FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO


# Automatic 3D Object Recognition and Localization for Robotic Grasping

**Bruno Miguel Silva Espírito Santo**

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Gil Manuel Magalhães de Andrade Gonçalves

Second Supervisor: Liliana Patrícia Saldanha Antão

July 6, 2020

# Abstract

With the advent of Industry 4.0 and its highly reconfigurable manufacturing context, the typical fixed-position grasping systems are no longer usable. This reality underlined the necessity for fully automatic and adaptable robotic grasping systems. The development of a Computer Vision system capable of detecting, identifying and estimating the 6D pose of an object would bring us closer to that reality. With that in mind, the main purpose of this thesis is to join Machine Learning models for detection and pose estimation into an automatic system to be used in a grasping environment.

To achieve this, extensive research was carried out on the current state-of-the-art approaches. The developed system uses Mask-RCNN and Densefusion models for the recognition and pose estimation of objects, respectively. The grasping is executed taking in to consideration both the pose and the object's ID, as well as allowing for user and application adaptability through an initial configuration. The system was tested both on a validation dataset and in a real world environment. The main results show that the system has more difficulty with complex objects, however, it shows promising results for simpler objects, even with training on a reduced dataset. It is also able to generalize to objects slightly different than the ones seen in training. In grasping experiments, there is a 60% success rate in the best cases, for simple grasping attempts.

# Acknowledgements

My deepest thanks to both my supervisors. It was an absolute pleasure to work on this thesis with such great people and in a very enjoyable environment, despite the difficult working situations this year. A lot was learned in developing this thesis and I could not have done it without their tireless help.

Bruno Miguel Silva Espírito Santo

# Contents

# List of Figures

# List of Tables

# List of Abbreviations and Symbols

| | |
|---|---|
| 3D | 3-dimensional |
| 6D | 6-dimensional |
| AI | Artificial Intelligence |
| (A)NN | (Artificial) Neural Network |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| CV | Computer Vision |
| DL | Deep Learning |
| DOF | Degrees of Freedom |
| FC | Fully Connected |
| Fps | Frames per second |
| GPU | Graphics Processing Unit |
| ICP | Iterative Closest Point |
| IoU | Intersection-over-Union |
| m(AP) | (Mean) Average Precision |
| ML | Machine Learning |
| NaN | Not a Number |
| PCA | Principal Component Analysis |
| RGB | Red, Green and Blue |
| ROI | Region of Interest |
| RPN | Region Proposal Network |
| SDK | Software Development Kit |
| SSIM | Structural Similarity |

# Chapter 1

# Introduction

## 1.1 Context

Object recognition and pose estimation are two of the main topics in Computer Vision, with an increasing number of solutions being explored over these last few years. Although applied to several different areas, from augmented/virtual reality to autonomous driving, one of the critical applications for these topics is robotic grasping.

In object recognition and pose estimation, as topics inside Computer Vision, the main focus is to enable computer systems to process, analyse and, ultimately, extract information or understanding of the objects to grasp from a digital image. Traditionally, this has been done using image processing techniques, which employed algorithms to enhance the quality of images and extract features using careful analysis and programming.

With the rise of automation in industry, robots have already shown to successfully execute grasping operations with these methods, contributing to increases in productivity by decreasing cycle times, and providing high-level agility and precision.

However, in this new reality of Industry 4.0, production processes are increasingly oriented towards product customization, demanding robotic solutions to be more and more flexible. Using only typical hard-coded robotic solutions where the objects' positions are fixed, this reality is not achievable due to a very restrictive implementation, when it comes to high product customization.

These types of fixed grasping solutions can not work in non-restricted environments and generally lead to the need for completely reprogramming and rearranging the implementation for each new product variant, directly impacting one of the main aspects searched in Industry 4.0: increasing reconfigurability and responsiveness. So, even though traditional approaches are considered effective, they are, in general, difficult and time-consuming, having significant limitations for generalization and applicability.

In the last decade, computer vision research has focused on applying image processing techniques in tandem with Machine Learning (ML) and Deep Learning (DL) models. This results in more efficient systems that respond better to the needs of automation.

ML is a field of research dedicated to computers' ability to interact with their environment and learn from the data collected. It is based on mathematical models, employed to enable the learning of patterns in data, and is often used to increase the system's intelligence/flexibility or when problems cannot be solved using traditional explicit programming.

On the other hand, Deep Learning is a branch of ML that describes a set of modified ML techniques inspired by the biological nervous system. Composed of networks of parallel and concurrent convolutions, as well as other mathematical operations performed directly on the input data, it acquires a group of representative heuristics between input and output data.

Due to the convincing results that have been achieved in the scope of computer vision, there is an increasing trend towards the implementation of DL algorithms in robotic grasping applications. With ML and DL, robots are able to interact with their environment and respond to various stimuli in a completely automated manner, enabling robots to use cameras to perceive their environment and even "understand" it when using these techniques.

The main challenges in the field of CV covered in this thesis are object detection/recognition and pose estimation, which will be tackled with the use of ML models. In object detection and recognition, the task is to identify objects in an image and attribute a semantic classification (identify the class it belongs to). As for pose estimation, there is a need to estimate the object's position and orientation relative to the camera.

## 1.2 Motivation

As previously mentioned, object detection/recognition and pose estimation is of great importance for robotic systems in several tasks. One of these tasks is robotic grasping: the act of a robotic manipulator handling objects in an automated manner.

For typical robotic grasping to be accomplished, the robot needs to be able to know where and how an object is positioned in its coordinate system. 6-dimensional (6D) poses, composed of 3 DOF for the position and 3 for orientation, are often used for this purpose. Contrary to typical 3D pose, 6D poses provide the rotation, giving input not only on the "where" but also on the "how."

Nevertheless, when compared to humans, robots present significantly lower success rates in grasping objects in dynamic environments, as one would expect. Humans are inherently good at perceiving their environment and the objects that surround them, identifying their characteristics, position, and deciding naturally where to grasp an object and how much force to exert on it.

With the reality of increasing Human-Robot Collaboration solutions in the industry, humans, and robots not only share physical space but work together as a team. It is of great importance for a team to have the same perception and considerations for the tasks they will fulfill.

Given this, having grasping solutions that do not take into consideration the same aspects as humans do, would lead to worse team performance due to mismatches in grasping positions for handovers, or even due to gripper damages in grasped material.

Inspired by this, motivation is provided to develop solutions that improve manipulation tasks in flexible and collaborative industrial environments, characteristic of the new industrial reality

imposed by Industry 4.0. By providing robots, in an industrial environment, with the ability to learn their grasping tasks autonomously in a similar manner to humans, collaborative robotics is also empowered.

This way, robots are given more perception of the objects in a manipulation task. Overall, robots are able to work more intuitively with human operators and even assist them in various tasks in a safe and shared environment.

## 1.3 Problem Definition

The problem of object detection and pose estimation is one of the active research that has suffered incremental improvements, as can be seen in chapter 2. Achieving accurate pose estimation of an object would enable robots to perform grasping in a fully automatic and efficient manner.

However, in order to improve robotic grasping, besides localization, providing object identification can also be useful. For this, ML is often used, classifying the semantic class of the object that can then be utilized to adapt that grasping to the object's characteristics. For instance, the gripping strength that the manipulator can apply without causing damage to the object can be inferred from its classification, or even choosing which object to pick out in a group of objects, by its characteristics.

There are already many challenges with detecting and estimating the pose of objects, arising with the variability of the surrounding environment, such as varying light conditions, occlusion of objects, and diversity of object dimensions. Other issues can also occur in terms of the accuracy of the sensors used and the types of noise introduced.

Given this, the system to be developed needs to be accurate and fast enough to work in a real industrial environment and be able to handle multiple objects of different dimensions, in situation of occlusions of part of the objects. In sum, challenges that arise in object detection and pose estimation systems, by themselves, are:

- The lighting conditions of the environment;
- Occlusion and truncation of certain objects;
- The need for models to be able to perform in real-time with low processing speeds;
- The need for high accuracy in industrial environments where systems need to be safe to operate.

There are already many object detection/classification systems, as well as pose estimation models for grasping. However, very few showing the benefits of joining both approaches for a more complete and autonomous grasping, much less fully automated or based on only camera frame information (both RGB and/or depth).

Therefore, there is a challenge for joining localization and categorization of arbitrary objects in 3D for grasping, in real-time, using live-feed camera monitoring. This would ultimately bring us a step closer for seamless Human-Robot Collaboration.

## 1.4 Objectives

Given the problem defined above, the purpose of this thesis is to explore and develop an intelligent and fully automated system for real-time object recognition and localization derived from the analysis of data from a stereo vision camera.

The detection/recognition of various objects and their localization with 6-DOF (position and orientation in 3D space) will be retrieved to then perform grasping tasks according to that data. This system should be able not only to categorize and accurately locate multiple objects but also to be robust to occlusions.

Finally, with this solution, a higher-level robot grasping is hoped to be achieved, improving flexible and autonomous robotic solutions. In sum, the following objectives are expected to be achieved by the end of the thesis:

1. Carry out extensive research of the current state-of-the-art approaches to object detection and recognition, as well as pose estimation;
2. Gain an understanding of the evolution of models in this field;
3. Create a system, based on ML, capable of detecting and classifying an object and estimating its pose with 6-DOF;
4. The system needs to be fully automated;
5. The system needs to be able to process data in real-time;
6. The ML model should be able to handle situations of occlusion;
7. A robotic manipulator should successfully and appropriately perform manipulation tasks of various objects when receiving inputs from the system;

## 1.5 Thesis Structure

The following text is composed of four chapters: Literature Review (chapter 2), 3D Object Recognition and Localization System (chapter 3), Experiments and Results (chapter 4) and, finally, Conclusions and Future Work (chapter 5). Each chapter, as well as its sections are explained in this section.

In chapter 2, the main goal is to carry out extensive research of the field in question. In this chapter, we will first briefly describe Machine Learning and its categories, followed by a more detailed look into Artificial Neural Networks and some of its architectures used in Computer Vision. A short overview of the definition and characteristics of the object's pose and rotation will also be given.

In chapter 3, first, an overview of the complete system for Automatic Object Recognition and Localization for robotic grasping is given, as well as a look at its data flow. A more in-depth

description is provided for the implementation of this system, giving detail not only on each machine learning architecture but also on each module and the system's operating modes as a State Machine.

In chapter 4, the various experiments and results are presented, as well as a discussion of those results and what they mean to the limitations of the system developed.

In chapter 5, the conclusions are made and a few ideas for future work are also presented.

# Chapter 2

# Literature Review

In this chapter, we will first briefly describe Machine Learning and its categories, followed by a more detailed look into Artificial Neural Networks and some of its architectures used in Computer Vision. A short overview of the definition and characteristics of the object's pose and rotation will also be given.

Finally, the related work regarding object detection and pose estimation is presented, detailing commonly used datasets, as well as Machine Learning methods for both object detection and identification and pose estimation.

## 2.1 Machine Learning

Machine Learning is a subset of Artificial Intelligence, and it deals with the ability of computer systems to recognize patterns and infer from them. This means that, instead of tasks having to be directly programmed, a ML model can be trained on sample data and make decisions based on the generalizations it makes from that data.

ML is of great importance in automation, enabling systems to perform both routine and complex tasks (such as analyzing large sets of data), with the added advantage of making the system flexible to changes. It has proven to be capable of solving a variety of tasks, from image detection to speech recognition, that can sometimes be considered hard for humans.

This branch of AI can be divided into three major categories: 1) Unsupervised learning, 2) Supervised learning, and 3) Reinforcement learning. Of those, only the first two will be reviewed, as they are the most relevant for the topic in question[2].

In Unsupervised learning, the data fed into the model is not labeled. This means that the ML model tries to find the patterns or characteristic features in the data it receives, creating a sort of *summary* without being offered the feedback that labeled training data would offer [2].

Methods from this category generally can not be directly applied to classification or regression problems, given that there is no precise knowledge of what the output data might be. Basically, this category exploits the unlabeled data variance and separability to evaluate feature relevance.

Some of the most known applications of unsupervised learning include:

- Clustering, which allows splitting the dataset automatically into groups ("clusters") according to similarity, often failing to treat data points as individuals

- Anomaly detection, where the goal is to detect atypical data points in the dataset automatically;

- Association mining, where sets of items that occur together often, are identified;

- Latent variable models that are usually utilized as methods for pre-processing data (such as dimensionality reduction) or facilitating data visualization tools.

The main algorithms in Unsupervised Learning include K-means and Principal Component Analysis (PCA).

On the other hand, with Supervised learning, the models receive labeled data, referred to as "training data". This means that the input consists of the "object" to be analyzed and the desired output value. This type of learning is usually done in the context of classification (map input data to output class) or regression (map input data to continuous output).

Usual algorithms in supervised learning include Logistic Regression, Support Vector Machines, or Artificial Neural Networks. The objective is the same in classification and regression: to find particular relationships in the input data that enable the model to compute the output correctly. In other words, the labels are used to calculate the correct mapping functions for the inputs received, which will enable the model to classify new unlabeled input data [3].

When performing supervised learning, model complexity should be considered. The complexity of the model usually is dependent on the nature of the training data. If the dataset is small, or if the data between classes is clearly distinct, one should choose a low-complexity model (with a high-complexity model, it would likely over-fit, i.e., it would not be able to generalize to other data points).

The problem is that labeled data is hard to obtain since human annotation is boring. Labeling may require experts or even special devices and can be very time-consuming. Given this, a possible solution is using semi-supervised learning.

This type of learning is different from the methods mentioned above since the input data is a mix of labeled and unlabeled data. Basically, it is an intermediate between supervised and unsupervised learning.

The standard process involves first clustering similar data using an unsupervised learning algorithm, and then, using the existing labeled data, the unlabeled data is labeled. The use of this method depends mainly on the application and the setting [3].

With this brief explanation of the major categories of ML and their characteristics, a more in-depth look at Artificial Neural Networks, a versatile and widely used ML model, very common in the topic of CV and object detection, will be given.

### 2.1.1 Artificial Neural Networks

A short introduction to artificial neural networks and its characteristics are provided here in brief. This type of model is one of the most prevalent in ML, due to the fact that ANNs are some of the most versatile and effective algorithms, and, as a result, they are one of the most significant subjects in the field of AI.

Most ANNs have, at their basic level, artificial neurons called "perceptrons" (Fig. 2.1).

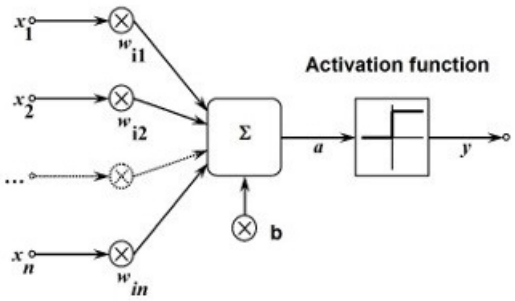



Figure 2.1: Representation of a perceptron. Image adapted from [4]

A perceptron (or neuron) receives various inputs ($x_i$) multiplied by its corresponding weights ($w_i$, numbers expressing the importance of the respective inputs to the output) and takes the sum of these results to produce a single activation (Eq. 2.1). There can also be an additional term, referred to as *bias* (b), that translates how easy it is to get the perceptron to *fire*.

$$a = \sum_{i=1}^{n} x_i \times w_i + b \tag{2.1}$$

The parameter $a$ in Eq. 2.1 is known as activation. The neuron's output ($y$) results in the *activation* of the perceptron and is dependent on the value $a$ and the activation function used. The activation function is used to introduce non-linearity into the output of the perceptron. This function is chosen accordingly to the nature of the data and the distribution of target variables. Each type of activation function entails a different fixed mathematical calculation.

A traditional Artificial Neural Network is made up of interconnected layers of perceptrons (neurons). The first layer is known as the *input layer*, the last as the *output layer* and the middle layers are referred to as *hidden layers*.

The input layer usually is equal to the number of different labels in a dataset, while the number of hidden layers can range from one to hundreds and varies according to the number of labels and computation capability.

As training data is fed through the ANN, the error between the actual and the desired outputs of the network are calculated using a *cost/loss function*; this error is then used to adjust the weights and biases of the ANN. How these adjustments are made depends on the optimization algorithm used, and on the parameters of the network that define the various types of architectures.

Moreover, the connections between neurons can also be bi-directional, leading to other different architectures. All of this contributes to the vast amount of ANNs in existence and the complexity of this subject [5]. In the next subsections, some of the most used ANN architectures in CV will be presented briefly.

