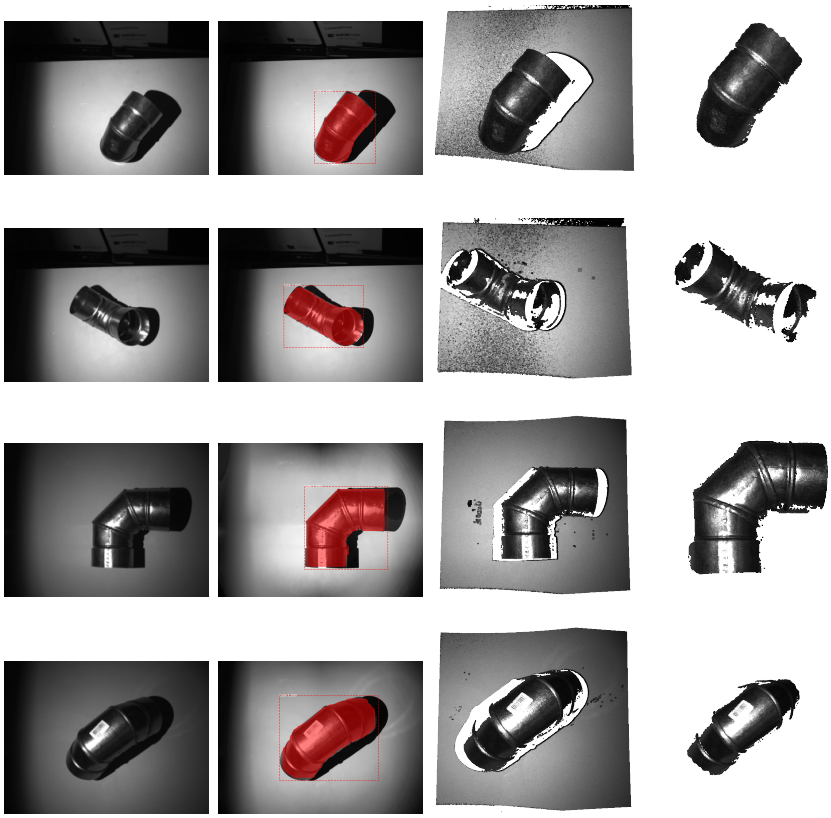


Figure A.1: Model A 1st batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

Table A.1: Model A 1st batch precision

| Samples | AP_50 | AP_75 | AR | AP | F1 | IOU |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.9 | 1 | 0.93 |
| 2 | 1 | 1 | 1 | 0.9 | 1 | 0.93 |
| 3 | 1 | 1 | 1 | 0.9 | 1 | 0.906 |
| 4 | 1 | 1 | 1 | 0.9 | 1 | 0.915 |
| 5 | 1 | 1 | 1 | 0.9 | 1 | 0.927 |
| 6 | 1 | 1 | 1 | 0.9 | 1 | 0.927 |
| 7 | 1 | 1 | 1 | 0.9 | 1 | 0.923 |
| 8 | 1 | 1 | 1 | 0.9 | 1 | 0.939 |
| 9 | 1 | 1 | 1 | 0.9 | 1 | 0.933 |
| 10 | 1 | 1 | 1 | 0.9 | 1 | 0.92 |
| 11 | 1 | 1 | 1 | 0.9 | 1 | 0.937 |
| 12 | 1 | 1 | 1 | 0.9 | 1 | 0.943 |
| 13 | 1 | 1 | 1 | 0.9 | 1 | 0.927 |
| 14 | 1 | 1 | 1 | 0.9 | 1 | 0.925 |
| 15 | 1 | 1 | 1 | 0.9 | 1 | 0.929 |
| 16 | 1 | 1 | 1 | 0.9 | 1 | 0.936 |
| 17 | 1 | 1 | 1 | 0.9 | 1 | 0.937 |
| 18 | 1 | 1 | 1 | 0.9 | 1 | 0.931 |
| 19 | 1 | 1 | 1 | 0.9 | 1 | 0.94 |
| 20 | 1 | 1 | 1 | 0.9 | 1 | 0.933 |

Table A.2: Model A 2nd batch precision

| Samples | AP_50 | AP_75 | AR | AP | F1 | IOU |
|---|---|---|---|---|---|---|
| 1 | 0.5 | 0.5 | 0.5 | 0.4 | 0.857 | 0.494 |
| 2 | 1 | 1 | 1 | 0.9 | 0.909 | 0.922 |
| 3 | 0.833 | 0.5 | 0.5 | 0.633 | 0.789 | 0.947 |
| 4 | 1 | 0.25 | 1 | 0.525 | 0.895 | 0.887 |
| 5 | 1 | 1 | 0.5 | 0.8 | 0.909 | 0.936 |
| 6 | 1 | 1 | 1 | 0.9 | 0.909 | 0.945 |
| 7 | 1 | 1 | 1 | 0.9 | 0.909 | 0.94 |
| 8 | 1 | 1 | 1 | 0.9 | 0.909 | 0.942 |
| 9 | 1 | 1 | 1 | 0.9 | 0.909 | 0.935 |
| 10 | 1 | 1 | 1 | 0.8 | 0.895 | 0.908 |
| 11 | 1 | 1 | 1 | 0.925 | 0.909 | 0.953 |
| 12 | 1 | 1 | 1 | 0.95 | 0.909 | 0.944 |
| 13 | 1 | 1 | 1 | 0.9 | 0.909 | 0.938 |
| 14 | 1 | 1 | 1 | 0.9 | 0.909 | 0.949 |
| 15 | 1 | 1 | 1 | 0.9 | 0.909 | 0.934 |
| 16 | 1 | 1 | 1 | 0.9 | 0.909 | 0.937 |
| 17 | 1 | 1 | 1 | 0.9 | 0.909 | 0.938 |

