UNIVERSITY OF SÃO PAULO
SÃO CARLOS SCHOOL OF ENGINEERING

Eduardo Godinho Ribeiro

A Deep Learning Approach to Visual Servo Control and Grasp Detection for Autonomous Robotic Manipulation

São Carlos

2020

Eduardo Godinho Ribeiro

A Deep Learning Approach to Visual Servo Control and Grasp Detection for Autonomous Robotic Manipulation

Master Thesis submitted to the São Carlos School of Engineering of the University of São Paulo in fulfillment of the requirements for the degree of Master of Science - Graduate Program in Electrical Engineering

Research area: Dynamic Systems

Supervisor: Prof. Dr. Valdir Grassi Junior

São Carlos

2020

This is the revised version of the dissertation. The original is available at EESC/USP that hosts the Graduate Program in Electrical Engineering

Catalog card prepared by Patron Service at "Prof. Dr. Sergio Rodrigues Fontes" Library at EESC/USP

Ribeiro, Eduardo Godinho
R848d A Deep learning approach to visual servo control and grasp detection for autonomous robotic manipulation / Eduardo Godinho Ribeiro; Thesis directed by Valdir Grassi Junior. -- São Carlos, 2020.

Master (Thesis) Graduate Program in Electrical Engineering and Research Area in Dynamic Systems – São Carlos School of Engineering, at University of São Paulo, 2020.

1. Robotic Grasping. 2. Visual servoing. 3. Deep learning. I. Title.

Elena Luzia Palloni Gonçalves – CRB 8/4464

Eduardo Godinho Ribeiro

Uma Abordagem Baseada em Aprendizagem Profunda para Controle Servo-Visual e Detecção de Pontos de Preensão para Manipulação Robótica Autônoma

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Ciências - Programa de Pós-Graduação em Engenharia Elétrica.

Área de concentração: Sistemas Dinâmicos

Orientador: Prof. Dr. Valdir Grassi Junior

São Carlos

2020

Trata-se da versão corrigida da dissertação. A versão original se encontra disponível na EESC/USP que aloja o programa de Pós-Graduação de Engenharia Elétrica.

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Ribeiro, Eduardo Godinho
R484u Uma abordagem baseada em aprendizagem profunda para controle servo-visual e detecção de pontos de preensão para manipulação robótica autônoma / Eduardo Godinho Ribeiro; orientador Valdir Grassi Junior. São Carlos, 2020.

Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia Elétrica e Área de Concentração em Sistemas Dinâmicos -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2020.

1. Preensão robótica. 2. Controle servo-visual. 3. Aprendizagem profunda. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE JULGAMENTO

Candidato: Bacharel **EDUARDO GODINHO RIBEIRO.**

Título da dissertação: "Uma abordagem baseada em aprendizagem profunda para controle servo-visual e detecção de pontos de preensão para manipulação robótica autônoma".

Data da defesa: 16/04/2020

Comissão Julgadora:	**Resultado:**
Prof. Dr. **Valdir Grassi Junior** **(Orientador)** (Escola de Engenharia de São Carlos – EESC/USP)	Aprovado
Prof. Titular **Glauco Augusto de Paula Caurin** (Escola de Engenharia de São Carlos – EESC/USP)	Aprovado
Prof. Dr. **Luiz Chaimowicz** (Universidade Federal de Minas Gerais/UFMG)	Aprovado

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica:
Prof. Titular **Ivan Nunes da Silva**

Presidente da Comissão de Pós-Graduação:
Prof. Titular **Murilo Araujo Romero**

To my parents, sisters and aunts. In memory of my grandparents.

ACKNOWLEDGEMENTS

To the University of São Paulo, for excellence in teaching and research, especially the Electrical and Computer Engineering Department of the São Carlos School of Engineering.

To the Brazilian National Council for Scientific and Technological Development - CNPq for the Master's scholarship and to the São Paulo Research Foundation - FAPESP for the assistance to participate in scientific conferences.

To my advisor Valdir Grassi Jr, for the availability, support and guidance.

To the USP Robotics Center - CRob and the National Institute of Science and Technology for Cooperative Autonomous Systems - InSAC, headed by professor Marco Henrique Terra, for the infrastructure and technical support.