#### 2.1.1.1 Autoencoders

An *Autoencoder* is an unsupervised ANN, that first learns to compress and encode data, and then learns to reconstruct back the data from the encoded version. In a nutshell, an autoencoder reduces data dimensions by learning how to ignore noise in the data. For this to happen autoencoders have four main parts: *encoder*; *botlleneck*, *decoder* and *reconstruction loss*.

An example of an autoencoder architecture is presented in Fig. 2.2, where $\hat{x}$ is the reconstruction of the original input $x$. The encoder takes its input and outputs a vector with its encoded information, compressing and reducing the input data dimensions. The bottleneck is a layer in which the input's compression representation is contained (the lowest possible dimensions of the input).

The decoder's job is to then take this compressed vector and output the closest match to the original input. The reconstruction loss function is then minimized, which helps both the encoder and decoder learn without the need for labeled data. This loss measures how well the decoder performs by calculating how close the output is to the original data.

However, if an encoder learns to match the original input exactly, there is no usefulness to the network. For that reason, restrictions are applied to limit how much of the input can be copied. This results in a function that prioritizes and only outputs the most important characteristics of its input [6].

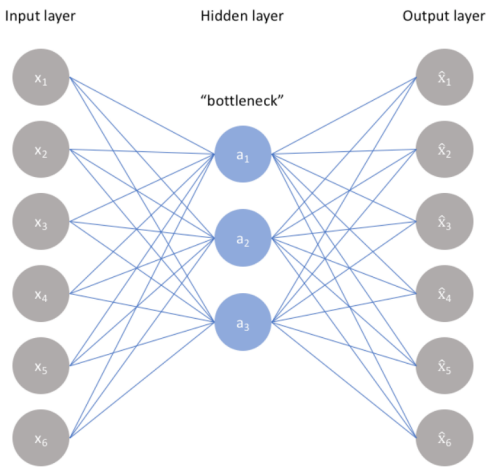



Figure 2.2: General architecture of an Autoencoder.

#### 2.1.1.2 Convolutional Neural Networks

Another type of ANN is the *Convolutional Neural Network* (CNN). Since, in a traditional neural network, all layers are fully connected, they are not able to take into account the image's spatial nature (structure and pattern). In other words, all input pixels in the image would be treated the same way, independently of being far apart or close together.

This is where CNNs are most useful. They are a class of deep neural networks (an ANN with multiple hidden layers) optimized for data in grid form, which makes it especially useful for image processing. For this reason, CNNs have evolved to be the most used method for solving various computer vision problems [7] and will appear in the majority of Section 2.3.

CNNs are identical to regular NN in terms of being composed by neurons, where the goal is to estimate weights and bias. However, CNNs have a much more efficient structure for image processing than NNs, given that each neuron is only connected to a specific region of the previous layer. The general architecture of a NN and CNN are presented in Fig. 2.3 for comparison. Each layer of a CNN is represented as 3D volume [8].

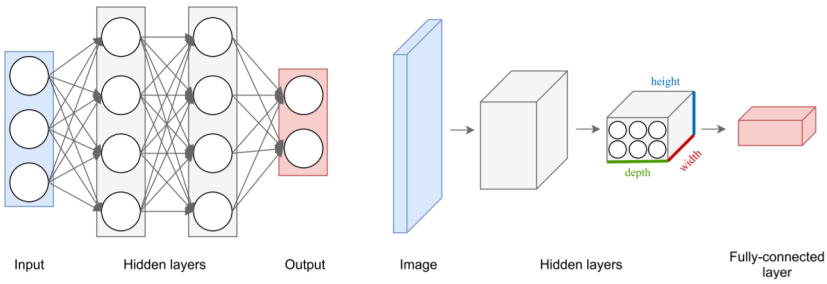



Figure 2.3: NN architecture VS CNN architecture. A standard NN is represented on the left, and a CNN on the right. Each circle represents a neuron [8].

This class of network uses variants of the mathematical operation *convolution*. Convolution (denoted as $*$) can be seen as the matrix multiplication of an input $I$ by a kernel $K$ (as seen in Eq. 2.2). Using a kernel with a size significantly inferior to the input size, a CNN can extract small features of an image. This also results in fewer calculations and, therefore, an increase in speed.

$$C(i,j) = (I * K)(i,j) = \sum_{i=1}^{m} \sum_{i=1}^{n} I(i-m)(j-n)K(m,n) \tag{2.2}$$

Typically, the layers of these networks are made up of three stages: convolution stage, detector stage, and pooling stage. In the convolution stage, multiple parallel convolutions are performed and result in a collection of linear activations. After that, in the detector stage, a non-linear activation function is used on the linear activations.

The problem arises due to the fact that these layers encode the information in the precise position it is in, meaning that if these layers process the same input with a slight change, it will result in a different output. This is where pooling functions are useful, making the network resistant, or *invariant*, to small changes in the input.

The most common pooling functions used are average pooling, max pooling, and global pooling. In average pooling, the average of all values within a patch of the input is calculated for every patch in the image. In max pooling, the maximum value in each patch is outputted. Global pooling, however, calculates either the average or maximum values of the entire input, reducing it to only one value.

The resulting outputs of each neuron in a layer are called *feature maps*, and they represent the features extracted from the input. Traversing a CNN along its layers results in increasingly more abstract feature maps that represent more complex features [7].

There are some types of CNNs which are used by many models. They are pre-trained and state-of-the-art in terms of efficiency and/or speed. These are often referred to as the *backbone architecture* of an ANN model. In the following items, we will provide a brief description of some of the most popular networks of this kind:

- AlexNet - First introduced in [9], was one of the first networks that achieved significant accuracy on the ImageNet dataset for image classification. It utilizes five convolutional layers, followed by three fully connected (FC) layers. This network improved upon its predecessors in both training speed and in reducing overfitting (when the network learns to predict its training examples but does not adjust to new ones).

- VGG [10] - Followed AlexNet and replaced large convolutional filters by multiple 3x3 kernels in a row. This had the effect of increasing the depth of the network and, consequently, its ability to learn more complex features while reducing the computation cost.

- Inception [11] - Built on the idea that some neurons will have null activations or will be redundant and, therefore, there was no need to have FC layers throughout the whole network. The Inception model has sparse connections between layers and small convolutional kernels of different sizes, enabling feature extraction at varying scales. The final change was introducing global average pooling after the last convolutional layer, reducing the total number of parameters. This network improved on both accuracy and speed, in relation to VGG.

- ResNet [12] - Increasing the depth of a network often increases its accuracy. However, the more layers that are added, the more the first layers become negligible - this is known as the *vanishing gradient* problem. This architecture was created to solve this problem by dividing a high depth network into various modules of networks. Each module follows the architecture of VGG but is independent of the others. The connections between modules are made by *identity layers*, which work much like multiplication by the identity matrix. This means that information between modules stays the same while enabling each module to have a training error that is unaffected by the others.

## 2.2 Pose and Rotations

In the scope of this thesis, it is also essential to define what is meant by the *pose* of an object. Pose with 6-DOF refers to the coordinates of an object in 3D space (translation) and the rotation of the object around the x,y, and z-axis, also known as *pitch*, *roll*, and *yaw*.

There are many representations of rotation that can be used. The three representations that are mostly used, and therefore will be of interest in this thesis are rotation matrices, Euler angles, and quaternions.

A vector $v = (x, y, z)$ can be rotated by any factor using a rotation matrix $R$ (Eq. 2.3).

$$v' = Rv \tag{2.3}$$

Where $v' = (x', y', z')$ are the new coordinates after rotation and $R$ [13]:

$$R = \begin{bmatrix} \cos(\theta_{x',x}) & \cos(\theta_{x',y}) & \cos(\theta_{x',z}) \\ \cos(\theta_{y',x}) & \cos(\theta_{y',y}) & \cos(\theta_{y',z}) \\ \cos(\theta_{z',x}) & \cos(\theta_{z',y}) & \cos(\theta_{z',z}) \end{bmatrix} \tag{2.4}$$

Euler's rotation theorem states that any rotation can be expressed as a single rotation of angle $\theta$ about an axis represented by the unit vector $u = (u_1, u_2, u_3)^T$ [13]. The rotation vector is then represented by Eq. 2.5.

$$r = \theta u \tag{2.5}$$

A quaternion, $q$, is an element of algebra which can be represented by Eq. 2.6.

$$q = q_1 + q_2 i + q_3 j + q_4 k = (q_1, q_2, q_3, q_4)^T \tag{2.6}$$

When a quaternion has a unit magnitude , it can represent the rotation of a point $p = (x, y, z)$. This point if first transformed into a quaternion representation in the form $p' = (0, x, y, z)^T$ [13]. The rotated point, $p_r$, is the result of Eq. 2.7.

$$p_r = -qp'\bar{q} \tag{2.7}$$

## 2.3 Related Work

In this Section, there will be an overview of not only the fastest and most accurate methods but also the most interesting approaches for object detection and pose estimation. All the approaches are based on CNNs, since, during research, they were found to be state-of-the-art in this field.

Understanding how each model works and how each method improved upon its predecessor with interesting changes is important for this thesis's scope and poses a significant learning experience. Therefore, the methods are presented to show the different approaches to problem-solving

over time. For each method, there is an explanation of the main ideas and an overview of the conclusions reached after evaluation. There is a comparison of results between methods in Sections 2.3.2.5 and 2.3.3.8.

### 2.3.1 Datasets

An essential aspect of training a ML model is the dataset used. A dataset is used for training and evaluation of a model, and the results obtained in each can serve as comparison points for competing architectures in the field.

The best datasets need to have variability in terms of the situations they present, in order not to introduce biases in learning. For detection, recognition, and pose estimation, this means having images of objects located in different places of the image and with different poses, in varying light conditions and situations such as occlusion or truncation, as well as having multiple objects per image to better match real-world situations.

The state-of-the-art datasets follow these rules and, as such, are used by nearly every new approach for both object detection/recognition and pose estimation. For detection and recognition, the main datasets used are the 2007 and 2012 Pascal Visual Object Classes (VOC) [14] and the Microsoft Common Objects in Context (COCO) [15] datasets. As for pose estimation, the main state-of-the-art datasets are LineMod [16], Occlusion LineMod [17] and YCB-Video [18].

In Pascal VOC, objects are presented in natural settings in a set of 11540 images, and there are 20 classes of objects with five different annotations. These annotations are the following:

- class: name of the class the object belongs to;
- Bounding box: dimensions and location of the bounding box of an object;
- Viewpoint: the position from which the image is taken;
- Truncation or occlusion: when the bounding box of the object does not correspond to the full object;
- Difficult: annotation specific to particularly challenging objects.

To evaluate the performance of a model three metrics are used: *IoU* (intersection-over-union), *AP* (average precision) per class and *mAP* (the mean of all APs). IoU is the ratio of the area of overlap between bounding boxes and the area of the union. AP is defined in Eq. 1 of [14], and it roughly measures the area under the precision-recall curve, where precision is defined by the number of true positives (TP) divided by the sum of TP with false positives (FP), and recall is the number of TP divided by the sum of TP with false negatives (FN). TP is the number of correct identifications of an object, FP the number of incorrect identifications of an object, and FN the number of objects incorrectly labeled as *no object*.

The MS COCO dataset, introduced in [15], looks to supply models with images of objects within their usual context. It has more categories (91 object types) and instances of objects

(328000+ images) than the VOC dataset. The evaluation metrics used are the same as the aforementioned dataset. However, there is a distinction between small, medium, and large objects. These datasets feature an added metric called *mean average recall*, which is the mean of the average recall for all classes.

LineMod features 15 video sequences with 15 3D objects without texture; each sequence has more than 1100 frames offering various viewpoints of the objects surrounded by clutter. In each image, pose estimation needs to be outputted for an individual object. In [17], an altered version of LineMod was introduced, offering more situations of occluded objects and 20 textured and texture-less objects. The YCB-Video dataset is optimized for robotic grasping training, offering 21 object types in a total of 92 RGB-D videos. The evaluation metrics used by all these datasets are known as ADD (average distance) and ADD-S. ADD was first introduced in Eq.1 and 2 of [16]. This metric computes the average distance between all correspondent points in the object 3D model and the proposed pose. As for ADD-S, it was introduced in [19] as a more stable metric when objects are symmetric.

### 2.3.2 Machine Learning in Object Detection and Recognition

#### 2.3.2.1 Faster R-CNN

Faster R-CNN (region-based CNN) [20] is an improvement on the Fast R-CNN model [21]. The main change occurs in the region proposal algorithm, where the selective search algorithm used in Fast R-CNN is replaced with a *Region Proposal Network* (RPN), bringing down calculation time for proposals from 2s to 10ms per image and also allowing the RPN to share layers with the following detection stages.

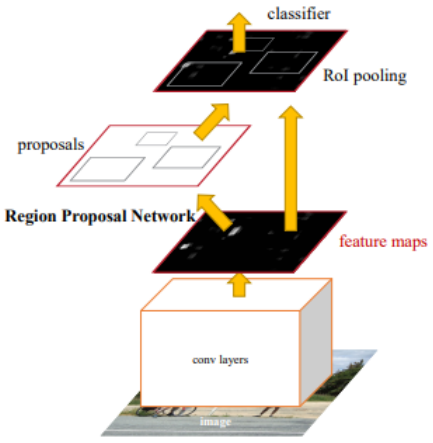



Figure 2.4: Illustration of the architecture of Faster R-CNN. [20]

The RPN's primary goal is to output a set of bounding-boxes around possible objects within the input image. This is done by applying a CNN (based on VGG) to the input image, which outputs the corresponding feature map. For every pixel in the feature map, a set of 9 *anchors*

(boxes with three different sizes and three different aspect ratios, whose center is the chosen pixel) is applied, and every anchor has two possible labels: *background* or *foreground*.

The network is divided into two final layers: a classification layer to label the anchor and a regression layer to refine the anchors' coordinates that overlap with the ground-truth bounding-boxes in the training dataset.

Training is done in *mini-batches* of 2 images, each with 64 ROIs (regions of interest). With this method, the computation cost is decreased, since ROIs from the same image share computation. Another method for increasing speed is dividing a FC layer with weight $W$ into two with weights $\Sigma V^T$ and $U$ (resulting from the singular value decomposition of $W$).

The feature map, with its corresponding ROIs, is then passed through a max pooling layer that acts on each ROI individually, resulting in a per-region feature vector. The vector is then fed into a CNN with two fully connected layers that branch into two output layers: one for class classification and another for bounding-box regression. The first layer outputs a probability for each class of object, and the second regresses to the correct sized bounding-box. [20][21]

In [20], Faster R-CNN was evaluated on the Pascal VOC and COCO datasets. Different evaluations are made using different parameters, such as training data, the region proposal method, and the backbone architecture. The following conclusions were made (Tables 3, 4 and 5 in [20]):

- The model runs faster when using VGG, as opposed to ZF-Net: 5 fps vs. 17 fps;
- Better mAP is obtained when using RPN instead of the previous region proposal method in [21]: 73.2% and 70.4% vs. 70% and 68.4%;
- Using COCO training data in conjunction with the VOC datasets leads to increased accuracy: 78.8% and 75.9% vs. 73.2% and 70.4%.

#### 2.3.2.2 R-FCN

Region-based algorithms are very accurate. However, they are costly in terms of computation since they work on every ROI separately. R-FCN looks to speed up the process by introducing a fully convolutional model (Fig. 2.5) that shares a big part of network computation on the entire image.

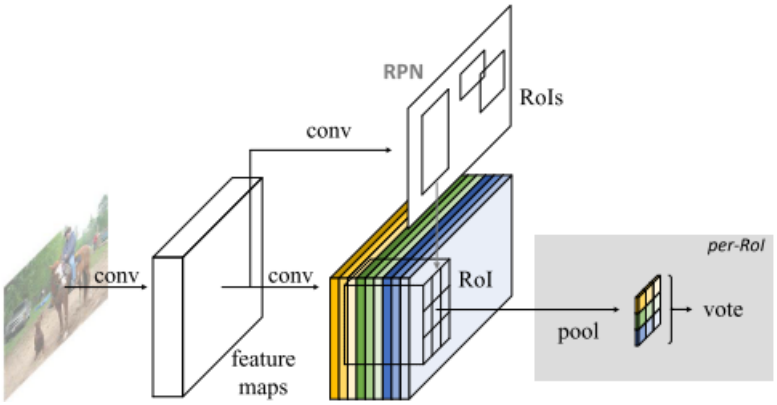



Figure 2.5: Illustration of the architecture of R-FCN. [22].

This model utilizes a two-stage approach consisting of a RPN (due to the accuracy it provides) and a region classification stage. The way of generating ROIs is the same as in 2.3.2.1; however, the backbone CNN architecture for feature extraction used is ResNet (instead of VGG), and the layers after obtaining the ROIs are removed. Passing the feature maps through another FC layer, 3x3 position-sensitive score maps (Fig. 2.6) are obtained, for each class of object.