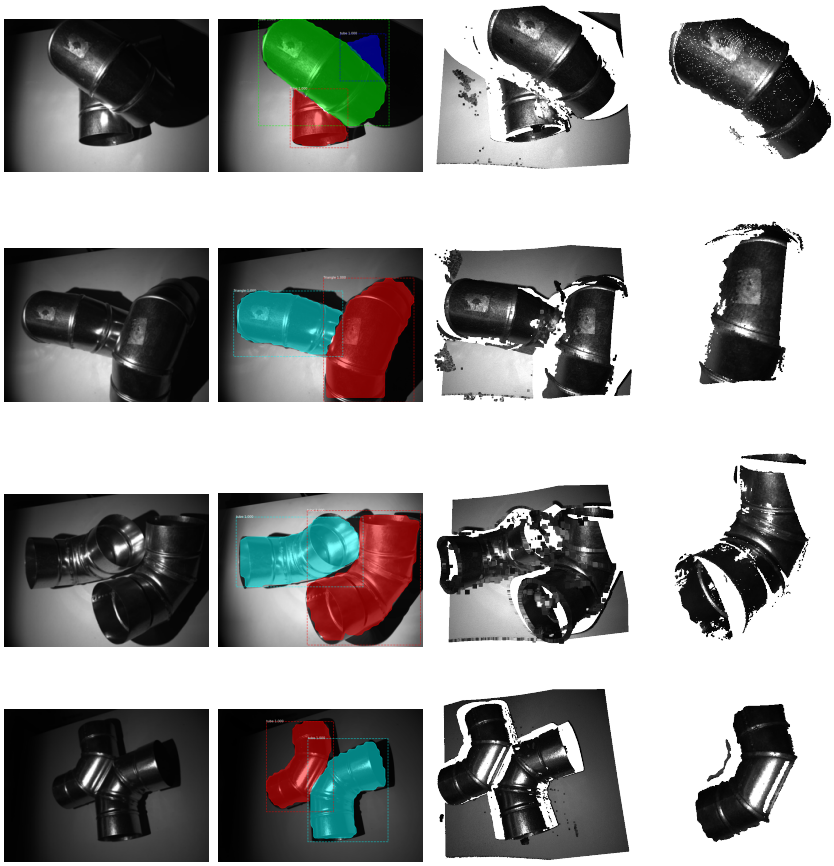


Figure A.2: Model A 2nd batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

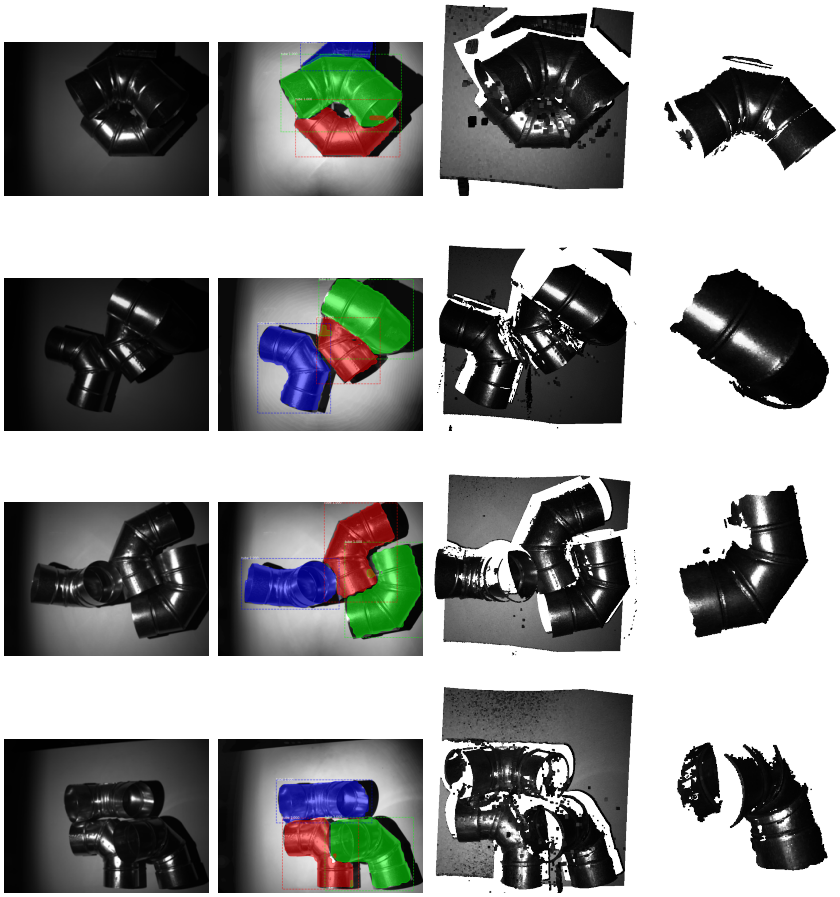


Figure A.3: Model A 3rd batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

Table A.3: Model A 3rd batch precision

| **Samples** | **AP_50** | **AP_75** | **AR** | **AP** | **F1** | **IOU** |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.9 | 0.857 | 0.926 |
| 2 | 1 | 1 | 1 | 0.9 | 0.857 | 0.946 |
| 3 | 1 | 1 | 1 | 0.9 | 0.857 | 0.925 |
| 4 | 0.667 | 0.667 | 0.667 | 0.6 | 0.8 | 0.674 |
| 5 | 1 | 1 | 1 | 0.9 | 0.857 | 0.945 |
| 6 | 1 | 0.667 | 1 | 0.567 | 0.772 | 0.921 |
| 7 | 1 | 1 | 1 | 0.833 | 0.857 | 0.939 |
| 8 | 1 | 1 | 1 | 0.833 | 0.857 | 0.923 |
| 9 | 1 | 1 | 1 | 0.9 | 0.857 | 0.913 |
| 10 | 1 | 1 | 1 | 0.911 | 0.857 | 0.926 |
| 11 | 1 | 1 | 1 | 0.9 | 0.857 | 0.928 |
| 12 | 1 | 1 | 1 | 0.844 | 0.857 | 0.919 |
| 13 | 1 | 1 | 1 | 0.933 | 0.857 | 0.939 |
| 14 | 1 | 1 | 1 | 0.9 | 0.857 | 0.921 |
| 15 | 1 | 1 | 1 | 0.917 | 0.857 | 0.938 |
| 16 | 0.667 | 0.667 | 0.667 | 0.6 | 0.8 | 0.69 |
| 17 | 1 | 1 | 1 | 0.9 | 0.857 | 0.927 |
| 18 | 1 | 1 | 1 | 0.9 | 0.857 | 0.935 |
| 19 | 1 | 1 | 1 | 0.844 | 0.857 | 0.929 |
| 20 | 1 | 1 | 1 | 0.867 | 0.857 | 0.942 |
| 21 | 1 | 1 | 1 | 0.933 | 0.857 | 0.936 |
| 22 | 1 | 1 | 1 | 0.9 | 0.857 | 0.933 |

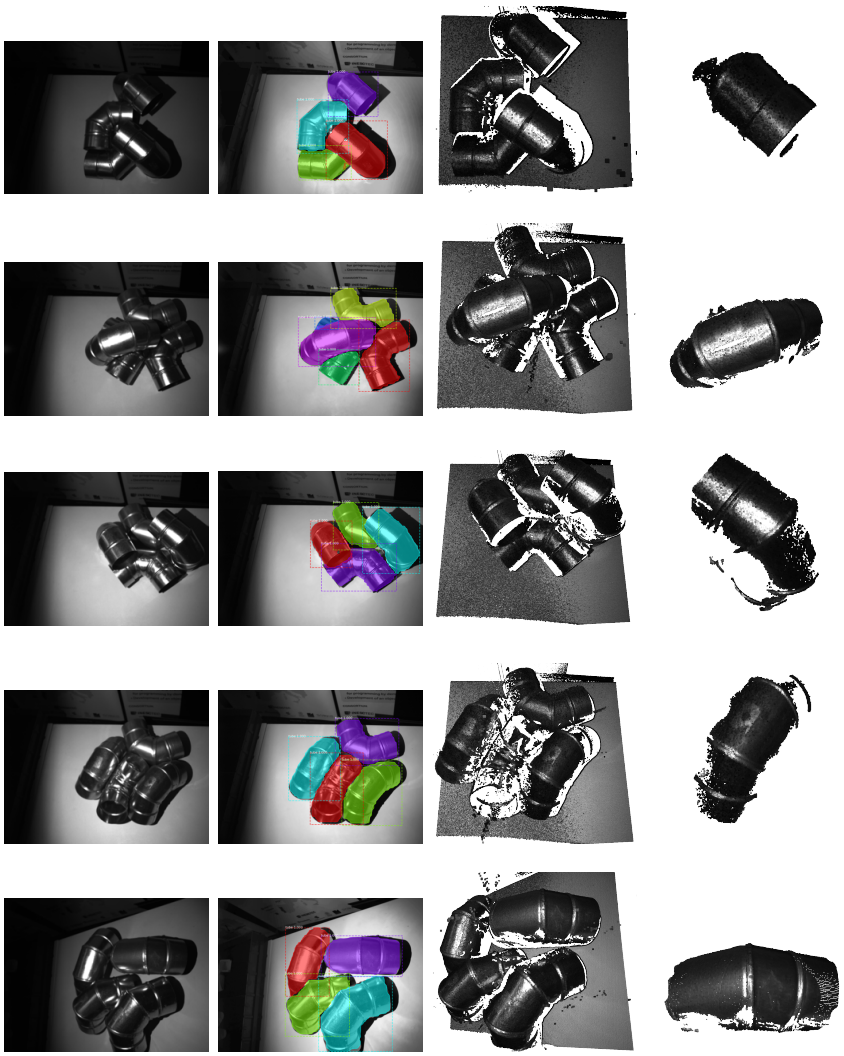


Figure A.4: Model A 4th batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

Table A.4: Model A 4th batch precision

| **Samples** | **AP_50** | **AP_75** | **AR** | **AP** | **F1** | **IOU** |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.9 | 0.824 | 0.924 |
| 2 | 1 | 1 | 1 | 0.9 | 0.824 | 0.931 |
| 3 | 1 | 1 | 1 | 0.856 | 0.824 | 0.937 |
| 4 | 1 | 1 | 1 | 0.9 | 0.824 | 0.933 |
| 5 | 1 | 1 | 1 | 0.9 | 0.824 | 0.93 |
| 6 | 1 | 1 | 1 | 0.869 | 0.824 | 0.916 |
| 7 | 1 | 1 | 1 | 0.869 | 0.824 | 0.921 |
| 8 | 1 | 1 | 1 | 0.9 | 0.824 | 0.925 |
| 9 | 1 | 1 | 1 | 0.9 | 0.824 | 0.933 |
| 10 | 1 | 0.75 | 1 | 0.75 | 0.824 | 0.906 |
| 11 | 1 | 1 | 1 | 0.808 | 0.824 | 0.922 |
| 12 | 1 | 1 | 1 | 0.9 | 0.824 | 0.921 |
| 13 | 1 | 1 | 1 | 0.875 | 0.824 | 0.902 |
| 14 | 1 | 1 | 0.75 | 0.825 | 0.824 | 0.929 |
| 15 | 1 | 0.65 | 1 | 0.685 | 0.845 | 0.938 |
| 16 | 1 | 1 | 1 | 0.9 | 0.824 | 0.927 |
| 17 | 1 | 1 | 1 | 0.856 | 0.824 | 0.936 |
| 18 | 1 | 1 | 1 | 0.9 | 0.824 | 0.937 |
| 19 | 1 | 1 | 1 | 0.9 | 0.824 | 0.922 |
| 20 | 1 | 1 | 1 | 0.808 | 0.824 | 0.92 |