To my parents, Carlos and Iêda, sisters, Paula and Patrícia, aunts, uncles and grandparents who have always made possible, helped and encouraged my total dedication to studies, especially in these two years of masters. I also thank my father and my aunt Maria José for the distinguished displaying of kindness and concern, and my aunt Regina, for the support with the real estate red tape.

To the friends of the Intelligent Systems Laboratory - LASI, for easing the tension of graduate school and for always cooperating in solving research problems, especially Nicolas, Kaio, Raul and Elizandra.

To the people I met in São Carlos, especially the ones that broadened my world view, my longtime friends from Lavras, my nephews and brother-in-law, my cat and my dog, for the moments of distraction.

To those who believe in the progress of science and education as a means of transforming lives and obliterating unfounded and prejudiced beliefs, for the motivation.

Finally, I thank God for filling the void of existence and for being present as the primordial foundation of every synapse that led me to complete this work.

"Look again at that dot. That's here. That's home. That's us. (...) The Earth is a very small stage in a vast cosmic arena. Think of the endless cruelties visited by the inhabitants of one corner of this pixel on the scarcely distinguishable inhabitants of some other corner, how frequent their misunderstandings, how eager they are to kill one another, how fervent their hatreds. Think of the rivers of blood spilled by all those generals and emperors so that, in glory and triumph, they could become the momentary masters of a fraction of a dot. Our posturings, our imagined self-importance, the delusion that we have some privileged position in the Universe, are challenged by this point of pale light. Our planet is a lonely speck in the great enveloping cosmic dark. In our obscurity, in all this vastness, there is no hint that help will come from elsewhere to save us from ourselves."

Pale Blue Dot: A Vision of the Human Future in Space - Carl Sagan, 1994

ABSTRACT

RIBEIRO, E. G. **A Deep Learning Approach to Visual Servo Control and Grasp Detection for Autonomous Robotic Manipulation**. 2020. 133p. Master Thesis - São Carlos School of Engineering, University of São Paulo, São Carlos, 2020.

The development of the robotics and artificial intelligence fields has not yet allowed robots to execute, with dexterity, simple actions performed by humans. One of them is the grasping of objects by robotic manipulators. Aiming to explore the use of deep learning algorithms, specifically Convolutional Neural Networks, to approach the robotic grasping problem, this work addresses the visual perception phase involved in the task. That is, the processing of visual data to obtain the location of the object to be grasped, its pose and the points at which the robot's grippers must make contact to ensure a stable grasp. For this, the dataset *Cornell Grasping* is used to train a convolutional neural network capable of considering these three stages simultaneously. In other words, having an image of the robot's workspace, containing a certain object, the network predicts a grasp rectangle that symbolizes the position, orientation and opening of the robot's parallel grippers in the instant before its closing. In addition to this network, capable of processing images in real-time, another network is designed so that it is possible to deal with situations in which the object moves in the environment. In this way, the second convolutional network is trained to perform a visual servo control which ensures that the object remains in the robot's field of view. This network predicts the proportional values of the linear and angular velocities that the camera must have so that the object is always in the image processed by the grasp network. The dataset used for training was generated, with reduced human supervision, by a Kinova Gen3 robotic manipulator with seven degrees of freedom. The robot is also used to evaluate the applicability in real-time and obtain practical results from the designed algorithms. In addition, the offline results obtained through validation sets are also analyzed and discussed taking into account their efficiency and processing speed. The results for grasping exceed 90% accuracy with state-of-the-art prediction speed. Regarding visual servoing, one of the designed models achieves millimeter positioning accuracy for a first-seen object. In a small evaluation, the complete system performed successful tracking and grasping of first-seen dynamic objects in 85% of attempts. So, this work presents a new system for autonomous robotic manipulation, able to generalize to different objects and with high processing speed, which allows its application in real-time and real-world robotic systems.

Keywords: Robotic Grasping, Visual Servoing, Deep Learning.

RESUMO

RIBEIRO, E. G. **Uma Abordagem Baseada em Aprendizagem Profunda para Controle Servo-Visual e Detecção de Pontos de Preensão para Manipulação Robótica Autônoma**. 2020. 133p. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2020.