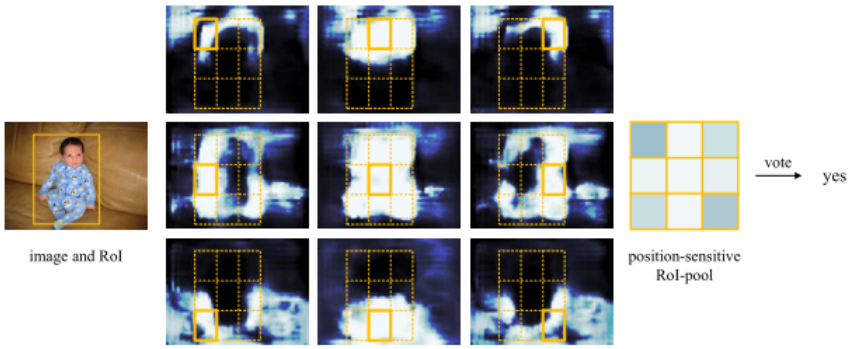



Figure 2.6: An example of a score map. [22].

The ROIs of the image are then divided into a 3x3 grid, and the overlap between grid spaces of the score maps and the ROI is computed (as exemplified in Fig. 2.6). The average between the votes for each grid cell is calculated, producing a vector that gives a probability of the object belonging to a particular class (calculations explained in Eq. 1 of [22]). For the regression of bounding-boxes, this model utilizes a similar approach to the one described in [21].

The model is evaluated on the Pascal VOC and MS COCO datasets. Different depths in the backbone architecture are used, as well as different region proposal algorithms. The main conclusions were (tables 4, 5 and 6 in [22]):

- The model has a lower testing time than Faster R-CNN: 2-3.5x faster;
- The accuracy (measured in mAP) is comparable to Faster R-CNN, except for smaller objects (on the COCO dataset) where it exceeds the competing model;

- Using both VOC and COCO training data leads to increased accuracy: 83.6% and 82% vs. 80.5% and 77.6%;
- Using the first 101 layers of ResNet results in the best accuracy: saturation occurs at 80.5%;
- The use of a RPN achieves superior results than other alternatives: 79.5% vs. 77.8% (for the closest alternative).

The architecture of the R-FCN removes the expensive computation that approaches such as Faster R-CNN use on each ROI (by passing each ROI through CNNs), in favor of a much more straightforward calculation of overlap, enabling the model to be faster but still maintain the accuracy offered by a region-based design.

#### 2.3.2.3 SSD: Single Shot Multibox Detector

SSD (Fig. 2.7) [23] is one of the single-shot approaches to object detection with the intent of being fast and accurate enough to be used in real-time applications. This detector removes the need for region proposals, while still maintaining accuracy.

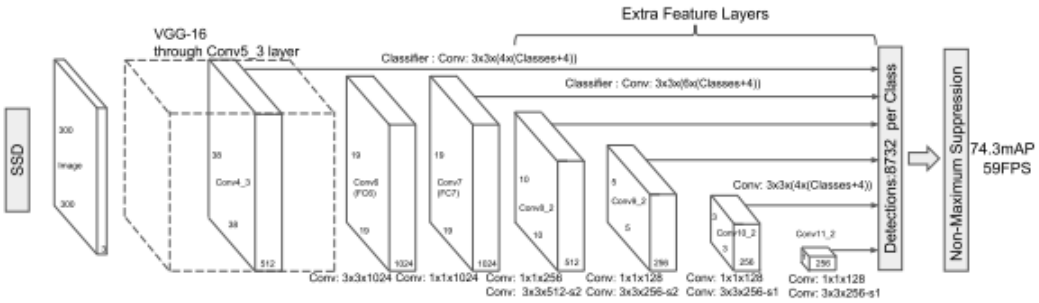



Figure 2.7: Overview of the SSD architecture. [23].

The first stage in SSD is the generation of feature maps at multiple scales, which allows for the detection of objects of different sizes - feature maps with higher resolution are able to detect smaller objects, while ones with lower resolution can detect bigger objects. The backbone architecture used is based on VGG, and other convolutional layers are appended at the end to allow for different scales.

In the second stage, detection is based on small convolutional kernels applied to each cell in the feature maps. For every cell, four default bounding-boxes of different aspect ratios are applied. These aspect ratios are chosen to accommodate objects of different shapes and sizes (Eq. 4 in [23]). For each box, a class probability score is computed, as well as different offsets for the size of the box (in order to better adjust the shape of the box to the shape of the object). This approach is similar to that of Faster R-CNN but applied to different resolutions of feature maps.

A particularly interesting aspect in [22] is the use of "data augmentation" for training. This means that, for each training image, either the entire image is used or a random patch is obtained

from it. In the case of a random patch, it can also suffer image distortions. The objective of data augmentation is to enable the model to handle various shapes and sizes of objects more robustly.

The model was evaluated on the Pascal VOC and MS COCO datasets, where the main conclusions were the following:

- As with previous models, training on both the VOC and COCO datasets increases accuracy (Tables 1, 4 and 5 in [22]);
- Using multiple outputs layers for multiple feature map resolutions results in higher accuracy (Table 3 in [22]);
- Data augmentation achieves better results, as expected Table 6 in [22]);
- SSD can achieve higher mAP than its competitors using a smaller resolution input image Table 7 in [22]).

#### 2.3.2.4 YOLO: You Only Look Once

YOLO was first introduced in [24], suffering subsequent alterations in [25] and some minor changes in [26]. It was introduced as a single-shot approach to object detection, by using a single network to solve a regression problem to predict bounding boxes and class probabilities for objects in an image.

The first version of YOLO divides its input image into a 7x7 grid, and each cell in the grid is responsible for identifying an object if its center is located within the cell. Each cell predicts a set of 2 bounding-boxes with associated confidence scores, as well as class probabilities. The confidence scores take into account the probability of the boxes having an object within them and also the accuracy of the box in terms of the shape of the object.

One problem that can arise from this approach is the same object being detected by more than one cell. In this case, the cell with the highest confidence for a specific object is chosen. The predictions of each cell are made by classifiers applied after feature extraction (which is done on the entire image by the network in Fig. 2.8).

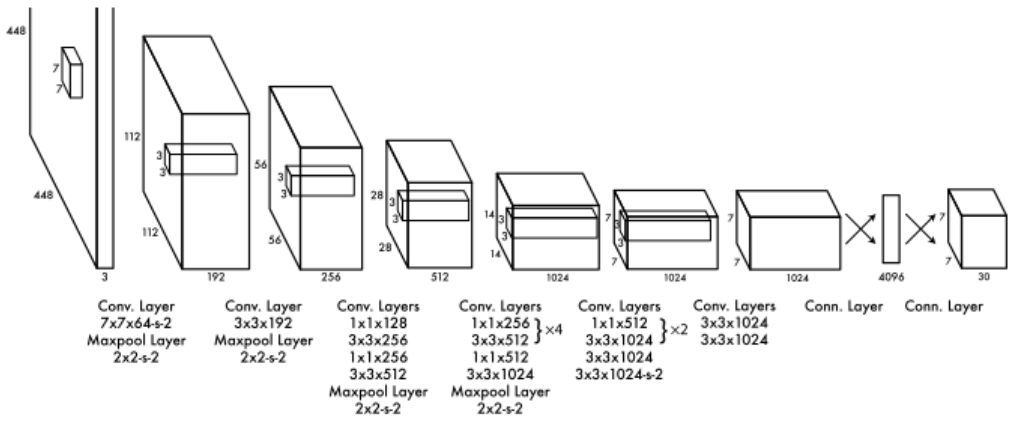



Figure 2.8: Overview of the original YOLO network. [24].

In the second version of YOLO [25], bounding-boxes were replaced by the same anchor boxes used in Faster R-CNN, since it made learning easier for the network and it also enabled each cell to predict more than one object. This had a small decrease in accuracy, but overall recall improved. The number of anchor boxes chosen for this approach was 5.

Another change was replacing the original backbone network (Fig. 2.8) with a custom network called Darknet-19, whose purpose was to reduce complexity and improve accuracy. A significant improvement in this version was that this model was able to detect more than 9000 object categories, compared to the original 20.

The third version [26] came with minor changes, the most important of which was the backbone CNN being expanded to Darknet-53, a CNN with 53 convolutional layers that is more powerful than its predecessor, but more effective than other used variants of ResNet. The most recent test results for YOLO were presented in [26]. They are the results of testing on the COCO dataset, and the following conclusions were made:

- When comparing mAP to the processing speed (Fig. 3 in [26]), YOLO achieves the best results;
- In terms of accuracy, YOLO is more accurate than SSD, but still inferior to two-stage approaches, such as Faster R-CNN.

#### 2.3.2.5 Comparison of Results

The following tables feature the results for mean average precision (in percentage) for each of the methods in each dataset. The values are taken from the papers that introduce each method. When a paper does not present results for a particular dataset, the value is omitted.

Tables 2.1 and 2.2 present the results for the VOC07 and VOC12 datasets, respectively.

Table 2.1: Comparison between methods on the Pascal VOC07 dataset

| Author/Reference | Year | Method | mAP (%) |
|---|---|---|---|
| Ren et al. [20] | 2017 | Faster R-CNN | 78.8 |
| Dai et al. [22] | 2016 | R-FCN | 83.6 |
| Liu et al. [23] | 2016 | SSD | 81.6 |
| Redmon and Farhadi [26] | 2018 | YOLO | – |

Table 2.2: Comparison between methods on the Pascal VOC12 dataset

| Author/Reference | Year | Method | mAP (%) |
|---|---|---|---|
| Ren et al. [20] | 2017 | Faster R-CNN | 75.9 |
| Dai et al. [22] | 2016 | R-FCN | 82.0 |
| Liu et al. [23] | 2016 | SSD | 80.0 |
| Redmon and Farhadi [26] | 2018 | YOLO | – |

Table 2.3 presents the results for the MS COCO dataset. On this dataset, mAP can be divided into two metrics. The first is the standard mAP when IoU between the object bounding-box and the ground-truth is 0.5. The second, is a more strict metric, for the mean average precision over various thresholds, for an IoU between 0.5 and 0.95.

Table 2.3: Comparison between methods on the MS COCO dataset

| Author/Reference | Year | Method | mAP@.5 (%) | mAP@[.5,.95] (%) |
|---|---|---|---|---|
| Ren et al. [20] | 2017 | Faster R-CNN | 42.7 | 21.9 |
| Dai et al. [22] | 2016 | R-FCN | 53.2 | 31.5 |
| Liu et al. [23] | 2016 | SSD | 46.5 | 26.8 |
| Redmon and Farhadi [26] | 2018 | YOLO | 57.9 | 33.0 |

As can be seen, R-FCN and SSD are the most accurate methods on the VOC datasets. SSD is the fastest of the two, being a one-shot approach. On the MS COCO dataset, YOLO achieves the best results. When IoU is between 0.5 and 0.95, there is a significant drop in accuracy, since it is difficult to obtain a high IoU.

### 2.3.3 Machine Learning in Pose Estimation

#### 2.3.3.1 SSD-6D

The objective of this network (Fig. 2.9), introduced in [27], is to build upon the structure of SSD to obtain a collection of outputs: class probabilities, coordinates of a 2D bounding-box, scores for possible viewpoints and in-plane rotations.

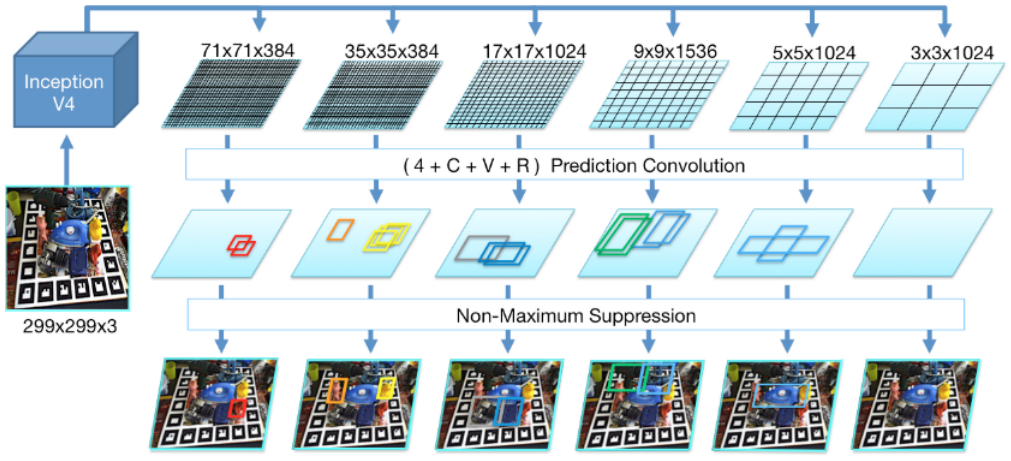



Figure 2.9: Overview of the SSD-6D model. [27].

The input image is passed through a backbone CNN based on InceptionV4 to output a set of feature maps at different scales. The same structure as SSD is used for prediction of class and

regression of the bounding-box. The main changes occur in the scoring for possible viewpoints and in-plane rotations.

As is shown in Fig. 2.9, each map is convolved with a kernel of shape (4 + C + V + R), where C represents the number of object classes, V the number of viewpoints and R the number of in-plane rotations. The viewpoint is the position in 3D space from which the object is viewed, which influences the aspect of the object as seen from the camera, while in-plane rotation is seen as a transformation of the same viewpoint.

As can be seen in Fig. 2.10, the viewpoints are sampled from a half-sphere around the object. For symmetrical objects, only the arc in green is sampled, and, for semi-symmetrical objects, only the points in red are used. In [27], it is argued that convolutional layers are more effective at scoring a viewpoint and in-plane rotation than using regression to predict a set of translations and rotations. Furthermore, by scoring viewpoints with a confidence value, all that is above a certain threshold can be accepted, thereby dealing well with symmetrical objects.

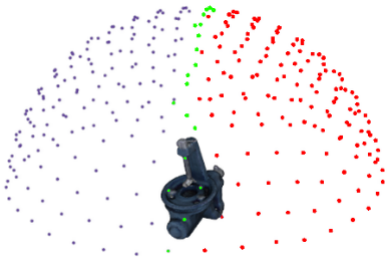


Figure 2.10: Illustration of every viewpoint considered. [27].

Knowing the parameters of the camera, the most confident scores can be pooled to calculate the translation and the rotation of the object, resulting in a set of 6-DOF pose hypotheses. Each hypothesis goes through a pose refinement step, which uses the Iterative Closest Point (ICP) algorithm (2.3.3.7). The best pose is chosen by comparing the refined poses to the depth data from RGB-D.

SSD-6D was evaluated on both the Linemod and the Tejani datasets. However, since none of the other methods are evaluated on the last dataset, there are no comparisons to be made. Tejani dataset results will, therefore, be omitted.

#### 2.3.3.2 PoseCNN

PoseCNN [19] revolutionized pose estimation algorithms by decoupling pose estimation into three separate tasks: semantic labeling, 3D translation estimation and 3D rotation regression (as can be seen in Fig. 2.11). This approach facilitates the job of the network by enabling it to model how each task relates to others. The backbone CNN takes an input image and outputs feature maps of different scales. The branches corresponding to each task then use these feature maps for calculations.

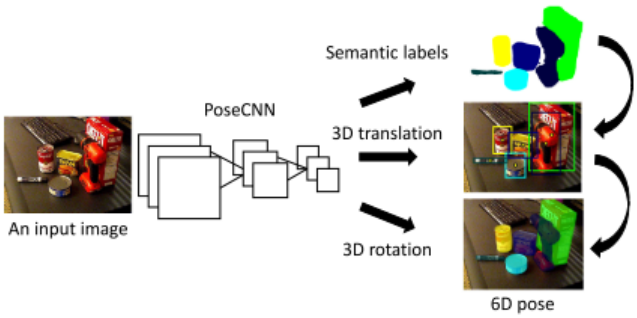



Figure 2.11: Overview of the PoseCNN system. [19].

In the semantic labeling task, two 512 channel maps are processed in order to obtain one of the same scale as the original image. Here, object detection is done by applying convolutional kernels to each pixel in the image and attributing a semantic label: each pixel is classified into an object class. This gives better results than the bounding-box approaches used in SSD-6D, for example, since it handles occlusions better.