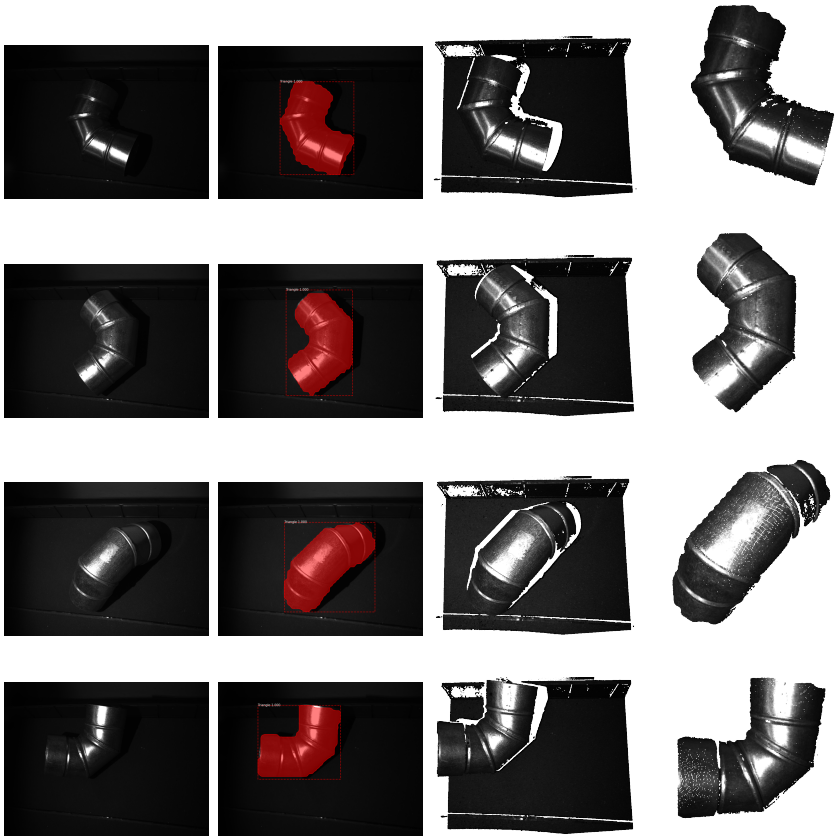


Figure A.5: Model A 5th batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

Table A.5: Model A 5th batch precision

| Samples | AP_50 | AP_75 | AR | AP | F1 | IOU |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.9 | 1 | 0.943 |
| 2 | 1 | 1 | 1 | 0.8 | 1 | 0.864 |
| 3 | 1 | 1 | 1 | 0.9 | 1 | 0.926 |
| 4 | 1 | 1 | 1 | 0.9 | 1 | 0.925 |
| 5 | 1 | 1 | 1 | 0.9 | 1 | 0.919 |
| 6 | 1 | 1 | 1 | 0.9 | 1 | 0.927 |
| 7 | 1 | 1 | 1 | 0.9 | 1 | 0.943 |
| 8 | 1 | 1 | 1 | 0.9 | 1 | 0.948 |
| 9 | 1 | 1 | 1 | 0.9 | 1 | 0.921 |
| 10 | 1 | 1 | 1 | 0.9 | 1 | 0.935 |
| 11 | 1 | 1 | 1 | 0.9 | 1 | 0.905 |
| 12 | 1 | 1 | 1 | 0.9 | 1 | 0.935 |
| 13 | 1 | 1 | 1 | 0.9 | 1 | 0.92 |
| 14 | 1 | 1 | 1 | 0.9 | 1 | 0.948 |
| 15 | 1 | 1 | 1 | 0.8 | 1 | 0.877 |
| 16 | 1 | 1 | 1 | 0.9 | 1 | 0.924 |
| 17 | 1 | 1 | 1 | 0.9 | 1 | 0.903 |
| 18 | 1 | 1 | 1 | 0.9 | 1 | 0.915 |
| 19 | 1 | 1 | 1 | 0.7 | 1 | 0.815 |
| 20 | 1 | 1 | 1 | 0.9 | 1 | 0.937 |

Table A.6: Model A 6th batch precision

| Samples | AP_50 | AP_75 | AR | AP | F1 | IOU |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.825 | 0.909 | 0.913 |
| 2 | 0.75 | 0.75 | 0.75 | 0.567 | 0.769 | 0.874 |
| 3 | 1 | 1 | 1 | 0.85 | 0.909 | 0.913 |
| 4 | 1 | 1 | 1 | 0.8 | 0.857 | 0.875 |
| 5 | 1 | 1 | 1 | 0.85 | 0.909 | 0.916 |
| 6 | 0.75 | 0.5 | 0.75 | 0.575 | 0.772 | 0.88 |
| 7 | 1 | 1 | 1 | 0.9 | 0.909 | 0.916 |
| 8 | 1 | 0.556 | 1 | 0.444 | 0.8 | 0.854 |
| 9 | 1 | 1 | 1 | 0.9 | 0.909 | 0.917 |
| 10 | 1 | 0.667 | 1 | 0.667 | 0.857 | 0.877 |
| 11 | 0.75 | 0.75 | 0.75 | 0.675 | 0.769 | 0.944 |
| 12 | 1 | 1 | 1 | 0.9 | 0.909 | 0.936 |
| 13 | 0.667 | 0.667 | 0.667 | 0.417 | 0.8 | 0.714 |
| 14 | 1 | 1 | 1 | 0.85 | 0.909 | 0.916 |
| 15 | 1 | 0.667 | 1 | 0.6 | 0.857 | 0.804 |
| 16 | 1 | 1 | 1 | 0.7 | 0.857 | 0.824 |
| 17 | 1 | 1 | 1 | 0.725 | 0.909 | 0.856 |
| 18 | 0.667 | 0.667 | 0.667 | 0.567 | 0.8 | 0.817 |

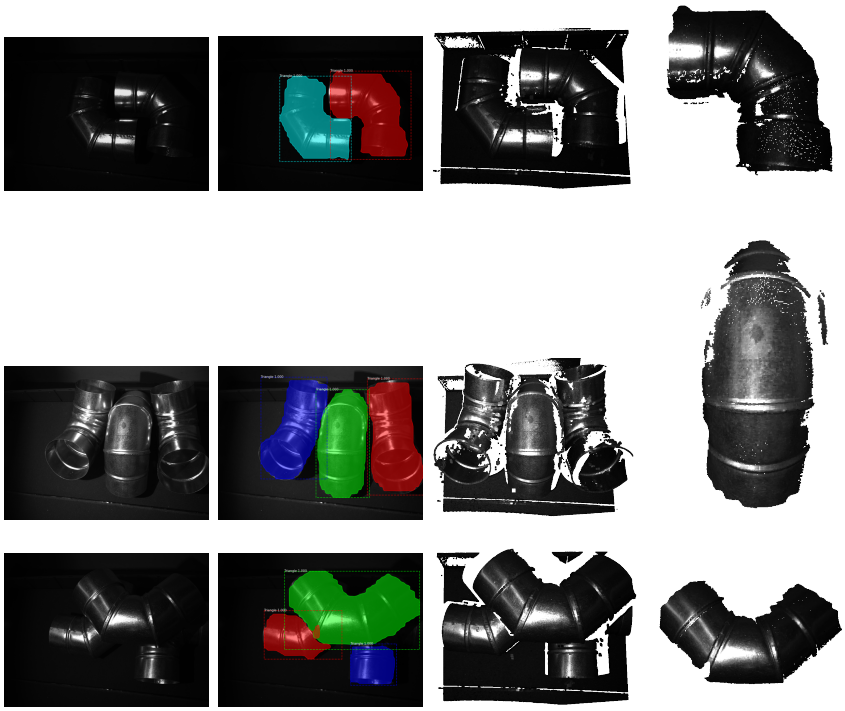


Figure A.6: Model A 6th batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

# Appendix B

# Detection and segmentation results (Model B)

This appendix presents the results obtained from the detection and segmentation system applied to the different batches of the model B evaluation dataset. This dataset, as explain in Chapter 4, has 5 batches. The first three are images used to train the model and are divided into different instance numbers (1, 3 and 5). The last two batches are composed by novel images, in a arduous environment, with low illumination, providing a difficult environment to detect and identify objects. There are two types of results exhibited, distinguished by figures and tables.