A evolução dos campos da Robótica e da Inteligência Artificial ainda não possibilitou que tarefas simples executadas pelo ser humano, sejam executadas com destreza por um robô. Uma delas é a manipulação de objetos por manipuladores robóticos. Visando explorar o uso de algoritmos de aprendizagem profunda, especificamente Redes Neurais Convolucionais, para abordar o problema de preensão robótica, este trabalho explora a fase de percepção visual envolvida na tarefa. Isto é, o processamento de dados visuais para que se possa obter a localização do objeto a ser pego, sua postura e os pontos nos quais as garras do robô devem fazer contato para garantir uma preensão estável. Para tal, o conjunto de dados *Cornell Grasping* foi utilizado para treinar uma rede neural convolucional capaz de considerar estas três etapas de forma simultânea. Ou seja, de posse de uma imagem do ambiente de trabalho do robô, contendo determinado objeto, a rede prediz um retângulo de preensão que simboliza a posição, orientação e abertura da garra paralela do robô no instante anterior ao seu fechamento. Em adição a esta rede, capaz de processar as imagens em tempo real, outra rede foi projetada para que seja possível lidar com situações em que o objeto se movimenta no ambiente. Desta forma, a segunda rede convolucional é treinada para realizar um controle servo-visual que assegura a permanência do objeto no campo de visão do robô. Esta rede prediz os valores proporcionais das velocidades linear e angular que a câmera deve possuir para que o objeto sempre esteja na imagem processada pela rede de preensão. O conjunto de dados utilizado para treinamento foi gerado, com reduzida supervisão humana, por um robô manipulador Kinova Gen3 com sete graus de liberdade. O robô também foi utilizado para avaliar a aplicabilidade em tempo real e obtenção de resultados práticos dos algoritmos projetados. Os resultados de preensão alcançam 90% de precisão com alta velocidade de processamento. Um dos modelos projetados para controle servo-visual alcança precisão milimétrica de posicionamento para um objeto visto pela primeira vez. Em uma pequena avaliação, o sistema completo executou o rastreamento e a preensão de objetos dinâmicos vistos pela primeira vez em 85% das tentativas. Assim, este trabalho apresenta um novo sistema de manipulação robótica autônoma, capaz de generalizar para diferentes objetos e com alta velocidade de processamento, o que permite sua aplicação em sistemas robóticos de tempo real.

Palavras-chave: Preensão Robótica, Controle Servo-Visual, Aprendizagem Profunda.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

Adam	Adaptive Moment Estimation
AGN	Attention Grasping Network
AI	Artificial Intelligence
ANN	Artificial Neural Network
BN	Batch Normalization
CGD	Cornell Grasping Dataset
CNN	Convolutional Neural Network
CS	Coordinate System
DH	Denavit-Hartenberg
DMP	Dynamic Movement Primitive
DoF	Degrees of Freedom
DVS	Direct Visual Servoing
FC	Fully Connected
FM	Feature Map
fps	frames per second
GD	Gradient Descent
GG-CNN	Generative Grasping CNN
GPU	Graphics Processing Unit
HCF	Hierarchical Cascaded Forests
IBVS	Image-Based Visual Servoing
IW	Image-Wise
ML	Machine Learning
MLP	Multi-layer Perceptron
MSE	Mean Squared Error

Nadam	Nesterov-accelerated Adaptive Moment Estimation
NM	Nesterov Momentum
OW	Object-Wise
PBVS	Position-Based Visual Servoing
RANSAC	Random Sample Consensus
R-CNN	Region-based CNN
ReLU	Rectified Linear Unit
RGB-D	Red, Green, Blue - Depth
RL	Reinforcement Learning
RoI	Regions of Interest
SD	Standard Deviation
SGD	Stochastic Gradient Descent
VS	Visual Servoing

LIST OF SYMBOLS

$\mathbb{R}$	Real number set
W	Network parameters (weights)
η	Learning rate of the gradient descent optimization algorithm
∇	Gradient operator
σ	Fraction of the update vector for previous iteration to be added in the current iteration of SGD
ν	Momentum term
m	Average with exponential decay of past gradients
β_1, β_2	Regulation terms for Adam's average terms
ϵ	Regulation term for Adam's update rule
a_i	Distance between the links z_i e z_{i+1} measured on the axis x_i
α_i	Angle between the links z_i e z_{i+1} measured on the axis z_i
d_i	Distance between the links x_{i-1} e x_i measured on the axis z_i
θ_i	Angle between the links x_{i-1} e x_i measured on the axis z_{i-1}
Π	Product operator
α	Rotation around x-axis
β	Rotation around y-axis
γ	Rotation around z-axis
$\mathbf{q}_i$	Joint positions
$\dot{x}$	Time derivative of x
J	Jacobian matrix
$\vec{F_i}$	Force vector in grasping
(x_c, y_c)	Coordinates of the grasp rectangle center point
w	Width of the grasp rectangle - gripper opening