For 3D translation estimation, the network needs to output a vector of coordinates $(T_x, T_y, T_z)$ for the object center (in the camera coordinate system). To that extent, each pixel that belongs to an object needs to vote for the center of the object in the image coordinate system, $(c_x, c_y)$. After that, $T_x$ and $T_y$ can be derived using Eq. 1 in [19]. Voting is implemented by a Hough voting layer in the network and is done as follows:

1. A regression network predicts an array $(n_x, n_y, T_z)$ for each pixel;
2. Each pixel votes for other pixels (on whether they are an object center) along the direction of the vector defined by $(n_x, n_y)$;
3. The pixel with the most votes is chosen as the object center.

The mean of all values of $T_z$ for each pixel in an object is chosen as the true value of $T_z$. Additionally, all pixels inside an object are known as inliers. The bounding-box that contains all inliers is also generated in this step and is used for the 3D rotation task.

In this task, the objective is to output a quaternion, which represents the estimated rotation. The first step is to apply 2 ROI pooling layers using the bounding-boxes generated in the previous task. In that way, the feature maps for each ROI can be obtained. These feature maps are then added and fed into 3 FC layers, the last of which outputs a quaternion.

PoseCNN is evaluated on the YCB-Video and the Occlusion LineMod datasets. The following conclusions can be made from tables 2 and 3 in [19]:

- This model outperforms coordinate regression algorithms;
- Results are far superior when using pose refinement;
- On the Occlusion LineMod dataset, PoseCNN with a pose refinement algorithm (ICP) achieves higher accuracy than other methods;

#### 2.3.3.3 DPOD: Dense Pose Object Detector

DPOD [28] takes a different approach to others mentioned previously. Its model (Fig. 2.12) is comprised of three blocks: correspondence block, pose block, and a final pose refinement block. The correspondence block takes as input the desired image and uses an encoder (based on the first 12 layers of ResNet) and three decoders that output a correspondence map (first two decoders) and the ID masks for each object (third decoder).

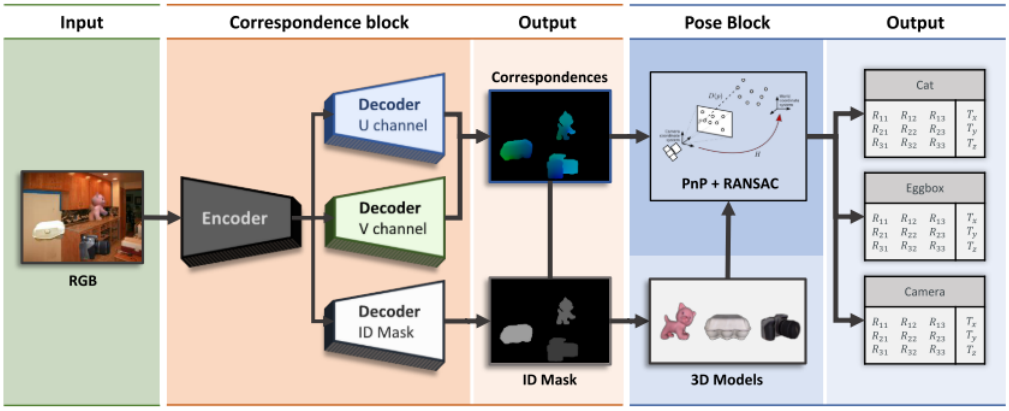



Figure 2.12: Overview of the DPOD system. [28].

A correspondence map is a 2-channel image with values from 0-255 that maps a pixel in the image to a vertex in the object 3D model. This results in more straightforward training of the network and an overall increase in quality since it just needs to solve a color classification problem to match the image correspondence map to the 3D model correspondence map, instead of having to regress the coordinates of the object.

The ID masks are a result of the probability that an object pixel belongs to a specific class (such as previous approaches have used). Using the ID masks, the 3D model for each class can be obtained. Both the correspondence map and the 3D models are fed into the pose block, which uses PnP+RANSAC to output a rotation matrix $R$ and a translation vector $T$.

As opposed to other forms of pose refinement (2.3.3.7), the DPOD model utilizes a pose refinement block (Fig. 2.13) based on the ResNet architecture. This block takes in a patch of the original image containing the object and a 3D rendering of the object in the estimated pose. These inputs are fed separately through two branches composed of the first five layers of ResNet (E11 and E12), and the outputs are subtracted and fed into another ResNet-like network (E2).

The difference feature vector that is outputted is used to compute the error between the predicted pose and the actual pose. This is done by feeding it into three separate regression layers (XY head, Z head and R head) that correct the coordinates for translation and the rotation matrix (as is seen in Fig. 2.13).

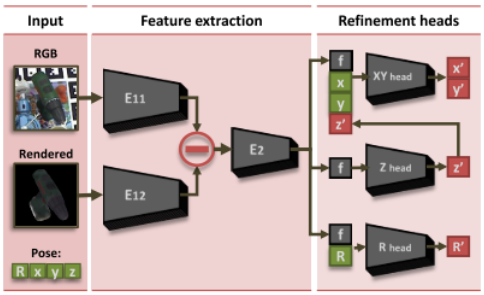



Figure 2.13: Illustration of the pose refinement block. [28].

The model was evaluated on the LineMod and the Occlusion LineMod datasets, having obtained the following conclusions:

- Comparatively, DPOD achieves state-of-the-art results, with only PVNet (explained in Section 2.3.3.4) having better accuracy in pose estimation (Table 1 in [28]);
- Pose refinement improves accuracy by almost 10% (Table 1 in [28]);
- DPOD's novel pose refinement technique achieves better results than DeepIM (Tables 1 and 5 in [28]).

#### 2.3.3.4 PVNet: Pixel-wise Voting Network

PVNet looks to improve upon the two-stage approach of keypoint detection and pose estimation by using a model more robust to occlusion and truncation. The main idea is to predict, for each pixel in an object, the unit vectors from that pixel to all the keypoints in the object - similar to what was done in PoseCNN for the localization of an object center. This approach is more robust to situations where the keypoints are not visible in the image since its position can be inferred from the other visible pixels.

In the first stage of PVNet, a backbone CNN based on ResNet is used to obtain class probabilities for each pixel (semantic segmentation) and the unit vectors that represent the direction from that pixel to every keypoint. The voting of each pixel is characterized in Eq. 1 and 2 of [29]. To reduce variance in localization, the model needs to choose the keypoints located on the surface of the object. The following algorithm (Farthest Point Sampling) is used to choose a set of 8 keypoints:

1. Add the center of the object to the set of keypoints;
2. Choose the farthest keypoint from the set and add it to the set;
3. Repeat the second step until eight keypoints are chosen.

Finally, the model needs to compute the 6-DOF pose using the set of keypoints. This is done using an altered version of the PnP algorithm that takes into account the mean and covariance (uncertainty) associated with each keypoint (which are calculated with Eq. 3 and 4 of [29]).

This method is evaluated on the LineMod, Occlusion LineMod, and YCB-Video datasets. It is concluded that PVNet achieves the best results in terms of the ADD(-S) metric when compared to other approaches (Table 3, 5, and 7 in [29]).

#### 2.3.3.5 DenseFusion

Most approaches so far have used 2D features from RGB images and used depth data only for pose refinement. DenseFusion's [30] model (Fig. 2.14) takes advantage of depth information from the first stage by fusing RGB data and point cloud values on a per-pixel basis. Since both RGB and point cloud data are very different data types, this model treats each one separately and uses a pixel-wise dense fusion algorithm to combine them.

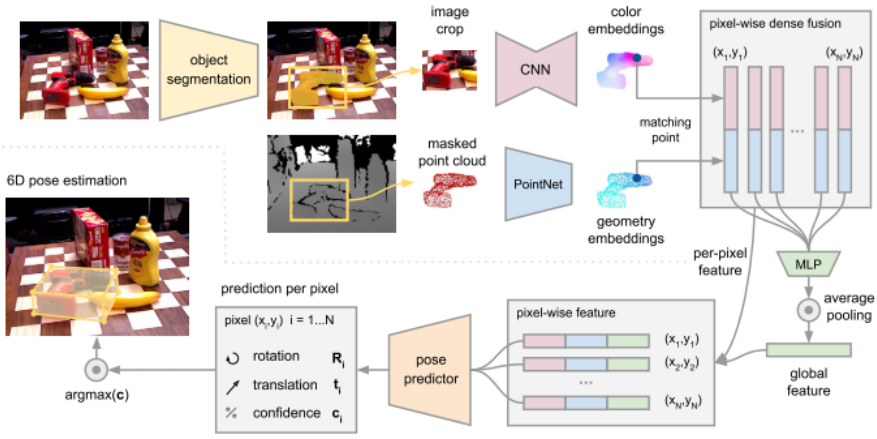



Figure 2.14: Overview of the DenseFusion architecture. [30].

In the first stage, the same semantic segmentation algorithm applied in PoseCNN is used to obtain segmentation masks for each object. A bounding-box that includes each mask is calculated and is used to obtain an image crop in its location. The depth data of pixels inside the bounding box is also used in order to convert it into a 3D point cloud representation.

The cropped image is fed into an autoencoder network (based on ResNet) that outputs color information. On the other hand, point cloud data is processed by a PointNet-like structure to obtain the geometric information of the object.

The fusion algorithm used combines color (color embeddings) and geometric information (geometry embeddings) into feature vectors for each pixel in the object. Every feature vector is then processed by a MLP with average pooling to obtain a global feature vector, which is then appended to each per-pixel vector, creating a pixel-wise feature (as shown in Fig. 2.14). The pixel-wise feature is fed into the pose predictor network which outputs an estimation of the pose for every pixel with an associated confidence value: $R_i$, $t_i$, and $c_i$.

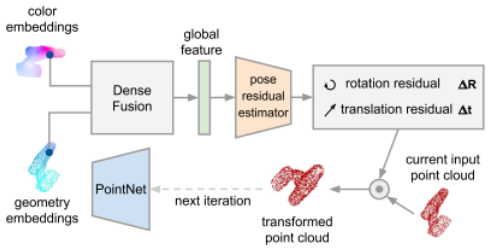



Figure 2.15: Illustration of the pose refinement network. [30].

DenseFusion also presents an iterative approach to pose refinement (Fig. 2.15) utilizing a correction network that is composed of 4 FC layers. However, this network needs to learn to correct the pose instead of predicting a new one. Therefore, its inputs are the color embeddings of the image and the geometry embeddings of a transformed point cloud.

This transformed point cloud is the result of altering the input point cloud by the factors $\Delta R$ and $\Delta t$ predicted by the pose residual estimator. This can be done iteratively, refining the estimation further and further in each step. The final pose predicted is obtained by concatenating every pose obtained in each iteration.

Evaluation of the model is done on the LineMod and YCB-Video datasets. The main conclusions obtained in [30] are the following:

1. With refinement, DenseFusion achieves the highest accuracy when compared to the second best method, PoseCNN+DeepIM (tables 1 and 2 in [30]);

2. Pose refinement increases pose accuracy by 4-8% (tables 1 and 2 in [30]);

3. In Table 3 of [30], it can be seen that DenseFusion is close to 200x faster than PoseCNN plus ICP;

A particularly relevant aspect (for this thesis) in [30] is the use of DenseFusion in a robotic grasping situation. Out of 60 attempts, the robot has a 73% success rate while using DenseFusion as a pose predictor.

#### 2.3.3.6 HybridPose

Most approaches have only used only one type of intermediate representation for an object, the most popular of which is a keypoint representation. In [31], the authors argue that, in a real-world setting, it is difficult to predict keypoints accurately based on a RGB image. HybridPose (Fig. 2.16) looks to solve this problem by adding more intermediate representations, enabling its model to work better in occlusion situations and having more information about the geometrical characteristics of the object.

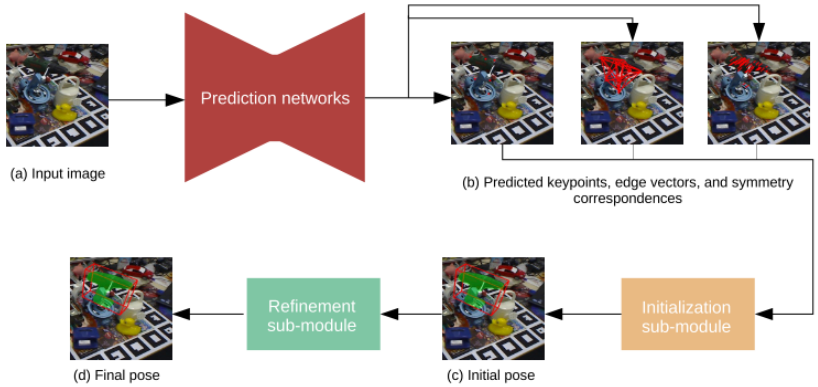



Figure 2.16: Illustration of the HybridPose model. [31].

The intermediate representations used are: keypoints, edge vectors, and dense pixel-wise symmetry correspondences. To obtain keypoints, the same prediction network as in PVNet is used. As for edge vectors, the set of keypoints is treated as a graph, and the edge vectors are the vectors connecting a pair of keypoints in the graph. The third representation establishes connections between each pixel and its symmetrical counterpart.

All the sub-networks for prediction of the representations are based on ResNet and share parameters between all layers, except for the last one, helping to keep training time low. The initialization sub-module in Fig. 2.16 takes all three representations as inputs and utilizes a modified EPnP structure to estimate the pose of the object. This initial prediction is optimized by the refinement sub-module, outputting a final pose prediction.

This model was evaluated on the LineMod and Occlusion LineMod datasets. The conclusions made were:

- Even though HybridPose is outperformed by DPOD on the Linemod dataset (Table 1 in [31]), it is the best performer on Occlusion Linemod by a large margin (Table 2 in [31]);
- The full model with every intermediate representation achieves less error in rotation and translation, as opposed to using only keypoint or keypoint and symmetry representations (Table 3 in [31]);

#### 2.3.3.7 Pose Refinement Algorithms

The use of pose refinement algorithms can improve the accuracy of an initial pose estimation. These algorithms can take the ground-truth pose and correct the estimated pose, achieving better results. One of the main algorithms employed is ICP (Iterative Closest Point); however, a new method for pose refinement has emerged, named DeepIM (Deep Iterative Matching).

The fundamental objective of ICP is to align a set of input points to a reference set, given some error metric. This algorithm has many variants that improve accuracy and speed; however, the most general steps applied to 3D models will be introduced, based on [32]:

1. Match a point in the input model to its closest point in the reference model;
2. Given a pair of input and reference points, estimate the correct rotation and translation to align both points. This is done by minimizing some distance metric between points;
3. Apply the transformation to the input points;
4. Repeat all steps above until the distance error between both models is at a defined minimum;

A more recent approach to pose refinement is presented in [33]. DeepIM takes as inputs an estimated pose, the object 3D model, and the image of the object (named the *observed image*) and applies an iterative method based on an ANN (as can be seen in Fig. 2.17).

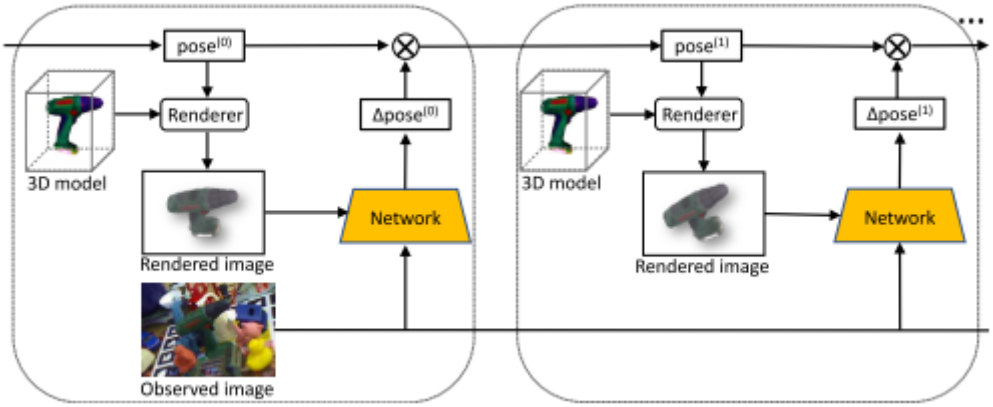



Figure 2.17: Overview of the DeepIM model. Image taken from [33]

The following steps are applied iteratively:

1. Use the 3D object model to create an image rendering in the estimated pose;
2. Apply a zoomed-in crop on the observed and rendered images to enlarge the object and give it greater detail;
3. Create a mask of the object for each of the previously mentioned images;
4. Input both images and their corresponding masks to a CNN based on FlowNet and followed by four additional FC layers;
5. The network outputs a quaternion for rotation correction and a translation correction of the previously estimated pose;
6. Use the corrected pose as the new input and repeat all steps until either an ideal error or a defined number of iterations are achieved.