Each figure is divided through batches, presenting some results of the several images utilized to test these systems. As observed in the next figures, the layout is constantly the same, the rows divide the different samples in the batch and the columns separate the different data acquired in each system. The detection mask is the result of the detection system according the specific input image, identified in the second column, the masks are demonstrated in different colors for a visualization purpose. The partial point cloud originates from the original point cloud based on the segmentation system and is exhibited in the last column.

Each table is exactly organized as the figures, however they display specific metric results about the developed framework, such as, average precision with 50% IOU(AP_50), average precision with 75% IOU (AP_75), average recall with 50% IOU (AR), average precision (AP) (measured based on an interval between 50% and 95% IOU), F1 score and IOU.

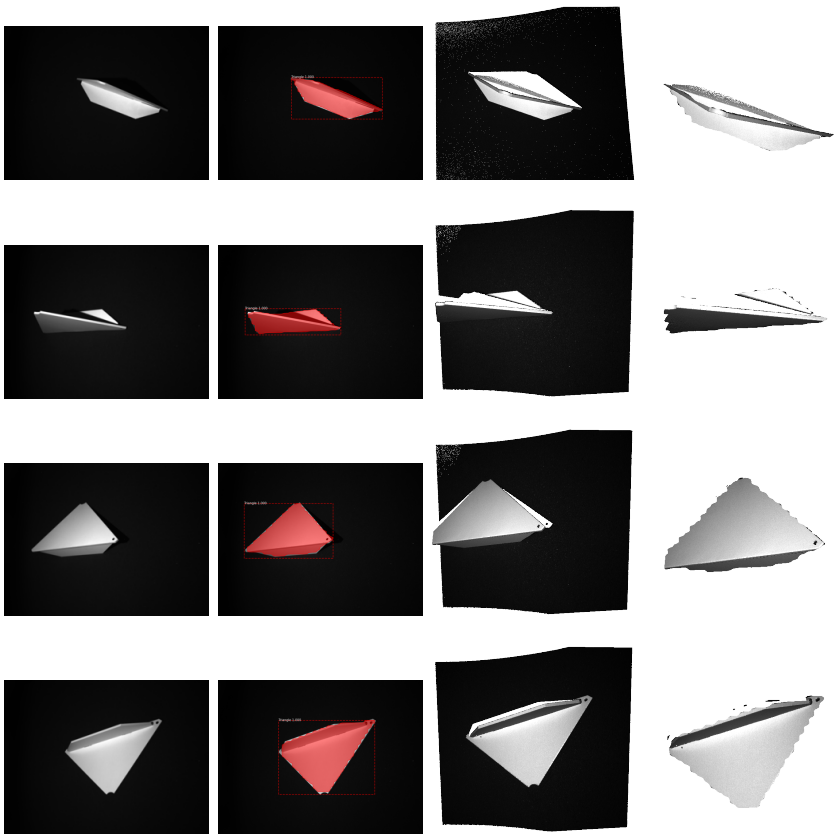


Figure B.1: Model B 1st batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

Table B.1: Model B 1st batch precision

| Samples | AP_50 | AP_75 | AR | AP | F1 | IOU |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.8 | 1 | 0.859 |
| 2 | 1 | 1 | 1 | 0.8 | 1 | 0.889 |
| 3 | 1 | 1 | 1 | 1 | 1 | 0.959 |
| 4 | 1 | 1 | 1 | 1 | 1 | 0.956 |
| 5 | 1 | 1 | 1 | 0.9 | 1 | 0.933 |
| 6 | 1 | 1 | 1 | 0.9 | 1 | 0.924 |
| 7 | 1 | 1 | 1 | 0.9 | 1 | 0.917 |
| 8 | 1 | 1 | 1 | 0.8 | 1 | 0.885 |
| 9 | 1 | 1 | 1 | 1 | 1 | 0.954 |
| 10 | 1 | 1 | 1 | 0.9 | 1 | 0.935 |
| 11 | 1 | 1 | 1 | 0.9 | 1 | 0.939 |
| 12 | 1 | 1 | 1 | 0.9 | 1 | 0.925 |
| 13 | 1 | 1 | 1 | 0.9 | 1 | 0.850 |
| 14 | 1 | 1 | 1 | 0.8 | 1 | 0.893 |
| 15 | 1 | 1 | 1 | 1 | 1 | 0.951 |
| 16 | 1 | 1 | 1 | 1 | 1 | 0.951 |
| 17 | 1 | 1 | 1 | 0.9 | 1 | 0.943 |

Table B.2: Model B 2nd batch precision

| Samples | AP_50 | AP_75 | AR | AP | F1 | IOU |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.856 | 0.857 | 0.924 |
| 2 | 1 | 1 | 1 | 0.944 | 0.857 | 0.94 |
| 3 | 1 | 1 | 1 | 0.867 | 0.857 | 0.933 |
| 4 | 1 | 1 | 1 | 0.917 | 0.857 | 0.94 |
| 5 | 1 | 1 | 1 | 0.856 | 0.857 | 0.916 |
| 6 | 1 | 1 | 1 | 0.833 | 0.857 | 0.936 |
| 7 | 1 | 1 | 1 | 0.844 | 0.857 | 0.938 |
| 8 | 1 | 1 | 1 | 0.867 | 0.857 | 0.933 |
| 9 | 1 | 1 | 1 | 0.9 | 0.857 | 0.94 |
| 10 | 1 | 1 | 1 | 0.817 | 0.857 | 0.931 |
| 11 | 1 | 1 | 1 | 0.911 | 0.857 | 0.948 |
| 12 | 1 | 1 | 1 | 0.883 | 0.857 | 0.931 |

Table B.3: Model B 3rd batch precision

| Samples | AP_50 | AP_75 | AR | AP | F1 | IOU |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.852 | 0.8 | 0.921 |
| 2 | 1 | 1 | 1 | 0.868 | 0.8 | 0.929 |
| 3 | 1 | 1 | 1 | 0.873 | 0.8 | 0.929 |
| 4 | 0.6 | 0.1 | 0.6 | 0.26 | 0.71 | 0.64 |
| 5 | 0.8 | 0.333 | 0.8 | 0.48 | 0.71 | 0.857 |
| 6 | 0.8 | 0.5 | 0.8 | 0.49 | 0.75 | 0.834 |