h	Height of the grasp rectangle - gripper size (constant)
θ	Orientation of the grasp rectangle
(o_x, o_y)	Principal point
(s_x, s_y)	Effective size of the image pixels
X_w	Coordinates of a point in the world reference frame
f	Focal length
M_{int}, M_{ext}	Intrinsic and extrinsic matrices
(x_{im}, y_{im})	Point expressed in image coordinates (pixels)
$J(r_p, r_c)$	Jaccard index between rectangles
${}^{A}\mathbf{T}_{B}$	Homogeneous transformation that express the origin of the frame $\{B\}$ relative to frame $\{A\}$
λ	Proportional gain for the visual servoing control
$\mathbf{L}_s$	Interaction matrix
v_c	Camera velocity
$\mathbf{e}(t)$	Error between feature vectors as a function of time
$\mathbf{s}$	Feature vector
$\mathbf{s}^*$	Desired feature vector
X^+	Moore-Penrose pseudo-inverse of X
$tr(R)$	Trace of matrix R
$\theta\mathbf{u}$	Angle-axis representation of a 3D rotation
$\mathcal{L}$	Candidate Lyapunov function

CONTENTS

1 INTRODUCTION

Automatic systems are those that do not require human intervention for their usual operation (AGUIRRE et al., 2007), or that work for themselves, dispensing operators (HOUAISS, 2001). However, taking into account philosophical discussions about what knowledge is, Habermas (2014) says that from a pragmatic point of view, the process of knowledge is represented by intelligent behavior that solves problems. Therefore, for automatic machines to handle unforeseen situations in their programming, and become, in fact, intelligent, artificial intelligence techniques can be used. In this way, detection, classification, and pattern recognition tools are key requirements for systems to make decisions. These techniques can be effective in many practical applications, gaining more space in the area of robotics, since robots are increasingly able to sense the environment and can, therefore, use this sensory information intelligently.

Object grasping is a canonical area of research in robotic manipulation. However, its relevance and importance extend to the present day, since it is regarded as an open problem of robotics, as can be noted in the lack of dexterity with which robots manipulate complex objects. Early robotic grasping research fits into model-based experiments in which the models of objects to be grasped should be known. Using these models, it is possible to find points in the object that configure good grasp choices - points at which grippers must make contact with the object, ensuring that the action of external forces does not lead it to instability - by means of analytic solutions.

Subsequent research, leveraged mainly by the improvement of data-based methods, began to focus on automatic detection of grasp points using datasets. But it still relied on 3D models of objects, often sustained in simulations. Aiming at the independence of a priori knowledge of object-associated models, recent research has sought to recognize grasp points only by visual data. These works tend to search for features in the image using image processing techniques and use these features as input to classical machine learning algorithms.

In this scenario, great assistance can be provided by deep learning tools, as this can automatically extract relevant features from the visual data captured by the robot, and use them to infer ideal points for grasping. Much of the research developed in grasp detection today is based, at least in part, on deep learning algorithms, such as convolutional neural networks and deep reinforcement learning.

In addition to the task of grasp detection, a grasping system should be able to adapt to changes in the workspace, be robust to inaccuracies in kinematic modeling and consider dynamic objects, using visual feedback. Thus, the robot must track the object so

that it does not leave the camera's field of view and the grasp detection be reactive to changes in its pose. This can be done using visual servoing, a control algorithm that uses visual information in the feedback so that the image acquired by the camera approaches a reference image, that is used for grasp detection.

Visual servoing is also based on the extraction and tracking of features, often based on 3D models, camera parameters and other information that must be known a priori. However, recently, techniques that aim to minimize the work of feature design, control modeling, and the need for a priori information, are being developed. Current approaches, however, fail to achieve all of these advantages simultaneously. Then, taking advantage of the great representation learning ability of deep learning, all of these requirements can be satisfied at once and, moreover, the learned controller can generalize for different situations.