DeepIM was evaluated on the Linemod, Occlusion Linemod, and YCB-Video datasets, using PoseCNN as the initial pose estimator. As can be seen in 2.3.3.8, DeepIM improves the initial pose estimation accuracy (of PoseCNN without refinement) by close to 20% on most datasets.

#### 2.3.3.8 Comparison of Results

Table 2.4 has the joint results for every pose estimation method on all datasets.

Table 2.4: Comparison between methods on the Linemod, Occlusion Linemod and YCB-Video datasets

| Author/Reference | Year | Method | Datasets | ADD-S (%) |
|---|---|---|---|---|
| Kehl et al. [27] | 2017 | SSD-6D | Linemod<br>Occlusion Linemod<br>YCB-Video | 90.9<br>–<br>– |
| Xiang et al. [19] | 2018 | PoseCNN | Linemod<br>Occlusion Linemod<br>YCB-Video | –<br>24.9<br>75.9 |
| | | PoseCNN+ICP | Linemod<br>Occlusion Linemod<br>YCB-Video | –<br>78.0<br>93.0 |
| Li et al. [33] | 2019 | PoseCNN+DeepIM | Linemod<br>Occlusion Linemod<br>YCB-Video | 97.5<br>55.5<br>94.0 |
| Zakharov et al. [28] | 2019 | DPOD | Linemod<br>Occlusion Linemod<br>YCB-Video | 95.2<br>47.3<br>– |
| Peng et al. [29] | 2020 | PVNet | Linemod<br>Occlusion Linemod<br>YCB-Video | 86.3<br>40.8<br>73.4 |
| Wang et al. [30] | 2020 | DenseFusion | Linemod<br>Occlusion Linemod<br>YCB-Video | 94.3<br>–<br>93.1 |
| Song et al. [31] | 2020 | HybridPose | Linemod<br>Occlusion Linemod<br>YCB-Video | 94.5<br>79.2<br>– |

As can be inferred, pose refinement is a crucial step to improve a model's accuracy: PoseCNN plus DeepIM, DenseFusion, and HybridPose achieve the highest accuracy by using pose refinement. In terms of occlusion, HybridPose is the best performer, due to its use of multiple intermediate representations.

# Chapter 3

# 3D Object Recognition and Localization System

In this chapter, first, an overview of the complete system for Automatic Object Recognition and Localization for robotic grasping is given, as well as a look at its data flow. A more in-depth description is provided for the implementation of this system, giving detail not only on each machine learning architecture but also on each module and the system's operating modes as a State Machine.

## 3.1 System Overview

For the purpose of creating a system able to detect and localize objects for automatic grasping, several different modules were joined. The complete general architecture of the system is presented in Fig. 3.1, where the data flow can also be seen.

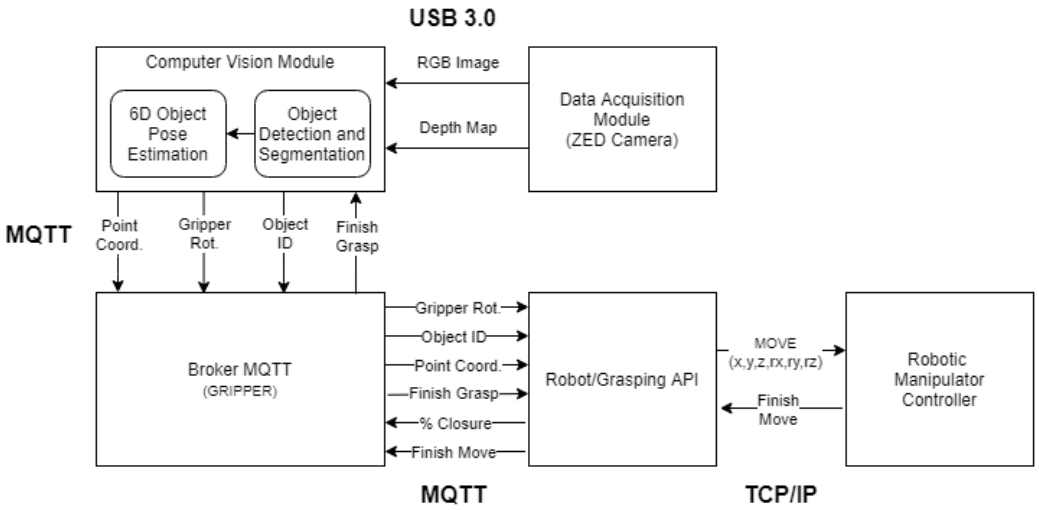



Figure 3.1: Illustration of the data flow in the system

First, for capturing the grasping workspace, a Data Acquisition module using a stereo-vision camera was implemented. This camera (ZED camera) is responsible for providing visual data to the computer program, through a USB 3.0 connection. This data is captured in the form of RGB images, as well as in a depth map. After visual data acquisition, the Computer Vision module (Object Detection/Segmentation plus Pose Estimation) processes the received information to obtain the object's 6D coordinates (three for translation (*Point Coord.*), and three for rotation (*Gripper Rot.*)), as well as identification (*Object ID*).

The Computer Vision Module was implemented in a Python script, that contains a MQTT (MQ Telemetry Transport) client. This client is connected to a MQTT broker located inside the gripper controller (a Raspberry Pi). In order to send the information for the robot, the 6D pose and object's ID are published as messages in two different MQTT topics: *"topic/pose"* and *"topic/id"*.

In the robotic system's side, a Robot/Grasping API was used, also implemented in Python with a MQTT Client. In this API, the client subscribes to the *pose* and *id* topics, receiving from the broker every published message in those topics. With the object's pose, the API is then able to send a direct *"MOVE"* command with those parameters to the robotic manipulator controller, via TCP/IP. The robot controller then sends back a signal when arriving at the defined position. After receiving this feedback, the system knows the grasping itself can be performed.

To achieve this, the Robot API generates a closing percentage for the gripper (*% Closure*), ranging from 0 (totally open) to 100 (totally closed), based on the object's ID. This percentage is then published in a MQTT topic ("*topic/gripper*"), triggering the gripper to close or open accordingly, and grasp the object in question. After finishing grasping, the gripper publishes a message to provide the API with that info, which then can send a new *"MOVE"* command for the object's goal position after grasping. Finally, the controller sends the finished movement signal, the API publishes a message for the gripper to open, and the gripper publishes a message of finishing the ungrasping to the vision system, concluding the process.

### 3.1.1 Operation Modes

As can be seen in Fig. 3.2, the complete system can be run in two different modes: *Configuration* mode and *Normal* mode. In the first mode, the objective is to create three configuration files that will be used by the system in normal operation. The second mode is responsible for the interaction with the system and the calculations for pose estimation.

#### 3.1.1.1 Configuration Mode

The *Configuration* mode has three steps and is intended to be used by a technician. Its purpose is to ease the creation of configuration files that are required and loaded into *Normal* mode. Each step can be skipped in case of an already existing configuration file in the program directory. These files are more practical and increase the efficiency and the memory optimization of the program. The purpose of each file is explained in Section 3.1.1.2.

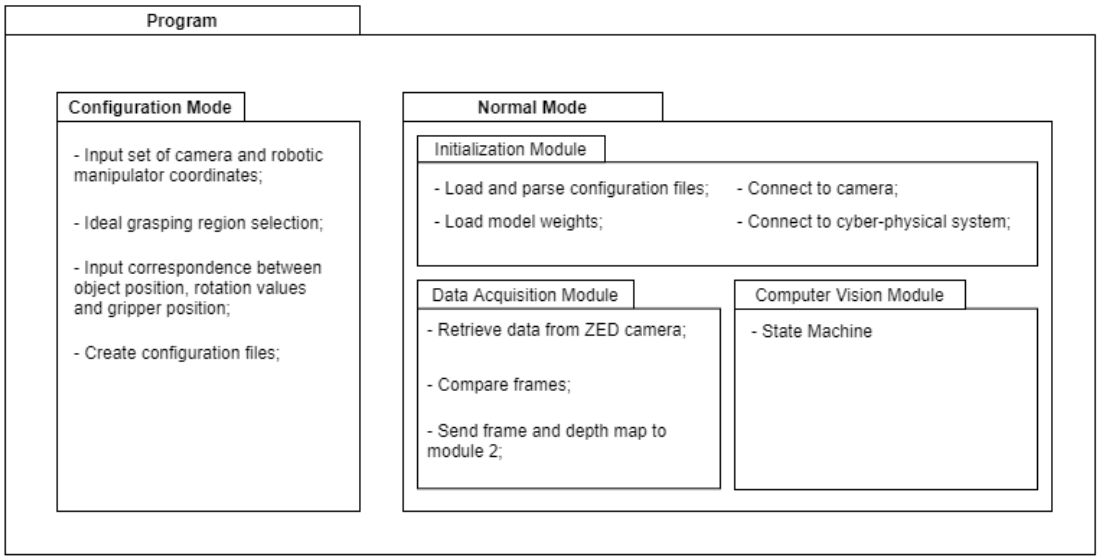



Figure 3.2: Overview of all operation modes

The first step in this mode is to input a set of matching points coordinates, both in the camera coordinate frame and in the robotic manipulator coordinate frame (the base joint, usually). These coordinates are then saved to the first configuration file. In the second step, the user is shown an interactive window for each object 3D model and is given the opportunity of choosing the region of points that are ideal for grasping. If the region selected is too small, the user is asked to select a bigger region, while, if it is too big, the set of points is reduced automatically to fit the intended size. This selection is important, as it gives the user the possibility of adapting the optimal grasping position according to the specific task being performed. The selected points are then saved to the second configuration file.

Finally, the third step asks the user to input the range of rotation values that corresponds to a particular position of the object (such as a rotation of 90 degrees around the x-axis corresponding to the *laying-down* position). Afterward, the user can also input a set of gripper rotations that give the optimal grasping for a specific object position. Once again, this selection allows the user to adapt the grasping to his own preference and application. These values are saved into the third configuration file.

#### 3.1.1.2 Normal Mode

This mode can be seen as the main operational mode for the program. It is divided into three different units. The first unit (Initialization Module) runs only on start-up and loads the various configuration files in the program directory, parses them, and creates program variables. It also loads the detection model weights into the CPU and the pose estimation model weights into the GPU (this is further explained in Section 4.2.3). Furthermore, a connection is established with the ZED camera and with the MQTT broker. The other two modules, however, run in parallel threads, for the duration of the program.

The Data Acquisition Module has two main functions. The first function deals with data retrieval from the ZED camera; it runs on an infinite loop, continuously retrieving the RGB image frame and the depth map. The other function calculates differences between consecutive frames, using the Structural Similarity (SSIM) algorithm. This algorithm outputs a value, from 0 to 1, that indicates how similar two images are, and the function compares this value to a threshold value (calculations of this value will be shown in Section 4.3.1), in order to indicate if there were changes between frames (such as an object moving). In case of changes, the function raises a global flag (indicating that a new frame is available) and passes the new frame and its corresponding depth map to the Computer Vision Module.

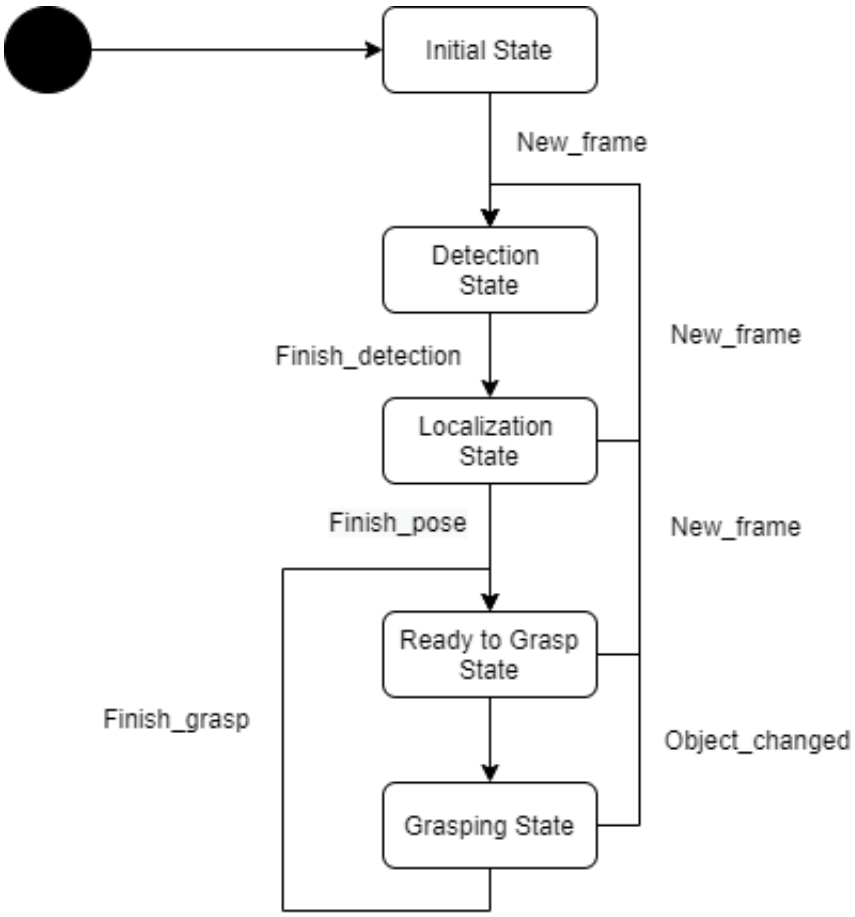



Figure 3.3: State machine of the pose estimation module

In the Computer Vision Module, a state machine (Fig. 3.3) is implemented. There is one *Initial* state and four main states:

- *Detection*: implements the Object Detection and Segmentation system explained in 3.2.2.1;
- *Localization*: implements the 6D Object Pose Estimation system explained in 3.2.2.2;
- *Ready to Grasp*: chooses which object to grasp and sends coordinates to robotic manipulator;

- *Grasping*: waits for grasping to finish and tries to detect changes in frame.

The transition between states depends on the changes between frames that are detected in module 1. If any object in the frame moves or disappears, the whole state machine has to reset to the *Detection* state.

## 3.2 System's Modules

### 3.2.1 Data Acquisition Module

In the Data Acquisition Module, the goal is to acquire usable data from the grasping workspace for the Computer Vision module to process and extract information needed for grasping tasks. In this case, the information wanted is 6D pose estimation and object classification. Given this, and regarding the requirement for a 3D based solution in our approach, a depth/3D camera was needed. In our solution, given several reported uses and availability, the ZED stereovision camera and its SKD were used.

The ZED camera was developed by Stereolabs and is composed of two side-by-side RGB cameras, which output synchronized RGB images. The two wide-angle lenses have 110° field-of-view, spaced at a baseline of 120 mm, allowing accurate depth estimation in the range of 0.7 to 20 meters. Camera picture, along with product dimensions, are shown in Fig. 3.4. The most relevant characteristics of the ZED camera are summarized in Table 3.1.


Figure 3.4: ZED Stereo Camera and dimensions illustration

For depth acquisition, the ZED camera imitates human vision, where each eye has a slightly different view of the world around, and by comparing the two views, depth can be inferred. Likewise, this camera, with its separate lens, can estimate depth by comparing the pixels' displacement between the left and right RGB images. The camera then provides a depth map (image containing information regarding the distance of the surfaces of scene objects ($z$), for every pixel $i,j$) in the camera coordinate system. This depth is expressed in metric units and calculated from the left camera's eye back to the grasping object.

Table 3.1: ZED Camera relevant features [1]

| | |
|---|---|
| **Size and Weight** | Dimensions:<br>175x30x33 mm<br>Weight: 159 g |
| **Individual image and depth resolution** | HD2K: 2208 x 1242 (15 FPS)<br>HD1080: 1920 x 1080 (30, 15 FPS)<br>HD720: 1280 x 720 (60, 30, 15 FPS)<br>WVGA: 672 x 376 (100, 60, 30, 15 FPS) |
| **Depth** | Range: 1-20 m<br>Format: 32 bits<br>Baseline: 120 mm |
| **Lens** | Field of View: 110°<br>f/2.0 aperture |
| **Connectivity** | USB 3.0 (5 V / 380 mA)<br>0°C to +45°C |
| **SDK System Requirements** | Windows or Linux<br>Dual-core 2.3 GHz<br>4 GB RAM<br>Nvidia GPU |

The camera also provides a SDK for facilitating camera control, both in C++ and Python. For our implementation, only the Python version was used. The ZED SDK is designed around OpenCV and CUDA libraries, with its calibration and depth estimation routines exploiting CUDA's parallel GPU computing capabilities. The SDK also offers optional depth map processing, including occlusion filling and edge sharpening. Unfortunately, these built-in post-processing techniques come with high computational costs, so they could not be used in our solution since real-time is a crucial requirement.