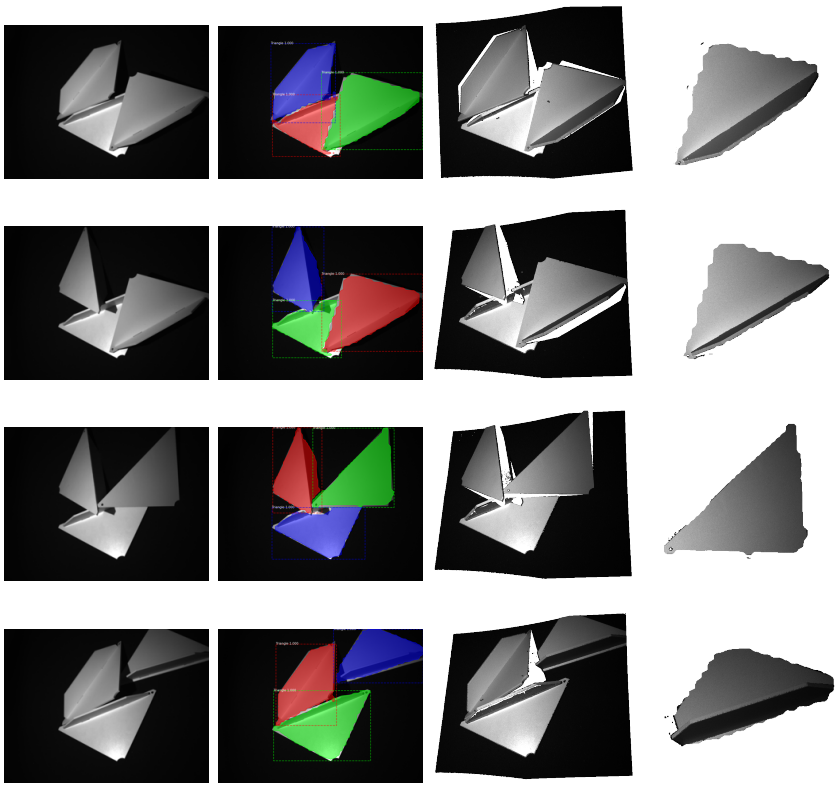


Figure B.2: Model B 2nd batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

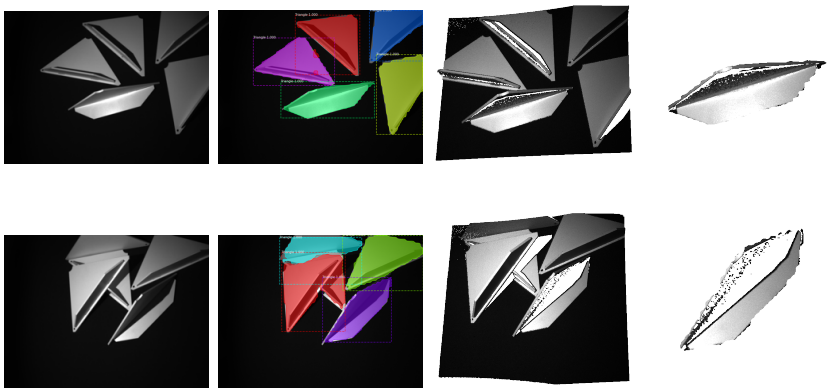


Figure B.3: Model B 3rd batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

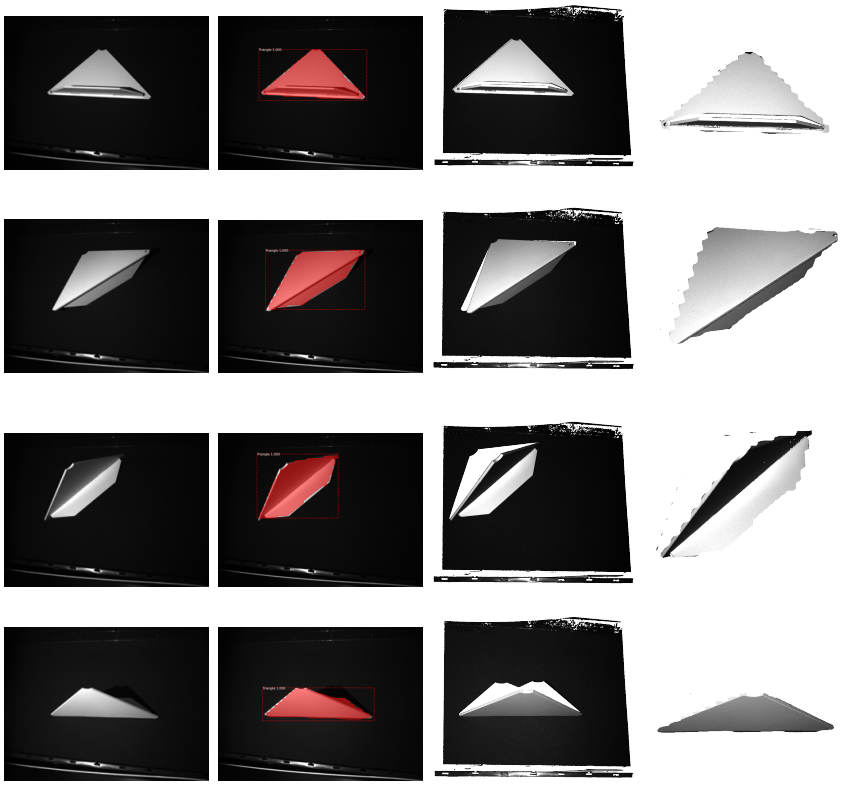


Figure B.4: Model B 4th batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

Table B.4: Model B 4th batch precision

| Samples | AP_50 | AP_75 | AR | AP | F1 | IOU |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 0.957 |
| 2 | 1 | 1 | 1 | 0.8 | 1 | 0.871 |
| 3 | 1 | 1 | 1 | 0.8 | 1 | 0.894 |
| 4 | 1 | 1 | 1 | 1 | 1 | 0.955 |
| 5 | 1 | 1 | 1 | 1 | 1 | 0.955 |
| 6 | 1 | 1 | 1 | 1 | 1 | 0.96 |
| 7 | 1 | 1 | 1 | 0.9 | 1 | 0.95 |
| 8 | 1 | 1 | 1 | 0.8 | 1 | 0.899 |
| 9 | 1 | 1 | 1 | 1 | 1 | 0.956 |
| 10 | 1 | 1 | 1 | 1 | 1 | 0.962 |
| 11 | 1 | 1 | 1 | 0.9 | 1 | 0.936 |
| 12 | 1 | 1 | 1 | 0.9 | 1 | 0.936 |
| 13 | 1 | 1 | 1 | 0.9 | 1 | 0.93 |
| 14 | 1 | 1 | 1 | 0.9 | 1 | 0.942 |
| 15 | 1 | 1 | 1 | 0.9 | 1 | 0.925 |
| 16 | 1 | 1 | 1 | 0.6 | 1 | 0.799 |
| 17 | 1 | 1 | 1 | 0.9 | 1 | 0.915 |
| 18 | 1 | 1 | 1 | 0.9 | 1 | 0.931 |
| 19 | 1 | 1 | 1 | 0.9 | 1 | 0.932 |

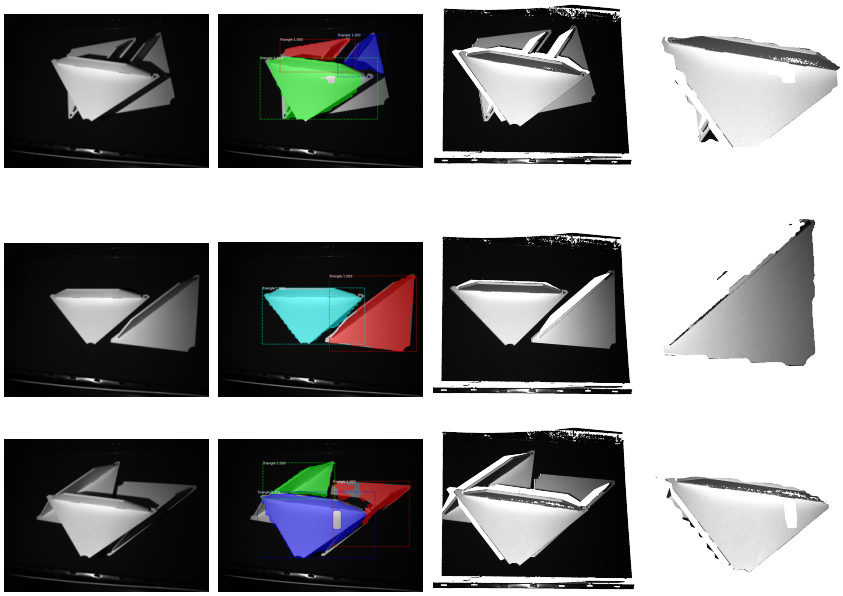


Figure B.5: Model B 5th batch evaluation results. (Left) original grayscale image (Middle-Left) Detection mask (Middle-Right) Original point cloud (Right) Partial point cloud segmentation

Table B.5: Model B 5th batch precision

| **Samples** | **AP_50** | **AP_75** | **AR** | **AP** | **F1** | **IOU** |
|---|---|---|---|---|---|---|
| 1 | 0.556 | 0.333 | 0.333 | 0.378 | 0.667 | 0.746 |
| 2 | 0.5 | 0.5 | 0.5 | 0.4 | 0.857 | 0.538 |
| 3 | 0.333 | 0 | 0.333 | 0.167 | 0.8 | 0.647 |
| 4 | 1 | 1 | 1 | 0.9 | 0.909 | 0.92 |
| 5 | 0.556 | 0.333 | 0.667 | 0.344 | 0.673 | 0.796 |
| 6 | 1 | 1 | 1 | 0.925 | 0.909 | 0.944 |
| 7 | 0.333 | 0.333 | 0.333 | 0.233 | 0.8 | 0.572 |
| 8 | 0.5 | 0 | 0.5 | 0.15 | 0.857 | 0.701 |

# Appendix C

# Metric precision and time performance

This appendix exhibits the time performance results across the evaluation datasets from model A and model B. This time is displayed in seconds, separating the measurements according to the different framework stages (detection and segmentation). Filtering was also considered to analyse the time consumed by the filters applied, to enhance their application.

The framework was executed with the processor, instead of the GPU, and with the rest of the system detailed in Section 3.2. Each table presents measurements from distinct batches of both model A and model B evaluation datasets, and the time is accurately measured with precised timers.