A grasping system that can combine the benefits of automatic grasp detection and that receives visual feedback to deal with unknown dynamic objects, in real-time, is one of the goals in robotics. This work, therefore, aims to advance towards this goal by designing a real-time, real-world, and reactive autonomous robotic manipulation system.

1.1 General objective

The general objective of this work is to apply Convolutional Neural Networks (CNN) to the problem of autonomous robotic manipulation, using only visual data.

1.2 Specific objectives

- Design a fast CNN for grasp detection of different objects.
- Design a fast CNN which can execute visual servoing in different target objects by generating a proportional velocity signal based only on the reference image and the current image.
- Encapsulate the two CNNs in a single grasping system.
- Implement the proposed algorithms in the 7 DoF KINOVA Gen 3 robotic manipulator for real-world validation.

1.3 Work organization

This dissertation is structured as follows: Chapter 2 brings the theoretical foundations of the techniques and algorithms used in this work. Chapter 3 presents the literature review containing related work and the state-of-the-art. Chapter 4 describes the methodology addressed to solving the problem and Chapter 5 presents the achieved results. Finally, in Chapter 6, the conclusions and future work proposals are presented.

2 THEORETICAL BACKGROUND

2.1 Deep Learning

Deep learning is a field of artificial intelligence. Specifically, it is a particular type of machine learning that achieves great power and flexibility of representation when approaching the world as an increasing hierarchy of simple concepts and with higher levels of abstraction (GOODFELLOW et al., 2016).

Conventional machine learning techniques require careful engineering and great technical mastery to build a feature extractor (LECUN; BENGIO; HINTON, 2015). This step must be done before making the data available to the intelligent system, usually a classifier, since it is unable to learn representations through the raw data, *e. g.* images (LECUN; BENGIO; HINTON, 2015).

Representation learning is the set of methods that automate the representation of raw data so that it can be delivered directly to the intelligent system (LECUN; BENGIO; HINTON, 2015). In this sense, deep learning can be defined as the set of representation learning methods (LECUN; BENGIO; HINTON, 2015) that have multiple levels of abstraction, as defined by Goodfellow et al. (2016). Through the composition of these transformations, notably complex functions can be learned.

Because of this great capacity of feature extraction from data, deep learning is now a state-of-the-art technique in some applications, such as computer vision (GUO et al., 2016; Islam et al., 2016) and some fields of robotics (LEVINE et al., 2016a; SCHMIDT et al., 2018), and a viable alternative in several others, such as chemistry (GOH; HODAS; VISHNU, 2017), virtual security (MOHAMMED; VINAYAKUMAR; SOMAN, 2018) and medicine (LITJENS et al., 2017). A review of the applications of deep learning can be found at Liu et al. (2017). In the next section, some fundamental concepts are quickly introduced so that the particularities involved in deep neural networks are better understood.

2.1.1 Machine Learning, Artificial Neural Networks and Neocognitron

Activities such as writing computer programs, solving mathematical equations or understanding a language, demand “intelligence” (NILSSON, 2014). In the last decades, some of these activities started to be solved by computer systems. It can be said that such systems, for performing tasks that demand natural intelligence from human beings, have “artificial intelligence” (NILSSON, 2014).

According to Russell and Norvig (2016), the term was coined in 1956, although it has been the subject of studies at least five years earlier. Kaplan and Haenlein (2019, p. 17) define Artificial Intelligence (AI) as the “system’s ability to interpret external data

correctly, to learn from such data, and to use those learnings to achieve specific goals [. . .]".

That is, AI aims to explain and emulate intelligent behavior through computational processes (SCHALKOFF, 1990). In this way, several fields of study can be categorized as belonging to AI: reasoning, knowledge representation (learning of representation), planning, perception, object manipulation and machine learning, which includes deep learning, responsible for a major advance in the ability to learn multidimensional data.

Machine learning (ML) can be defined as the set of methods that can detect patterns in data automatically, and use these patterns to predict future data, or perform other types of decision making through uncertainties (ROBERT, 2014). This search for patterns in data is an old and fundamental problem (BISHOP, 2006) and can be classified into *Supervised*, *Unsupervised* or *Reinforcement learning*.