The camera is configured to run at 15fps, with 720p resolution. The minimum distance for depth estimation is 0.3m, and the depth estimation was set to*quality mode*, meaning it outputs more accurate depth values at a small computational cost.

### 3.2.2 Computer Vision Module

As shown in Fig. 3.1, the Computer Vision Module is the most important in the proposed approach, since it is through its processing capabilities that the object's poses and categories are identified for performing grasping operations. This module is composed of two different but interconnected models: the Object Detection and Segmentation model, and the Object Pose estimation model. In the following Subsections, details on their implementation will be provided.

#### 3.2.2.1 Detection and Segmentation Model

The object detection and segmentation model is composed of a pre-processing stage and an inference stage. In pre-processing, the data received from the Data Acquisition module is processed, so it can be fed into the inference stage, which is made up of a Mask-RCNN model [34]. This stage

then outputs a set of object identifications, bounding-boxes, segmentation masks, and confidence values for each object in the camera frame.

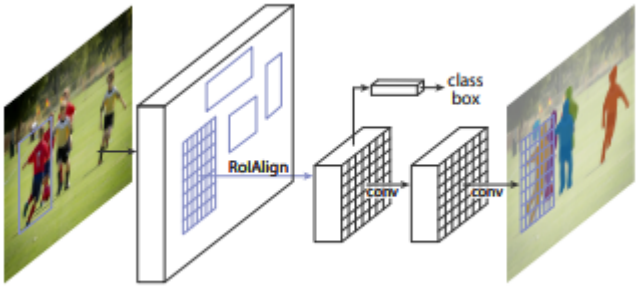



Figure 3.5: Overview of the Mask-RCNN model [34]

In the pre-processing stage, three RGB image frames and corresponding depth maps are received by the pipeline, and an average of the pixel values for both the RGB frames and the depth maps are calculated. This is to help reduce the impact of small illumination differences, pixel shifting, and random noise when a frame is acquired, which in turn reduces the flickering effect in the object masks. This is discussed at greater length in Section 4.3.2. Afterward, the image frame is resized to fit the input size of the inference stage.

The Mask-RCNN model (overviewed in Fig. ) is an extension of the Faster R-CNN model 2.3.2.1, by applying a small FCN each ROI and outputting a segmentation mask. As can be seen in the state-of-the-art analysis in section 2.3.2.5, Faster R-CNN is not the top-performing model when it comes to pure object detection, so why is it the one used? The answer is that pure object detection is not the only factor in this project. The final output should be the 6D pose of the object, which means that an additional model (in this case, the 6D pose estimation model discussed further on, in Section 3.2.2.2) is needed. One of the model requirements is the input of a segmentation mask for each object and YOLO - the top-performing model (in object detection), both in terms of accuracy and speed - does not output segmentation masks.

Two different alternatives were analyzed: using YOLO with a segmentation mask network overhead (much like how Mask-RCNN improves upon Faster R-CNN) or using Mask-RCNN instead. The conclusions were that Mask R-CNN is actually more accurate and marginally faster than the other alternative; therefore, it is actually a better choice for this implementation.

#### 3.2.2.2 Pose Estimation Model

This system, as opposed to object detection and segmentation, works on objects one by one. It begins with a pre-processing stage, which processes the data received from the object detection model to optimize the inputs of the pose estimation stage. In pose estimation, a Densefusion 2.3.3.5 architecture is used for inference, and it outputs a set of rotation quaternions and translation vectors for each object. A pose refinement stage is then applied to give a better pose estimation.

Finally, a post-processing stage is executed to calculate the final optimal grasping point coordinates and gripper pose for every object in the frame.

Due to the ZED camera limitations, the depth map might return values such as infinite and *NaN*. This occurs when points are too close/far from the camera (in the case of infinite values), or when depth cannot be estimated due to occlusions (in the case of *NaN*). To make sure these values do not contribute to poor calculations, a mask of the depth map is created where only real values are masked.

After the depth map mask is obtained, it is multiplied by the segmentation mask to obtain all the real depth values of only the points that belong to the object. Since the camera in the real-world environment is positioned closer to the objects than the camera in the dataset, this means more points in the image will represent an object, so choosing more points gives us more geometrical information about the object. Using more points is also a good way of making outliers (points in the mask with poor depth values) less influential in the calculations.

A set of 2000 points is then randomly chosen from the object depth values. The value of 2000 points, after some testing, was chosen because it is a good trade-off between accuracy and speed. The fact that the points are chosen at random also means that the set of points will have a proper distribution throughout the whole object. With the points chosen, the point cloud value for each point is calculated, following Equations 3.1, 3.2 and 3.3.

$$z = \frac{depth}{scale} \tag{3.1}$$

$$x = \frac{(a - cx) \times z}{fx} \tag{3.2}$$

$$y = \frac{(b - cy) \times z}{fy} \tag{3.3}$$

In these equations, *depth* represents the depth value for a point, *scale* depends on the measurement units used by the camera, $a$ and $b$ are the pixel coordinates for a point and $cx$, $cy$, $fx$ and $fy$ are intrinsic camera parameters. This is the same way the ZED camera calculates its point cloud, however, the SDK calculates a point cloud for the whole image and each point can be accessed individually, which increases computation times significantly more.

Following all this pre-processing, the Densefusion model takes as input a crop of the camera image frame (in the dimensions of the object's bounding-box), the calculated point cloud, the 2000 points from the mask, and the object's ID. The pose estimation is outputted in the form of rotation quaternions and a translation vector, which then goes through two iterations of pose refinement, to produce better results. Doing two iterations is faster and has comparable accuracy to doing more iterations.

The output of Densefusion is not the final pose estimation, however. These values create a transformation matrix that transforms the points in the object 3D model from the canonical frame

defined in the dataset to the camera coordinate frame. To obtain a final pose estimation, post-processing is crucial. In post-processing, the set of optimal grasping points (which were loaded from one of the configuration files), are transformed according to the transformation given by the Densefusion model. The center of this group of points is then calculated to give the optimal point of contact for grasping.

# Chapter 4

# Experiments and Results

## 4.1 Dataset

In order to test and validate our proposed system, and given that a machine learning model is only as good as the data it is fed, a dataset was chosen to train and test the computer vision module. The dataset used to train both the object detection and the pose estimation models was the YCB-Video dataset, which uses 20 objects from the YCB Object Model Set [35]. Using a RGB-D camera, different groups of the 20 objects were filmed in different environments, with varying backgrounds, different lighting conditions, and situations of occlusion. The camera moved in relation to the objects to create different viewpoints of every object, which resulted in a total of 133827 frames of video. Furthermore, the 3D point cloud models of every object were obtained with a scanning rig.

From a quantitative standpoint, this dataset has 133827 RGB images (with 480x640 pixels) and their corresponding depth maps, as well as a collection of the point cloud 3D models of all the objects. For every frame and each of the objects in the frame, there are annotations for class label, ground-truth segmentation mask, and pose values of the object (given as a transformation, from the canonical frame of the object to the camera coordinate frame). The camera's intrinsic parameters and its positions in the world are also given for each frame.

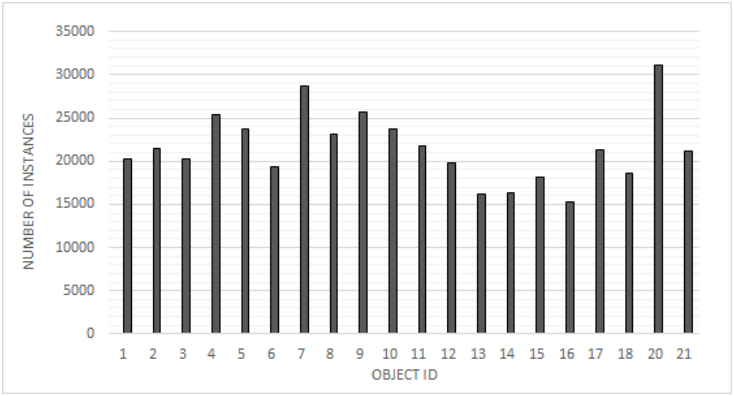
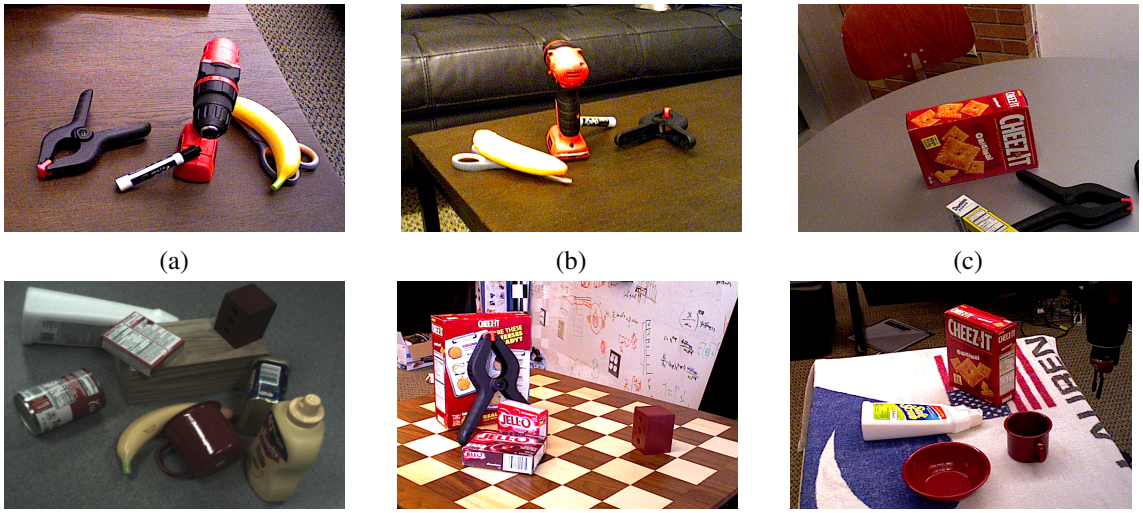



Figure 4.1: Distribution of number of instances per object

In order to evaluate if the dataset was imbalanced, the number of instances per type of object in the dataset was plotted. As can be seen in Fig. 4.1, the distribution of instances for each object is fairly balanced, and each object has, at least, 15000 instances. This contributes to a more balanced training of the machine learning models and helps to reduce bias. This characteristic explains why this dataset is one of the most important and one the most used for the training and validation of the pose estimation models reviewed in Section 2.3.3.

From a qualitative standpoint, the YCB-Video dataset presents a lot of advantages in terms of variability. One of them is the difference in background, which can be both uncluttered (Fig. 4.2a) and cluttered (Fig. 4.2e), as well as light or dark, and simple or complex (as is shown in all the images of Fig. 4.2).




(a) (b) (c)

(d) (e) (f)

Figure 4.2: Selection of illustrative images from the dataset

When comparing Figs. 4.2a and 4.2b, it can be seen that the same objects, in the same scene, are shown in a variety of viewpoints. This is mainly due to the movement of the camera, which helps to create a substantial range of poses for the training of the pose estimation model. Another important aspect of the dataset is the change in lighting that occurs between scenes. Comparing Figs. 4.2b and 4.2d, there are examples of an overexposed and an underexposed image, respectively. This variability can help to make any model more robust to lighting changes.

Finally, it is also important to mention the way occlusions are created for an object. The varying viewpoints help create natural occlusions - when objects overlap - and truncation of an object - when it leaves the field of view of the camera (such as in Fig. 4.2c). Moreover, the arrangement of the objects in the scene also helps to create complex viewpoints of an object, where overlaps may occur (for example, in figs. 4.2a, 4.2d and 4.2e). All these aspects were relevant to the choice of dataset, given that with more variability in different aspects for each object, the robustness of the trained model will improve.

## 4.2 Offline Testing Phase

In this section, the training and validation of each ML model in the Computer Vision Module, will be discussed. The models were trained and validated on the dataset analyzed in the section above (Section 4.1). After this, the pose estimation model was tested using as inputs the outputs given by the object detection model.

### 4.2.1 Training the Object Detection Model

The training of this model was heavily based on the idea of transfer learning. This method consists of using pre-trained weights from a model - obtained on one dataset - as a starting point for training on a different dataset. This approach speeds up training time and increases performance, but only when the pre-trained model learned relevant general features on a balanced dataset. In our first approach, the pre-trained weights were used for every branch, except the network heads; on the second approach, the pre-trained weights were utilized only for the backbone feature extraction network, except for the final layers. The first approach resulted in a 14% drop in mask generation accuracy and a 5% drop in class label accuracy, in relation to the second approach. Although the first approach only takes 33 hours to train compared to the 46 hours of training in the second approach, the accuracy of the segmentation masks is paramount, so this is an acceptable trade-off between speed and accuracy.

Table 4.1: Comparison of original and fine-tuned hyper-parameter values

| **Hyper-parameter** | **Original value** | **Fine-tuned value** |
|---|---|---|
| Number_classes | NA | 21 |
| Max_gt_instances | 100 | 10 |
| Detection_min_confidence | 0.8 | 0.7 |
| ROI_minibatch_size | 512 | 128 |
| RPN_nms_threshold | 0.7 | 0.5 |
| Learning_rate | 0.02 | 0.002 |
| RPN_class_loss | 1.0 | 1.0 |
| RPN_bbox_loss | 1.0 | 0.01 |
| Mrcnn_class_loss | 1.0 | 1.0 |
| Mrcnn_bbox_loss | 1.0 | 0.01 |
| Mrcnn_mask_loss | 1.0 | 10.0 |

Given that the original hyper-parameters of the model were optimized for the COCO dataset, the hyper-parameters of the object detection model were tuned. For that, Tensorboard (a visualization and optimization tool for tuning parameters in Tensorflow) was used. In Table 4.1, a comparison between the original and the fine-tuned model parameters is shown.

The *number_classes* parameter takes the value of 21, since there are 20 objects in the dataset and one background class. *Max_gt_classes* was changed to 10 from 100 due to the fact that each image in the dataset has a maximum of 10 object's instances. To allow for more predictions

where the object may be occluded or where the error might influence the detection confidence, *detection_min_confidence* was decreased to 0.7.

This network generates region proposals per image depending on the values of *ROI_minibatch_size* and *RPN_nms_threshold*. A third of the number of region proposals are positive proposals, and the rest are negative proposals. Since the number of objects per image is small, *RPN_nms_threshold*, needs to be reduced in order to reduce the number of region proposals and improve performance. ROI_minibatch_size is decreased to preserve the ratio of 1:3 positive to negative proposals and prevent over-fitting on negative ROI proposals. In the original implementation of Mask-RCNN (implemented in Caffe), the learning rate was 0.02. However, in Tensorflow, this causes the weights to increase exponentially, due to the different optimizers used in Caffe and Tensorflow. Decreasing the learning rate by a factor of 10 stabilizes weight adjustment.

For this implementation, the main outputs needed by the object detection model are the class label and the segmentation mask, which must be as accurate as possible. With that in mind, *RPN_bbox_loss* and *mrcnn_bbox_loss* can be decreased to have less input in loss calculation. The loss weight responsible for the segmentation mask, however, should be increased to improve accuracy. This is the reason for increasing *mrcnn_mask_loss* by a factor of 10; a larger value does not improve performance.

Table 4.2: Results for bounding-box and segmentation mask precision on all objects

| **Class name** | **Object ID** | **Bounding-box AP** | **Segmentation mask AP** |
|---|---|---|---|
| Master chef can | 1 | 31.7 | 37.3 |
| Cracker box | 2 | 32.3 | 36.9 |
| Sugar box | 3 | 31.3 | 37.1 |
| Tomato soup can | 4 | 32.0 | 37.0 |
| Mustard bottle | 5 | 31.5 | 36.5 |
| Tuna fish can | 6 | 28.9 | 35.8 |
| Pudding box | 7 | 31.9 | 37.1 |
| Gelatin box | 8 | 32.3 | 37.4 |
| Potted meat can | 9 | 29.6 | 35.9 |
| Banana | 10 | 32.2 | 37.1 |
| Pitcher base | 11 | 31.8 | 36.8 |
| Bleach cleanser | 12 | 30.7 | 36.0 |
| Bowl | 13 | 32.2 | 37.2 |
| Mug | 14 | 31.8 | 35.8 |
| Power drill | 15 | 29.5 | 36.8 |
| Wood block | 16 | 32.3 | 37.0 |
| Scissors | 17 | 28.6 | 35.6 |
| Marker | 18 | 29.1 | 36.1 |
| Clamp | 19 | 28.9 | 35.7 |
| Foam brick | 20 | 31.0 | 36.7 |

The results in Table 4.2 show that the average bounding-box and segmentation mask precision is comparable to the baseline values obtained on the COCO dataset in [34]. These values produce

accurate segmentation masks for most objects, which was the intended result.