Table C.1: Time performance (Model A 1sr batch)

| **Samples** | **Det(s)** | **Seg(s)** | **Seg + Filters(s)** |
|---|---|---|---|
| 1 | 0.51 | 0.044 | 0.47 |
| 2 | 0.52 | 0.031 | 0.34 |
| 3 | 0.49 | 0.046 | 0.47 |
| 4 | 0.46 | 0.032 | 0.38 |
| 5 | 0.48 | 0.046 | 0.48 |
| 6 | 0.46 | 0.042 | 0.40 |
| 7 | 0.45 | 0.035 | 0.39 |
| 8 | 0.51 | 0.050 | 0.47 |
| 9 | 0.48 | 0.047 | 0.49 |
| 10 | 0.53 | 0.043 | 0.34 |
| 11 | 0.45 | 0.050 | 0.49 |
| 12 | 0.44 | 0.045 | 0.40 |
| 13 | 0.46 | 0.057 | 0.51 |
| 14 | 0.45 | 0.049 | 0.46 |
| 15 | 0.44 | 0.050 | 0.50 |
| 16 | 0.46 | 0.054 | 0.47 |
| 17 | 0.44 | 0.056 | 0.50 |
| 18 | 0.47 | 0.059 | 0.52 |
| 19 | 0.45 | 0.059 | 0.44 |
| 20 | 0.45 | 0.060 | 0.51 |

Table C.2: Time performance (Model A 2nd batch)

| **Samples** | **Det(s)** | **Seg(s)** | **Seg + Filters(s)** |
|---|---|---|---|
| 1 | 0.51 | 0.071 | 0.44 |
| 2 | 0.52 | 0.086 | 0.44 |
| 3 | 0.49 | 0.061 | 0.27 |
| 4 | 0.46 | 0.097 | 0.39 |
| 5 | 0.48 | 0.085 | 0.55 |
| 6 | 0.46 | 0.076 | 0.36 |
| 7 | 0.45 | 0.075 | 0.38 |
| 8 | 0.51 | 0.077 | 0.41 |
| 9 | 0.48 | 0.095 | 0.50 |
| 10 | 0.53 | 0.093 | 0.52 |
| 11 | 0.47 | 0.079 | 0.48 |
| 12 | 0.47 | 0.072 | 0.42 |
| 13 | 0.44 | 0.049 | 0.50 |
| 14 | 0.46 | 0.041 | 0.51 |
| 15 | 0.44 | 0.042 | 0.42 |
| 16 | 0.46 | 0.043 | 0.45 |
| 17 | 0.46 | 0.045 | 0.27 |

Table C.3: Time performance (Model A 3rd batch)

| **Samples** | **Det(s)** | **Seg(s)** | **Seg + Filters(s)** |
|---|---|---|---|
| 1 | 0.48 | 0.042 | 0.48 |
| 2 | 0.50 | 0.043 | 0.45 |
| 3 | 0.49 | 0.038 | 0.44 |
| 4 | 0.50 | 0.043 | 0.34 |
| 5 | 0.51 | 0.043 | 0.37 |
| 6 | 0.50 | 0.080 | 0.50 |
| 7 | 0.54 | 0.044 | 0.38 |
| 8 | 0.52 | 0.043 | 0.38 |
| 9 | 0.53 | 0.042 | 0.42 |
| 10 | 0.52 | 0.045 | 0.45 |
| 11 | 0.52 | 0.044 | 0.46 |
| 12 | 0.50 | 0.034 | 0.33 |
| 13 | 0.49 | 0.043 | 0.45 |
| 14 | 0.50 | 0.043 | 0.40 |
| 15 | 0.53 | 0.047 | 0.39 |
| 16 | 0.51 | 0.033 | 0.28 |
| 17 | 0.54 | 0.045 | 0.50 |
| 18 | 0.53 | 0.050 | 0.48 |
| 19 | 0.52 | 0.053 | 0.47 |
| 20 | 0.52 | 0.054 | 0.29 |
| 21 | 0.53 | 0.046 | 0.48 |
| 22 | 0.55 | 0.054 | 0.40 |

Table C.4: Time performance (Model A 4th batch)

| **Samples** | **Det(s)** | **Seg(s)** | **Seg + Filters(s)** |
|---|---|---|---|
| 1 | 0.49 | 0.039 | 0.48 |
| 2 | 0.51 | 0.032 | 0.41 |
| 3 | 0.49 | 0.040 | 0.60 |
| 4 | 0.50 | 0.033 | 0.36 |
| 5 | 0.52 | 0.050 | 0.52 |
| 6 | 0.53 | 0.039 | 0.43 |
| 7 | 0.51 | 0.039 | 0.50 |
| 8 | 0.51 | 0.038 | 0.55 |
| 9 | 0.50 | 0.037 | 0.52 |
| 10 | 0.51 | 0.037 | 0.53 |
| 11 | 0.50 | 0.033 | 0.51 |
| 12 | 0.52 | 0.044 | 0.65 |
| 13 | 0.54 | 0.036 | 0.47 |
| 14 | 0.51 | 0.039 | 0.51 |
| 15 | 0.50 | 0.035 | 0.44 |
| 16 | 0.52 | 0.034 | 0.59 |
| 17 | 0.52 | 0.033 | 0.36 |
| 18 | 0.53 | 0.035 | 0.38 |
| 19 | 0.53 | 0.042 | 0.41 |
| 20 | 0.52 | 0.038 | 0.42 |

Table C.5: Time performance (Model B 1st batch)

| **Samples** | **Det(s)** | **Seg(s)** | **Seg + Filters(s)** |
|---|---|---|---|
| 1 | 0.45 | 0.027 | 0.28 |
| 2 | 0.45 | 0.034 | 0.27 |
| 3 | 0.45 | 0.045 | 0.30 |
| 4 | 0.46 | 0.034 | 0.25 |
| 5 | 0.47 | 0.039 | 0.27 |
| 6 | 0.46 | 0.034 | 0.33 |
| 7 | 0.44 | 0.025 | 0.25 |
| 8 | 0.45 | 0.025 | 0.18 |
| 9 | 0.47 | 0.028 | 0.13 |
| 10 | 0.48 | 0.030 | 0.38 |
| 11 | 0.45 | 0.031 | 0.38 |
| 12 | 0.48 | 0.032 | 0.33 |
| 13 | 0.45 | 0.027 | 0.26 |
| 14 | 0.47 | 0.028 | 0.26 |
| 15 | 0.48 | 0.030 | 0.38 |
| 16 | 0.46 | 0.043 | 0.38 |
| 17 | 0.45 | 0.041 | 0.41 |

Table C.6: Time performance (Model B 2nd batch)

| **Samples** | **Det(s)** | **Seg(s)** | **Seg + Filters(s)** |
|---|---|---|---|
| 1 | 0.53 | 0.036 | 0.23 |
| 2 | 0.49 | 0.041 | 0.29 |
| 3 | 0.48 | 0.036 | 0.24 |
| 4 | 0.52 | 0.037 | 0.26 |
| 5 | 0.50 | 0.036 | 0.23 |
| 6 | 0.51 | 0.039 | 0.33 |
| 7 | 0.50 | 0.030 | 0.39 |
| 8 | 0.49 | 0.037 | 0.33 |
| 9 | 0.50 | 0.038 | 0.24 |
| 10 | 0.50 | 0.035 | 0.28 |
| 11 | 0.50 | 0.041 | 0.26 |
| 12 | 0.51 | 0.038 | 0.24 |

Table C.7: Time performance (Model B 3rd batch)

| **Samples** | **Det(s)** | **Seg(s)** | **Seg + Filters(s)** |
|---|---|---|---|
| 1 | 0.52 | 0.030 | 0.23 |
| 2 | 0.51 | 0.027 | 0.22 |
| 3 | 0.50 | 0.031 | 0.22 |
| 4 | 0.49 | 0.028 | 0.24 |
| 5 | 0.49 | 0.023 | 0.11 |
| 6 | 0.50 | 0.029 | 0.14 |

# Appendix D

# Resorted software and tools

This chapter introduces several software and tools utilized throughout the dissertation development. Each section briefly describes each software, such as tensorflow, ROS, and blender, it also details its features to provide a better understanding of its capabilities, and explains why these specific software applications were selected to be implemented in this dissertation.

## D.1 TensorFlow

Tensorflow is an open-source deep learning software library, launched in November 2015, developed by researchers at Google [110]. It supports defining, training, evaluating and deploying machine learning models. The core algorithms are written in highly optimized C++ and Compute Unified Device Architecture (CUDA). Apart from these languages, it can be used by other ones, like Python, adopting Application Programming Interfaces (API).

This library consists of two core sections: building a graph of computations, which is described as a construction phase, and running the computational graph (execution phase). In the first stage, the graph is composed of edges and nodes, where the edges represent data, such as vector, matrix or data arrays, and nodes represent computations, which essentially are simple mathematical operations like multiplication and addition. In the second phase (execution phase), the graph is executed over a few numbers of updating steps to train the model, these steps can be calculated from the size of the data and the batch value. In order to run through the graph and execute computation, Tensorflow provides a mechanism named tf.Session [110].

Essentially, Tensorflow was resorted to train and evaluate the neural network utilized to detect 2D shapes (polygons) in a bin picking environment. The neural

network was based on Mask-RCNN with a ResNet101 backbone, described in more detail in the section 3.1. Additionally, several features were also employed, such as tensorboard, described in section D.1.1, to evaluate step by step Mask-RCNN training process.