In supervised learning problems, the following formal model leads to learning: the agent has access to the *space of instances* and *space of labels*, which constitute the *training set* . Through these, it must generate an *output function*, which can be tested using a *test set*. The space of instances is an arbitrary X set of items to be labeled. Usually, this set is represented by a feature vector (SHALEV-SHWARTZ; BEN-DAVID, 2014).

Cases in which the objective is to categorize each input vector into a finite number of discrete categories, are called *classification* problems (BISHOP, 2006). In this case, the Y label space is represented by $\{0, 1, .., n\}$, where n represents the total of categories. If the desired output is formed by continuous variables, then the problem is called *regression* (BISHOP, 2006). In this case, the Y label space is represented by $\{x_1, x_2, .., x_n\}$, where x_i belongs to the set $\mathbb{R}$ and n is the number of variables to be predicted.

The training set $S = ((x_1, y_1), ..., (x_m, y_m))$ is a finite sequence of labeled pairs X, Y. This is the entry that the agent will have access to and will conduct its learning. The agent should obtain an output function $h : X \to Y$ as a result. This function is also called a predictor, hypothesis or classifier. It will be used to predict the label of new instances (SHALEV-SHWARTZ; BEN-DAVID, 2014).

Lastly, the test set $T = ((x_1, y_1), ..., (x_n, y_n))$ will be used to analyze the generalization of the obtained output function. For this purpose, only the instances (X) are delivered to the agent and the labels produced will be compared to the labels of the set Y. Through this set, we have the performance of the agent for new data.

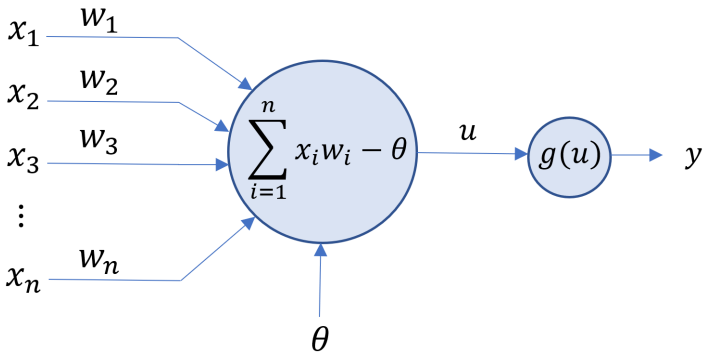
In unsupervised pattern recognition problems, the X instance set exists, but the Y label set is not available. The objective of these problems may be the discovery of groups with similar characteristics within the data set (*grouping*), determining the distribution of data in the input space, among others (BISHOP, 2006). Finally, the reinforcement learning technique is related to the search for appropriate actions to be taken in order

to maximize a reward (BISHOP, 2006). In this case, there is no training set of the type $((x_1, y_1), ..., (x_m, y_m))$, but this must be found by trial and error.

Among the knowledge representation bases in the field of ML, the most prominent are the Artificial Neural Networks (ANN). These were inspired by the functioning of the human brain and the way it conducts its learning mechanisms. The main characteristics of the ANNs, according to Silva et al. (2017), are adaptation by experience, learning capacity, generalization skills, data organization, distributed storage, fault tolerance and prototyping facility. Potential applications include function approximation, process control, pattern classification, data clustering, forecasting systems, among others.

Since artificial neural networks mimic the brain's information flow process, its basic morphological structure, the artificial neuron, attempts to reproduce the functioning of the biological neuron. Although these cells are extremely complex, their computational essence in terms of inputs and outputs is relatively straightforward. The artificial neuron model, called perceptron, was developed by Rosenblatt (1958) from the primordial model created by McCulloch and Pitts (1943). Its structure is illustrated in Fig. 1.