The worst performing objects are the smaller objects (tuna fish can, potted meat can and marker) and the more complex objects (power drill, scissors and clamp). Typically, in object detection, small objects always suffer from worse detection than larger ones, due to being more influenced by noise because of their reduced pixel resolution. Complex objects, on the other hand, need the network to learn more complex features than simple objects, which also influences accuracy.

### 4.2.2 Pose Estimation Model Training

Since Densefusion was already trained on the YCB-Video dataset, pre-trained weights and parameters were readily available for the implementation described in [30]. Therefore, this model's training strategy was to validate it with the pre-trained weights on the YCB-Video dataset. There was no need for parameter optimization, as the parameters discovered in [30] were already optimal. The results of validation are presented in Table 4.3, where ADD-S represents the average distance between every point in the ground-truth pose and the estimated pose.

Table 4.3: Results of validation of the pose estimation model

| **Class name** | **Object ID** | **ADD-S (%)** |
|---|---|---|
| Master chef can | 1 | 95.8 |
| Cracker box | 2 | 94.7 |
| Sugar box | 3 | 97.1 |
| Tomato soup can | 4 | 93.4 |
| Mustard bottle | 5 | 97.0 |
| Tuna fish can | 6 | 96.3 |
| Pudding box | 7 | 95.5 |
| Gelatin box | 8 | 98.0 |
| Potted meat can | 9 | 91.1 |
| Banana | 10 | 96.1 |
| Pitcher base | 11 | 96.9 |
| Bleach cleanser | 12 | 95.3 |
| Bowl | 13 | 86.1 |
| Mug | 14 | 96.7 |
| Power drill | 15 | 95.5 |
| Wood block | 16 | 89.2 |
| Scissors | 17 | 95.0 |
| Marker | 18 | 97.1 |
| Clamp | 19 | 70.7 |
| Foam brick | 20 | 91.8 |

The worst performing objects are the bowl, the clamp and the tuna fish can. This is due to the bowl and the clamp being symmetrical objects, which influences pose estimation, since the same viewpoint results in different poses, but also due to the fact that they are texture-less objects,

which makes it harder for the network to extract feature keypoints from them. The tuna fish can shows poor results due to its small size (for the same reasons that it had poor detection results).

### 4.2.3 Computer Vision Module Testing

After training both models that compose the Computer Vision Module on the YCB dataset, the complete pipeline was analysed. In this testing phase, the output of the localization model was tested using the as input the outputs given by the object detection model. This is an important test to understand the impact of detection errors on pose estimation.

One of the first issues encountered was loading both model weights into the GPU at the same time. Since model weights take up a lot of GPU memory, the pose estimation model would fail, due to the GPU running out of available memory during calculations. The solution to this problem was to load the less computationally expensive model (object detection model) weights into CPU memory while keeping the pose estimation model running on the GPU. After solving this problem, the Computer Vision module was used to identify and estimate the position of objects in the dataset's validation subset. The results in Table 4.4 show the accuracy of the pose estimations, following the same metric used in 4.3.

Table 4.4: Results of testing on the Computer Vision Module

| **Class name** | **Object ID** | **ADD-S (%)** |
|---|---|---|
| Master chef can | 1 | 77.5 |
| Cracker box | 2 | 72.7 |
| Sugar box | 3 | 72.6 |
| Tomato soup can | 4 | 71.5 |
| Mustard bottle | 5 | 78.4 |
| Tuna fish can | 6 | 70.0 |
| Pudding box | 7 | 79.4 |
| Gelatin box | 8 | 80.3 |
| Potted meat can | 9 | 74.0 |
| Banana | 10 | 78.6 |
| Pitcher base | 11 | 77.2 |
| Bleach cleanser | 12 | 78.1 |
| Bowl | 13 | 69.8 |
| Mug | 14 | 80.6 |
| Power drill | 15 | 73.0 |
| Wood block | 16 | 70.2 |
| Scissors | 17 | 73.9 |
| Marker | 18 | 71.4 |
| Clamp | 19 | 56.5 |
| Foam brick | 20 | 68.8 |

The results in Table 4.4 drop close to 20 points in accuracy, for almost every object, in relation

to the results presented in Table 4.3. This drop is an expected outcome since the original Densefusion model uses exact segmentation masks from the ground-truth, which gives an accurate region of points belonging to a given object. The object detection model, however, has some error in its segmentation masks. This error results in the segmentation mask covering parts of different objects, especially when multiple objects are close together, on one extreme, or not covering the entire object, on the other. Therefore, the geometric information passed on to the pose estimation model is erroneous in some cases, which translates to less accurate results.

## 4.3 Online Testing Phase

With the training and offline testing completed, the module needed to be deployed and tested in a live manner directly from the camera feed. For online testing, a group of 6 objects was selected from the 20 in the dataset. It was important to choose two objects with the worst performance in offline testing, as well as two objects that were top-performers. This enables the testing to validate if the patterns observed in the simulation are reproduced in a real environment. Due to the industrial context of the project, two tools were also selected. The list of objects used is as follows:

- Bowl;
- Clamp;
- Gelatin box;
- Marker;
- Mug;
- Power drill;

The test setting was an environment with controlled lighting and a table where the objects would be placed. A ZED camera was positioned in front of the table and connected (via USB 3.0) to a computer running the system and equipped with a GeForce RTX 2080 Ti GPU. A collaborative robotic arm with a 5kg payload (Universal Robots' UR5) was used for the grasping experiments. The following tests were performed:

1. Testing the Data Acquisition Module (frame acquisition and analysis);
2. Testing the Object Detection and Segmentation model for identifying each object in various positions and visually validate the object's produced mask;
3. Testing the final Computer Vision Module output (Identification + Pose) with a grasping experiment and measuring the processing time of each component;

Due to complications in compatibility between library versions, it is essential to note that the original object detection model used in offline testing had to be replaced. This resulted in the model having to be re-trained. Since time constraints did not allow for the re-training of the model on the entire dataset (which would have taken 40+ hours), the training dataset used for this model was a smaller version with only 2000 instances per object. This presents issues that will be mentioned in the discussion of results; however, it also presents an exciting analysis opportunity on the effects of using a smaller dataset with transfer learning.

### 4.3.1 Data Acquisition Module Testing and Analysis

In this test, it was needed to validate the retrieval of frames from the ZED camera, as well as to adjust the threshold value for the calculation of changes between frames. The testing method for the threshold calculation was based on three configurations of objects on the workspace: a small object by itself, a large object by itself, and a group of objects.

For each scenario, a total of 30 frames were retrieved. For each pair of consecutive frames, the SSIM algorithm (explained in 3.1.1.2) was applied, and the values recorded. The values taken correspond to the following situations: no movements between frames, a small movement (<2cm) of a small/large object between frames, a big movement (>12cm) of a small/large object, and the removal of an object between frames. The average of the values taken was calculated and are shown in Table 4.14

Table 4.5: Results of SSIM experiment

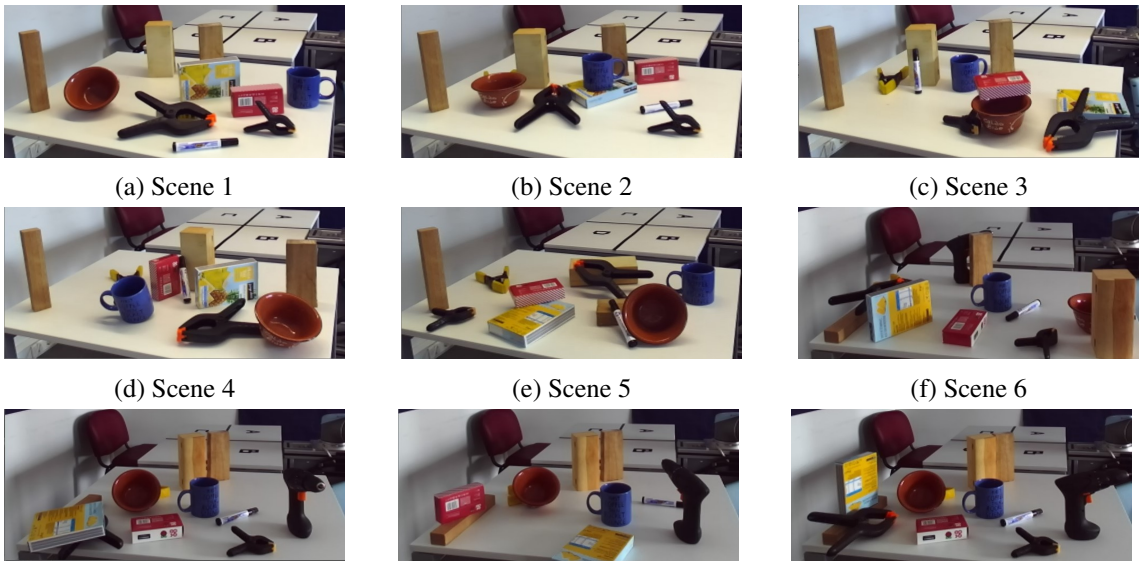
| **Scenario** | **No change** | **Small movement (<2cm)** | **Big movement (>12cm)** | **Removal** |
|---|---|---|---|---|
| Only small object | 0.993668 | 0.9833307 | 0.979674 | 0.966702 |
| Only large object | 0.990412 | 0.981307 | 0.949943 | 0.963506 |
| Multiple objects | 0.993004 | 0.988883 | 0.968512 | 0.953972 |

The values presented demonstrate that even with the object's removal from the frame, the SSIM value is still very close to its maximum of 1. This is to be expected as the majority of the image remains unchanged, and an object makes up a small part of the entire frame. However, these values show there is a clear correlation between the decrease of the SSIM value and the magnitude of the change. The difference between the values for small movements and big movements shows that it is possible to detect a small change of less than 2 centimeters in the object position. However, this is unnecessary since it is expected that a small movement will not affect pose estimation a significant amount due to the associated errors with the pose calculation.

### 4.3.2 Object Detection and Segmentation Testing

In this test, the aim was to validate the object detection and segmentation model and evaluate its performance in a typical grasping setting. Several objects were arranged on the workspace (as can

be seen in Fig. 4.3), and close to ten seconds of frames (at 15fps) were retrieved and analyzed by the Object Detection and Segmentation model. The results measure the percentage of accurate, inaccurate, and no detections, as well as the model's average confidence.



(a) Scene 1 (b) Scene 2 (c) Scene 3

(d) Scene 4 (e) Scene 5 (f) Scene 6

(g) Scene 7 (h) Scene 8 (i) Scene 9

Figure 4.3: Overview of all the scenarios used in detection testing

One issue noticed during real-time testing was the occurrence of a flickering effect, where the object would not be detected during a couple of frames. This is due to small imperceptible changes (to the human eye) between consecutive frames, such as illumination differences, pixel shifting or noise. To solve this, a temporal smoothing phase was implemented, which takes an average of the pixels in three frames, resulting in less variability and, therefore, reducing this effect.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 157 | 63.4 | 0 | 36.6 | 0.78 |
| Bowl | 13 | 1 | 157 | 100 | 0 | 0 | 0.96 |
| Mug | 14 | 1 | 157 | 100 | 0 | 0 | 0.99 |
| Large marker | 18 | 1 | 157 | 79.0 | 0 | 21.0 | 0.91 |
| Clamp | 19 | 2 | 157 | 0 | 100 | 0 | NA |

Table 4.6: Detection results on scene 1

In scene 1 (Fig. 4.3a), the intention was to establish a baseline result where every object is in a natural and easy to detect position. Like is shown in Table 4.6, most of the objects were well detected, except for the clamp, which was inaccurately detected as scissors. This is due to both objects having very similar features and the clamp being a complex object. Given the reduced size

dataset used in training the model, it did not learn how to differentiate these features in a complex object correctly.

In scene 2 (Fig. 4.3b), the three main changes were the variation of viewpoint of the bowl, the overlap between one of the gelatin boxes and the mug, and the partial occlusion of the marker by the smaller clamp. Like is shown in 4.7, this had an immediate impact on the detection of the gelatin box. There is only 50% accurate detections which correspond to the non-occluded box. The inaccurate detections of the bowl correspond to it being identified as a mug, due to the shape in this viewpoint being similar to that of a mug. The partial occlusion of the marker causes the marker to be undetected.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 153 | 50 | 0 | 50 | 0.85 |
| Bowl | 13 | 1 | 153 | 81.7 | 18.3 | 0 | 0.72 |
| Mug | 14 | 1 | 153 | 100 | 0 | 0 | 0.98 |
| Large marker | 18 | 1 | 153 | 0 | 0 | 100 | NA |
| Clamp | 19 | 2 | 153 | 0 | 74.5 | 25.5 | NA |

Table 4.7: Detection results on scene 2

In scene 3 (Fig. 4.3c), the smaller gelatin box is overlapped on top of the bowl, creating a partial occlusion of the bowl. This occlusion creates a viewpoint very similar to the mug viewpoint, which explains the inaccurate detection rate (which can be seen in Table 4.8). The smaller gelatin box is still well detected even on top of the bowl; however, the occluded gelatin box remains undetected, while it is occluded by the clamp. The marker, in an upright position, presents good detection results.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 116 | 50.0 | 0 | 50.0 | 0.89 |
| Bowl | 13 | 1 | 116 | 88.8 | 11.2 | 0 | 0.90 |
| Mug | 14 | 1 | 116 | 100 | 0 | 0 | 0.97 |
| Large marker | 18 | 1 | 116 | 89.7 | 0 | 10.3 | 0.76 |
| Clamp | 19 | 3 | 116 | 0 | 66.7 | 33.3 | NA |

Table 4.8: Detection results on scene 3

Scene 4 (Fig. 4.3d) presents all the objects closer together, with partial occlusions to the smaller gelatin box. This proximity between objects makes it hard for the detection model to

correctly identify the objects. In fact, the two best performing objects (according to the results in Table 4.9) - the bowl and the mug - are the more isolated in the group.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 115 | 4.3 | 0 | 95.7 | 0.72 |
| Bowl | 13 | 1 | 115 | 100 | 0 | 0 | 0.95 |
| Mug | 14 | 1 | 115 | 100 | 0 | 0 | 0.97 |
| Large marker | 18 | 1 | 115 | 0 | 0 | 100 | NA |
| Clamp | 19 | 2 | 115 | 0 | 44.8 | 55.2 | NA |

Table 4.9: Detection results on scene 4

Scene 5 (Fig. 4.3e) presents a frontal view of the top of the bowl. Since this viewpoint is very unique to this type of object, the detection is 100% accurate (Table 4.10). In a similar situation to scene 4, the marker is undetected due to its proximity to the bowl and the wood block behind.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 113 | 100 | 0 | 0 | 0.87 |
| Bowl | 13 | 1 | 113 | 100 | 0 | 0 | 0.87 |
| Mug | 14 | 1 | 113 | 72.6 | 0 | 27.4 | 0.74 |
| Large marker | 18 | 1 | 113 | 0 | 0 | 100 | NA |
| Clamp | 19 | 3 | 113 | 0 | 0 | 100 | NA |

Table 4.10: Detection results on scene 5

As can be seen in Table 4.11, even with the heavy occlusion of the bowl in scene 6 (Fig. 4.3f), is still presents a reasonable detection rate. In this scene, the power drill was introduced, however, due to its occlusion, the viewpoint generated is not representative enough of the shape of the object, in order for the model to detect it.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 140 | 91.4 | 0 | 8.6 | 0.81 |
| Bowl | 13 | 1 | 140 | 88.6 | 0 | 11.4 | 0.83 |
| Mug | 14 | 1 | 140 | 100 | 0 | 0 | 0.97 |
| Power drill | 15 | 1 | 140 | 0 | 0 | 100 | NA |
| Large marker | 18 | 1 | 140 | 100 | 0 | 0 | 0.92 |
| Clamp | 19 | 2 | 140 | 0 | 0 | 100 | NA |

Table 4.11: Detection results on scene 6

For scene 7, the results of Table 4.12 show that even when occlusion does not occur, certain positions of the gelatin boxes cause the model not to be able to detect it. This is due to the small amount of viewpoints in the reduced dataset used for online testing. All other objects, except the clamp, have a 100% detection rate.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 209 | 50 | 0 | 50 | 0.83 |
| Bowl | 13 | 1 | 209 | 100 | 0 | 0 | 0.83 |
| Mug | 14 | 1 | 209 | 100 | 0 | 0 | 0.96 |
| Power drill | 15 | 1 | 209 | 100 | 0 | 0 | 0.83 |
| Large marker | 18 | 1 | 209 | 100 | 0 | 0 | 0.92 |
| Clamp | 19 | 1 | 209 | 0 | 100 | 0 | NA |