Similar to Tensorflow, there are other libraries, such as MLC++ (1994), OpenCV (2000), Torch (2002), Accord (2008), and Caffe (2014), among others [111], distinguished by their architecture, frontend languages, speed and objectives, focusing more on certain aspects, varying between object recognition, object segmentation, and keypoints feature, among others mostly refereed in section 2.1.1. This specific library was chosen for several reasons. In the first place, Tensorflow has a tremendous success around deep learning, secondly the name behind it, Google is probably one of the biggest and most prestigious companies in this area. Thirdly, Tensorflow has one of the most amounts of contributions comparing all libraries, meaning that has relevant community support. It has a high-level, object-oriented API, and can run on multiple platforms, just like Graphics Processing Unit (GPU), Central Process Unit (CPU) and Tensor Processing Unit (TPU). In this particular case was only executed in CPU and GPU.

### D.1.1 Tensorboard

One of the core elements of this library is tensorboard, introduced as a visualization tool. This tool was developed with the intention of being user friendly, making it easier to understand the training procedure and debug Tensorflow programs, by exhibiting computation graphs, training metrics, and parameter values of a specific model [110], visualized in Figure D.1. Tensorboard displays different parameters step by step and is possible to visualize the process values while training a model, improving the connection between the human supervisor and the machine. This feature is very useful to improve the efficiency in the training phase because the user does not need to wait until the end of the training procedure to analyse the neural network parameters. These parameters can disclose several pieces of information about this stage, for example, one of the most examined parameters is Mean Squared Error (MSE), demonstrating the loss value between the detection, such as boxes or segmentation mask with the ground truth dataset data. A higher loss value will probably represent overall bad detection results.

### D.1.2 Google Colaboratory

Google colaboratory, in short, ”colab”, is essentially a free jupyter notebook environment, that requires no set up, and runs in the cloud, with certain restrictions in terms of space with the free version. Colab also provides free access to computing resources, such as TPU and GPU, as observed in Figure D.2. In terms of GPU, is provided a TESLA K80 (Figure D.4) with 11441 mebibytes (MiB) of mem-

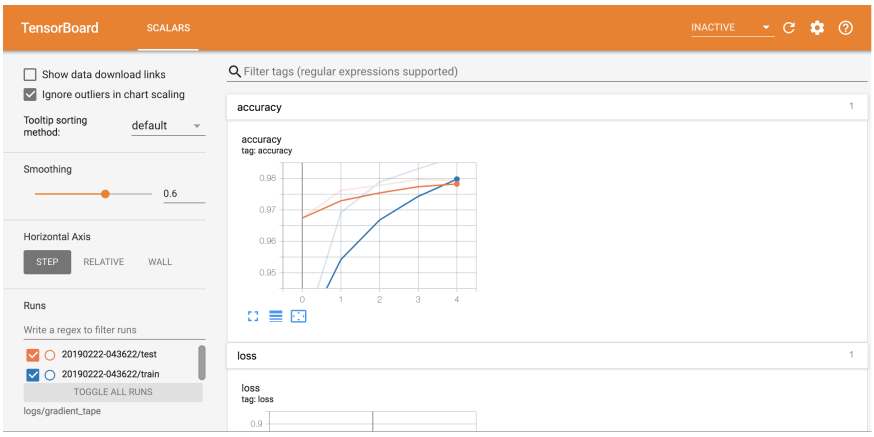



Figure D.1: Tensorboard

ory, equivalent to approximately 12 GB, and with CUDA already installed, as demonstrated in Figure D.3. This GPU enables a tremendous processing power, mainly because of the memory, allowing many features, principally in a machine learning area, including neural network training possibilities [112].

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   32C    P8    26W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure D.2: GPU version

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC455_06.29190527_0
```

Figure D.3: CUDA version

The TPU provided by Google Colab is a cloud TPU version 2. It is a hardware-based ML accelerators that can run machine learning workloads using frameworks like Tensorflow, Pytorch and JAX.

TPU v2 boards, displayed in Figure D.5, are composed of four TPU chips and 64 gibibyte (GiB) high bandwidth memory (HBM). Each chip contains two cores, and each core has a matrix multiplication unit (MMU), a vector unit, and a scalar unit. In 2018 was developed the third generation of TPUs, where the main architecture differences were: the HBM increased value by double in each chip; each TPU core has now 2 MMU, instead of one; and the increased FLoating-point Operations Per Second (FLOPS) per core.

These cloud TPU can connect and interact with each other, similar to GPU, defined as TPU Pod or TPU Pod slice. Essentially, each chip communicates

directly with the other TPU chips using high-speed interconnect. The data distributed to each TPU core is automatically handled by a TPU software.

Figure D.4: Tesla K80

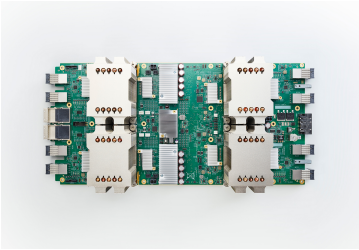


Figure D.5: TPU v2

Colab allows to combine executable code with images, HTML, LaTeX, and more. In this case, the most valuable feature of colab, is that it is possible to import images dataset, train a neural network and evaluate the model, basically is possible to apply the essential machine learning process in an online notebook.

### D.1.3 Tensorflow Hub

Tensorflow hub, as the name suggest, is a repository for machine learning models. This tool allows to search several trained machine learning models divided into different areas, such as text problem domains, image problem domains, video problem domains and audio problem domains.

## D.2 ROS

The Robot Operating System (ROS) is an open-source robotic middleware, composed of a set of software libraries and tools that helps to build robot applications [113]. The major advantage of this software is that it's all open-source, accommodating drivers, developer's tools and state of art algorithms developed from previous projects.

In this thesis, ROS is resorted as the main software, to simplify several algorithms, like acquiring data from the manipulator sensor (photonex camera), publishing and subscribing to this data, and employing packages, such as the point cloud library, as it is presented in the next section, some of these methods are referred in [114]. Essentially, ROS is used after applying object segmentation, up to the ending phase, referred to as pose estimation algorithm, in order to predict the position and orientation of objects.

### D.2.1 Nodes

ROS nodes are system processes that are combined into a graph, basically, they are executable files inside ROS packages. They can communicate with one another through topics, RPC services and the Parameter server. A node is written with a ROS client library that is either roscpp or rospy [115].

The communication between all the nodes can be provided by a master node, that in this case is the core node of ROS system, named roscore. The main core can be started using roscore command or using roslaunch for launch files.

Roscore provides a master node, a Parameter Server, detailed in D.2.6, and a rosout logging node. Allowing, not only, to produce a master node for communication between all the nodes, but also a parameter server, to share systems configuration with all the nodes.

ROS has a great feature, in terms of how it is possible to start nodes. When calling a specific node, is possible to remap all resource names, this can be done by command line or from a launch file. This feature brings robustness to the system, enabling it to adapt several nodes to different approaches. Essentially the arguments are resolved before doing the match, that way making it possible to remap the full name and not only a specific string [115].

### D.2.2 Messages

ROS messages are data structures, comprising typed fields published by ROS nodes on different topics. They support several types, such as integer, floating-point, and boolean, among others. These messages can be defined in server files (srv files), which consist of a request and response of a msg type, for example, int value. ROS nodes can call these requests and response messages as a service call. ROS messages have a specific data structure, so it can easier interpret data, with tools that automatically generate code for each message type [116].

ROS messages are referred to using package resource names, for example $sensor\_msgs/msg/PointCLoud2.msg$ can be referred as $sensor\_msgs/PointCloud2$

### D.2.3 Services

ROS services are based on a request/reply system, which is defined by messages like it was described on D.2.2. This system starts with an offer of a service from a ROS node with a specific string name and is expected the client to call the service by sending a request message to this service and awaiting the reply.

Services are defined in a srv file, that requires to be defined in CmakeList as a service file. They are persistent, meaning that a client can be constantly connected to the service, and one service can host several clients. To achieve

this is only needed an interface to service client connection, attainable with, for example, ros::ServiceClient or rospy.ServiceProxy() [117].

### D.2.4 Topics

ROS topics are intended for unidirectional, streaming communication. They are buses with a name that identifies them, so there can not be two topics with the same name, that exchange information (messages) with other nodes. A topic can have multiple publishers or subscribers, implying that is important to control the rate of this process, otherwise they are going to overwrite, and produce missed information.

In order to generate a topic, we need to identify a specific name, as mentioned, and the type of the message that it is going to be transmitted, for example, $sensor\_msgs :: PointCloud2$. It is also possible to define the rate of communication in Hertz, the higher the value the faster the topic is published or subscribed [118].

### D.2.5 Action

ROS action is a ROS service with more features. It does the same task as services, but additionally, it has other features, as the ability to cancel a certain request during execution, if it is taking too long or if the user desires, is also possible to get feedback on the executed progress, where the measurement of the progress and the identification of which progress is determined by the user. In addition, these actions can be used for pre-loading parameters for time-consuming algorithms and executing long-running goals that can be preempted [119].

### D.2.6 Parameter server

ROS parameter servers are, as the name suggests, servers to store and retrieve parameters, that can have different data types, such as strings, and integer, among others, while at runtime. These parameters can be accessed via API, like roscpp parameter API, with two different versions: the bare version, which is based on the ros::param namespace, and the handle version, based on ros::NodeHandle.

It is possible to set several parameters in a script or a yalm file, and then load them into several nodes and different areas of the system. Parameters that are received into a specific nodehandle namespace are resolved relative to that nodehandle. In addition, there is an option to define parameters only to a particular node, using private parameters, these can still be accessed from other parts of the system, but they are protected from accidental name collisions [120].

Furthermore, it is possible to check the parameters existence, delete, search, and retrieve list of parameters.

### D.2.7 Point Cloud Library (PCL)

Point Cloud Library (PCL) is an open-source project for 2D/3D image and point cloud processing, with a series of modular libraries that can be adjusted according to the needed requirements [121]. Among all the modules with state of art algorithms, the most known ones, are filters, keypoints, features, segmentation, and visualization, among many others.

PCL can be applied to ROS, consulting PCL/ROS overview, to subscribe and publish point clouds, preserving all the modules to process point clouds. In terms of pointclouds data types, ROS suports three types: $sensor\_msgs :: PointCLoud$, $sensor\_msgs :: PointCLoud2$ and $pcl :: PointCloud < T >$. These data types have different field names, for pointcloud2 we have different characteristics to specific points on a pointcloud, referred in [122], influencing the information acquired from each point.

The amount of information provided for point cloud data by PCL empowers the implementation of several state of art algorithms, some mentioned earlier in this chapter. In some cases, is not needed or even suitable for algorithms to acquire all the information from point clouds. For example, the 3d point cloud segmentation implementation, detailed in 3.6.3, only used specific point types: $(x, y, z)$, $(rgba)$, $normal_(x, y, z)$ and curvature because it was not necessary to involve more information for the following steps since the pose matching algorithm did not require more parameters.

Related to the architecture, PCL is a modern C++ library written with efficiency and performance. All the algorithms are defined in classes, simplifying the user interface, while making them compact and clean. These classes can be generalized to a generic pipeline, divided into four stages: create the processing object, input the point cloud dataset, set parameters, and call compute to get output [121].

In terms of visualization, PCL has its own library called VTK, which provides a comprehensive visualization layer for point cloud structures. This library offers: methods for rendering and setting visual properties, methods for drawing basic 3D shapes on a screen, histogram visualization module, a multitude of geometry and colour handlers, RangeImage visualization modules [121].

The main reason for the application of the PCL library in this dissertation, and in particular in the object detection framework, was to facilitate the 3d information manipulation. Given this case, filters and segmentation were the mainly resorted algorithms. More information about PCL filters and which were implemented in this algorithm, are explained in section 3.5. A good source for an in depth description of PCL filters and their utilization is [106].

## D.3 Blender

Blender is a free open-source software designed for 3D creation, such as still images, 3D animations, visual effects shots, and video editing. It has a GUI based on Open Graphics Library (OpenGL) with several features, customizable with Python scripts [123][124]. These scripts can be used to customize numerous parameters, such as object position and orientation, camera position and orientation, render distance, and image rendering, among many others, adjusting the software from a standard GUI to an autonomous data rendering software.

Blender was utilized to generate simulated data (images), and labels (2D segmentation), with different points of view, alternating the object position and orientation inside a bin picking container in each sample. It was chosen between other 3D creation suited software, considering all the features exhibited, and primarily because of the python script feature.

Concluding, in the deep learning field, this software is a good tool to develop automated labelling from simulated data, simplifying the extensive labelling process with a few lines of code.

## D.4 OpenCV functions

OpenCV provides an optimize computer vision library, tools and hardware to improve and simplify the interaction with users. In this implementation were used two OpenCV functions to equalize the image and apply denoise.

FastNlMeansDenoising is an OpenCV function that performs image denoising using a Non-local Means Denoising algorithm. It receives an input array (image), computes the weighted average value of pixels explicit in different search windows, and patches them according to a block size and the strength of the filter. Higher strength removes a lot of image noise but also details; on the other hand, lower values of strength preserve image details but also noise [125].

CLAHE is a histogram equalization that considers the global contrast of the image. An image is divided into small blocks called tiles, with a specific size (example: 8x8). If any histogram block is above the specified contrast limit, those pixels are clipped and distributed uniformly to other bins before applying histogram equalization through bilinear interpolation. The great advantage of this function is that, unlike a regular histogram equalization technique, it uses search mechanics to apply the equalization, lowering the over-brightness cases, that lose image information. On the other hand, it can enhance image noise [126].

## D.5 Summary

Chapter 3 functions as an informational stage, providing crucial information about the functional principle of each software utilized in the proposed implementation. Additionally, also explains important libraries, such as PCL, Tensorflow, among others, and the tools utilized within each library.

The information displayed in this chapter does not cover every software and library utilized in the entire implementation, but represents the essential proposed framework, giving a comprehensive review of the most important software and libraries.

# Appendix E

# Neural network evaluation index

The metrics detailed in the equations of this section, such as IoU_mask, Precision, Recall and F1, are the normal metrics utilized to evaluate a deep learning neural network.

Intersection over Union (IoU) can be related to the detected bounding box or the detected mask. In this dissertation the IoU will be always related to the detected mask, as explicit in equation E.1, representing the intersection over union of the detected mask and the ground truth mask, because the objective is to evaluate the 2D object segmentation of the system.

As observed in Figure E.1, the information shown is based on a bounding box, however, the same concept can be applied to a mask. IoU_mask quantifies the overlap percentage between the ground truth mask and the prediction mask, dividing their intersection by their union. Essentially, relates the total number of common pixels by the total number of pixels of both masks (predicted and ground truth). IoU is also common to the Dice coefficient.

$$IoU\_mask = \frac{target\_mask \cap prediction\_mask}{target\_mask \cup prediction\_mask} \tag{E.1}$$

Average precision and average recall is based on precision and recall, defined in equations E.2 and E.3.

As observed in Figure E.2, TP are the true positives, meaning that a prediction of a target mask exceeds an acceptable IoU score. False-positive (FP) indicates a predicted mask associated with no ground truth object mask, meaning that, it predicted a mask of a region that was not an object. False-negative (FN) indicates that a ground truth mask was not predicted, meaning, there was an object but the neural network did not detect it.

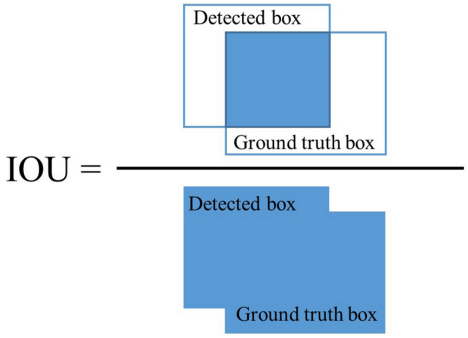



Figure E.1: Intersection Over Union [32]

$$Precision = \frac{TP}{TP + FP} \tag{E.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{E.3}$$


Figure E.2: Confusion Matrix [33]

Precision measures how accurate are the predictions, evaluating each annotation in an image, between the predicted annotation and the target annotation. It covers only the detected masks, as displayed in equation E.2

Recall measures the robustness of the detection, analysing how many positive predictions were acquired from all the ground truth data, therefore in equation E.3 the true positives are divided by TP plus FN, covering all the ground truth data.

Precision and recall are measured assuming a $IoU\_mask$ threshold, most commonly diverging from 50% and 75%. In some works it can be used 90% or demonstrated a graph between IoU and average precision, presenting all the thresholds possible.

Average precision integrates the area under a precision-recall curve, so displaying average precision is essentially displaying the relation between the precision and recall of the model. This measurement is also divided based on different values of IoU; the standard average precision with no underscore means that the measurement was acquired with an interval between 50% and 95% of IoU.

F1 Score is the weighted average of Precision and Recall, giving equal weight to precision and recall, as observed in equation E.4. If the value is high both precision and recall are high. If the value is medium, either the precision or recall is low, and if the value is low, both precision and recall are low.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{E.4}$$