Table 4.12: Detection results on scene 7

Scene 8 (shown in Fig. 4.3h) presents a fairly standard view of the objects, this time with the removal of the clamp from the workspace. Table 4.13 shows that most of the objects are well detected, with the exception of the marker and the power drill.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 143 | 97.2 | 0 | 2.8 | 0.83 |
| Bowl | 13 | 1 | 143 | 100 | 0 | 0 | 0.88 |
| Mug | 14 | 1 | 143 | 100 | 0 | 0 | 0.99 |
| Power drill | 15 | 1 | 143 | 24.5 | 0 | 75.5 | 0.73 |
| Large marker | 18 | 1 | 143 | 0 | 0 | 100 | NA |

Table 4.13: Detection results on scene 8

Finally, in scene 9 (Fig. 4.3i), even with very clear viewpoints of the objects, detection results are poor for the gelatin box, the power drill and the clamp.

| **Name** | **ID** | **Count** | **Number of frames** | **Accurate Detection (%)** | **Inaccurate Detection (%)** | **No detection (%)** | **Average Confidence (0-1)** |
|---|---|---|---|---|---|---|---|
| Gelatin box | 8 | 2 | 161 | 50 | 0 | 50 | 0.8 |
| Bowl | 13 | 1 | 161 | 100 | 0 | 0 | 0.91 |
| Mug | 14 | 1 | 161 | 100 | 0 | 0 | 0.99 |
| Power drill | 15 | 1 | 161 | 0 | 0 | 100 | NA |
| Large marker | 18 | 1 | 161 | 91.9 | 0 | 8.1 | 0.76 |
| Clamp | 19 | 2 | 161 | 0 | 50 | 50 | NA |

Table 4.14: Detection results on scene 9

As can be seen, the reduced dataset has a big impact on detection performance. The main issues with the use of so little instances of each object results in the following problems:

- For complex objects (such as the clamp), the model needs more data to be able to distinguish features between objects, resulting in situations of incorrect classification;
- Even for objects with simple features (like the gelatin box, the bowl, and the marker), the model over-fitted to certain viewpoints (which were more prevalent in the reduced dataset), which resulted in no detection in certain object positions;
- When objects were too close together, the model had trouble in correctly detecting them, since this ability is a more complex feature which needed more training data;

However, as in any other problem, besides the issues, there are some positive aspects worth mentioning regarding the model's performance:

- The model is able to generalize, and correctly identify objects which are different from the dataset, even with limited data. This is the case of the gelatin boxes - which have different

height and width in relation to the original object -, the bowl - which is taller and less wide than the original model - and the mug - which has a different color and is more narrow than the dataset model;

- With the limited amount of training data, the model is still able to detect accurately and with great confidence, the less complex objects, like the gelatin boxes, the bowl, and the mug;
- Even with heavy occlusion (as seen in Fig. 4.3f), the model can correctly detect the bowl;

One problem that is common to both the reduced dataset and the original dataset is the lack of variety in the number of objects per image. Since this dataset was originally made for pose estimation, the main objective was to create a complex environment with proximity and occlusion between objects. This is the right approach for pose estimation; however, in the object detection model's training, it can lead to over-fitting.

In testing, when an object was positioned alone in the workspace, the detection rate dropped significantly and was very inconsistent. However, when other objects were positioned close to it, the detection rate improved and was more stable. This issue is due to over-fitting for situations with a group of objects.

### 4.3.3 Grasping Experiments

In the third and final test, the computer vision module was validated. The testing methodology used was a grasping experiment on the 5 of the 6 objects that were chosen for testing. This was due to the available gripper being too small to be able to grasp the power drill.

Figure 4.4: Illustration of one grasping position per object

Each object was placed in a different workspace position, surrounded by other objects. A grasping attempt was made for each position, for a total of five attempts. Illustrations of one of the

positions for each object are presented in Fig. 4.4. The results obtained were recorded in Table 4.15.

| **Scene** | **Object ID** | **Successful attempts (%)** | **Average error (cm)** |
|---|---|---|---|
| 1 | 8 | 60 | 2 |
| 2 | 8 | 60 | 5 |
| 3 | 13 | 20 | 2.7 |
| 4 | 14 | 80 | 0.5 |
| 5 | 18 | 60 | 3.5 |
| 6 | 19 | 40 | 0.75 |

Table 4.15: Grasping results

For each object, the Computer Vision Module ran detection, generation of the segmentation mask, and pose estimation. The coordinates of the center point of the object were obtained, and a transformation was applied from the center point to the optimal grasping point to enable correct grasping.

Due to the gripper model not being readily available and also time-restrictions for testing, the transformation of the set of optimal grasping points (discussed in Section 3.2.2.2) was not implemented. However, this test serves as a proof of concept, since the transformation of all object points is equivalent to the transformation of only a subset of optimal grasping points.

Furthermore, since the primary purpose of this project is to develop a system for object recognition and localization, the main focus was on the creation of a pipeline with those capabilities where the post-processing step involved in grasping was not the main focus.

The worst performing objects are the bowl and the clamp. These are expected results when looking at both the results in Table 4.3 and in Section 4.3.2. The clamp, being a complex object with the worst result in Table 4.3 and with poor detection results, was unlikely to present successful grasping attempts. As for the bowl, it was the second-worst performing object in Table 4.15, due to being a symmetrical object. Both the gelatin boxes have comparable success rates; however, the average error is more significant for the larger box, since small errors in the pose's estimation have a more substantial influence in larger objects.

One of the main sources of pose estimation error during this phase of testing was the inconsistency and inaccuracy of the segmentation masks outputted by the object detection model. This was already an issue in Section 4.2.2, however, the small acquisition errors from the camera also contribute to the overall detection error. Additionally, as is concluded in [1], the ZED camera has an associated error for depth data (Fig. 4.5). Due to the workspace being at a maximum distance of 1.1m from the camera, this depth estimation error is very small, however, it is still influential.

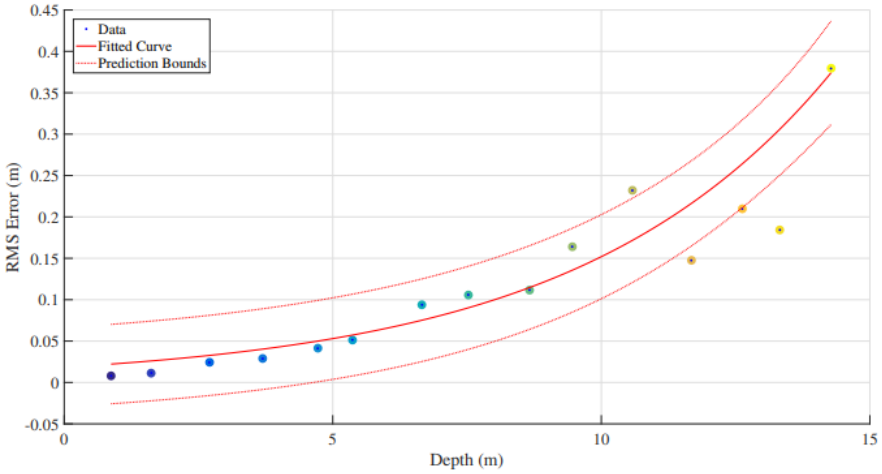



Figure 4.5: RMS error in depth data for 720p resolution [1]

In this phase of testing, given the real-time constrains for grasping applications, there was also a measurement of the processing time for each component: data acquisition module, object detection, and segmentation model, and 6D object pose estimation model. On average, frame acquisition and analysis took 10ms. The object detection and object localization models took on average 346ms and 29.6ms, respectively. Since the computer vision module runs parallel with frame acquisition, the whole program takes about 375ms per frame on average. This means it can run at about 2.6fps.

A frame-rate of 2.6fps is very slow when applied to normal video-playback or complex tasks, such as real-time autonomous driving. However, considering the not very dynamic environment on which this system aims to work in and considering the slow speed of a full grasping attempt by the robotic manipulator used for testing (about 5s), this frame-rate is adequate.

## 4.4 Limitations

With the experiments and tests performed, some limitations in our approach were found. One of the limitations of the proposed system is that the output of ideal gripper rotations is based on a previously created configuration file. While this is a valid solution that resolves many errors that can occur when calculating these rotations automatically, while also allowing the user to choose the ideal grasping according to his own preference and application, it also asks much overhead work from anyone who tries to implement the system. However, since the system's main objective is to output object detection and pose estimations for executing the grasping tasks, it is not a very impactful limitation. Overall, the trade-off between allowing for user adaptability and user overhead work may be beneficial.

The developed system also affected errors in the acquisition of visual data and lighting conditions due to the characteristics of the ML models used. Lighting is always a big part of errors

introduced in CV models, so this is an acceptable limitation. The main way of solving this limitation would be to train the model on an even bigger dataset with a range of lighting conditions, which would be too costly.

A severe limitation is the fact that the detection rate drops when an object is by itself in the workspace due to over-fitting for situations with a group of objects. This is an issue of training the detection model on a pose estimation dataset. Even though, when it comes to detection, the system shows that it generalizes to objects with different dimensions and characteristics from the dataset, in the case of pose estimation, objects that differ too much from the dataset models will not be correctly estimated since the pose estimation pipeline takes into account the canonical object 3D model from the dataset. Another limitation is the fact that the system cannot choose the absolute optimal object to grasp in a cluttered situation. It only chooses the objects that it has the highest confidence for, without taking into account its position in relation to the other objects.

# Chapter 5

# Conclusions and Future Work

## 5.1 Conclusions

In this thesis, the problem of developing an automatic object recognition and pose estimation system was addressed. This system combines object localization and identification to improve robotic grasping. The main contribution of this work was the development of a framework integrating two different ML models to create a system for more complete and autonomous grasping tasks.

Extensive research was carried out on the current state of Machine Learning methods, particularly Convolutional Neural Networks. This research resulted in a deeper understanding of this field and its applications in Computer Vision: from the most basic to the more advanced developments that have occurred throughout the years. The main state-of-the-art approaches to object detection/recognition and pose estimation were reviewed and compared, which led to more informed development of the proposed system.

This thesis' main focus was on the integration of two different ML models for detection and pose estimation. Both Mask-RCNN and Densefusion were correctly integrated and show adequate results in a real testing environment. The developed system is capable of detecting movements and differences between video frames, therefore reducing the need for constant calculation and inference on each video frame. The proposed solution runs entirely automatically, after an initial user parameters' selection and can generate grasping outputs from video data at a rate of 2.6fps.

An interesting positive aspect of the proposed solution is the ability of a human to be in control of choosing the optimal grasping points for an object and creating a configuration file with these points, thus allowing for user and application adaptability, while increasing accuracy and speed of the system, since it does not estimate these points during run time.

The main limitations found in the proposed solution were the overhead work needed to output correct gripper rotations for certain object positions, the impact of lighting conditions, and the dependency on previously scanned 3D models of the real objects.

## 5.2 Future Work

Due to time restrictions, the real environment testing had to be done using a smaller dataset for training. The main future work would be to test the system with a detection model trained on the entire dataset in order to get more reliable results. Also, training the model with a our own dataset originally retrieved with the ZED camera and our objects would very likely improve the system's performance (some of the objects used differed in some characteristics from the ones utilized in training).

It would also be interesting to analyze the impact of data augmentation while testing the detection model to improve detection results and the system's overall performance. This data augmentation would be in the form of adding images of single objects to eliminate over-fitting to a group of objects, as well as transforming the pre-existing images in the dataset by adding random rotations, translations, and noise.

A big reason for the performance drop when the entire Computer Vision module was tested was that the segmentation masks outputted by the detection model had errors. One aspect of future work could be to analyze the impact of training the pose estimation model with the segmentation masks outputted by the detection model, instead of the ground-truth masks in the dataset. Finally, an interesting case study would be to create the configuration file with gripper rotations and object pose correspondences using a grasping simulator to test grasping with this functionality.

# References


[1] Luis Enrique Ortiz, Elizabeth V Cabrera, and Luiz M Gonçalves. Depth data error modeling of the zed 3d vision sensor from stereolabs. *ELCVIA: electronic letters on computer vision and image analysis*, 17(1):0001–15, 2018.

[2] Shai Shalev-Shwartz and Shai Ben-David. In *Understanding Machine Learning: From Theory to Algorithms*, chapter Introduction, page 1–10. Cambridge University Press, 2014. doi:10.1017/CBO9781107298019.002.

[3] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. In *Foundations of Machine Learning*, chapter Introduction, pages 1–10. The MIT Press, 2012. ISBN: 026201825X.

[4] Jayesh Bapu Ahire. Perceptron and backpropagation, Oct 2019. URL: https://medium.com/@jayeshbahire/perceptron-and-backpropagation-970d752f4e44.

[5] Jürgen Schmidhuber. Deep Learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. arXiv:1404.7828, doi:10.1016/j.neunet.2014.09.003.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. In *Deep Learning*, chapter 14: Autoencoders. MIT Press, 2016. http://www.deeplearningbook.org.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. In *Deep Learning*, chapter 9: Convolutional Networks. MIT Press, 2016. http://www.deeplearningbook.org.

[8] João Borrego. Mind the gap! bridging the reality gap in visual perception and robotic grasping with domain randomisation. Master's thesis, Instituto Superior Técnico da Universidade de Lisboa, 7 2018.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. URL: https://doi.org/10.1145/3065386, doi:10.1145/3065386.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. arXiv:1409.1556.

[11] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. URL: http://dx.doi.org/10.1109/CVPR.2016.308, doi:10.1109/cvpr.2016.308.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. URL: http://dx.doi.org/10.1109/CVPR.2016.90, doi:10.1109/cvpr.2016.90.

[13] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58:1–35, 2006. URL: ftp://sbai2009.ene.unb.br/Projects/GPS-IMU/George/arquivos/Bibliografia/79.pdf, doi:10.1093/jxb/erm298.

[14] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. doi:10.1007/s11263-009-0275-4.

[15] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8693 LNCS(PART 5):740–755, 2014. arXiv:1405.0312, doi:10.1007/978-3-319-10602-1_48.

[16] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7724 LNCS(PART 1):548–562, 2013. doi:10.1007/978-3-642-37331-2_42.

[17] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6D object pose estimation using 3D object coordinates. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8690 LNCS(PART 2):536–551, 2014. doi:10.1007/978-3-319-10605-2_35.

[18] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In IEEE, editor, *Proceedings of IEEE International Conference on Advanced Robotics (ICAR)*, July 2015.

[19] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. 2018. arXiv:1711.00199, doi:10.15607/rss.2018.xiv.019.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. arXiv:1506.01497, doi:10.1109/TPAMI.2016.2577031.

[21] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448, 2015. arXiv:1504.08083, doi:10.1109/ICCV.2015.169.

[22] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. *Advances in Neural Information Processing Systems*, pages 379–387, 2016. arXiv:1605.06409.

[23] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37, 2016. arXiv:1512.02325, doi:10.1007/978-3-319-46448-0_2.

[24] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Dec:779–788, 2016. arXiv:1506.02640, doi:10.1109/CVPR.2016.91.

[25] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Jan:6517–6525, 2017. arXiv:1612.08242, doi:10.1109/CVPR.2017.690.

[26] Joseph Redmon and Ali Farhadi. YOLO v.3. *Tech report*, pages 1–6, 2018. URL: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[27] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Oct:1530–1538, 2017. arXiv:1711.10006, doi:10.1109/ICCV.2017.169.

[28] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. DPOD: 6D Pose Object Detector and Refiner. 2019. URL: http://arxiv.org/abs/1902.11020, arXiv:1902.11020.

[29] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019. URL: http://dx.doi.org/10.1109/CVPR.2019.00469, doi:10.1109/cvpr.2019.00469.

[30] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martin-Martin, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019. URL: http://dx.doi.org/10.1109/CVPR.2019.00346, doi:10.1109/cvpr.2019.00346.

[31] Chen Song, Jiaru Song, and Qixing Huang. Hybridpose: 6d object pose estimation under hybrid representations, 2020. arXiv:2001.01869.

[32] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, May 2001. doi:10.1109/IM.2001.924423.

[33] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. Deepim: Deep iterative matching for 6d pose estimation. *International Journal of Computer Vision*, Nov 2019. URL: http://dx.doi.org/10.1007/s11263-019-01250-9, doi:10.1007/s11263-019-01250-9.

[34] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2017. arXiv:1703.06870.

[35] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52, 2015.