Figure 1: Perceptron



Source: Author

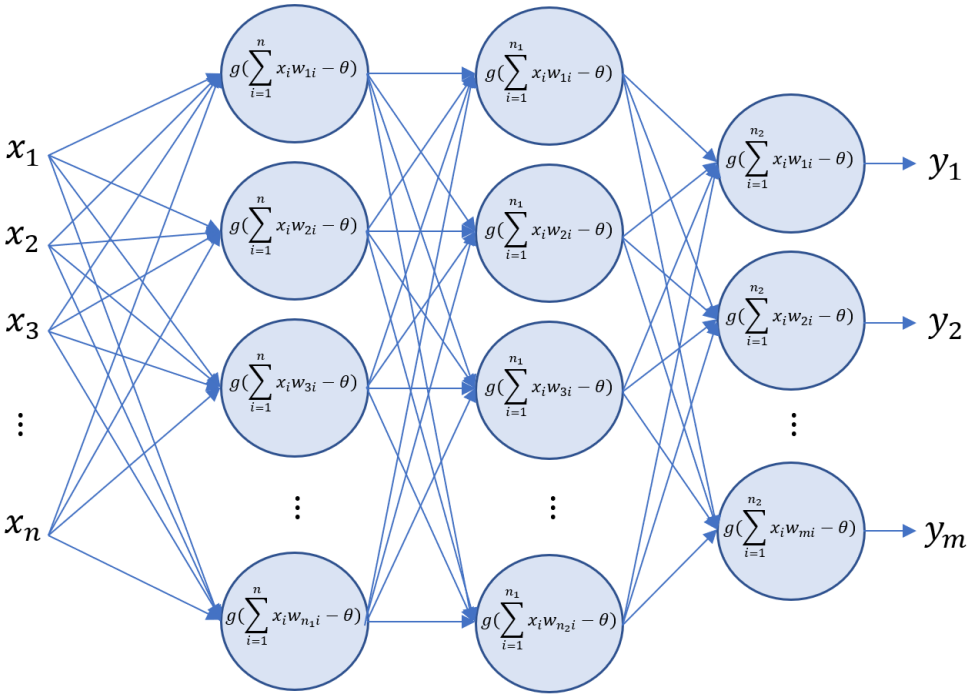
The perceptron processes a set of input variables $\{x_1, x_2, ..., x_n\}$ multiplying each one by its respective synaptic weight $\{w_1, w_2, ..., w_n\}$. This result is subtracted from an activation threshold θ, producing an activation potential u (SILVA et al., 2017). After obtaining u, an activation function g must be applied in order to limit the neuron output.

The Rectified Linear Unit (ReLU) function is the main activation function used in deep networks, as its piecewise structure considerably improves the performance of the network and allows the development of much deeper models (GOODFELLOW et al., 2016). They also preserve many properties of linear functions that facilitate the gradient-based optimization process and ensure good generalization. Its operation is limited to repeating the input, if it is > 0, and returning 0, if it is ≤ 0, as described in Eq. 2.1.

$$g(u) = max(0, u) \tag{2.1}$$

Multi-Layer Perceptron (MLP) networks can represent different types of functions and can be used in nonlinear classification problems, among many other applications. The greater number of parameters to be trained makes the network capable of knowledge abstractions far superior to those of the perceptron networks. As the name demonstrates, MLP networks are made up of more than one layer of perceptrons. Fig. 2 illustrates an MLP with its input and output layers and two more hidden layers.

Figure 2: Common structure of an MLP with two hidden layers



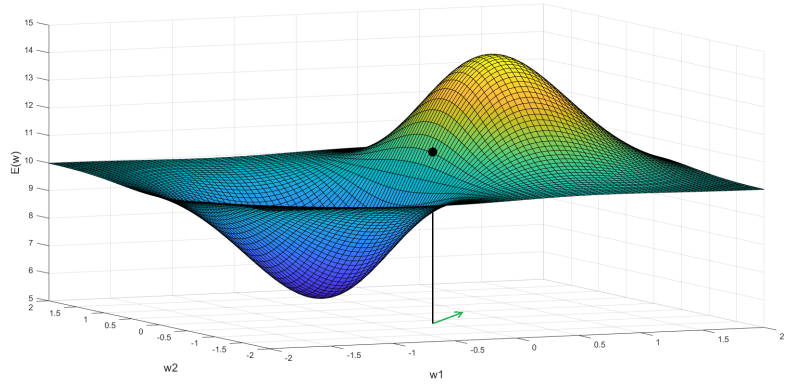
Source: Author

The network training error is a function of its synaptic weights. Therefore, a way of adjusting these parameters is to search for those that correspond to the minimum value of this error function.

The method used to pursue this minimum is based on the gradient operator. Since the gradient is a vector that points to the direction of the greatest variation of a function, following the opposite direction of this gradient makes it possible to reach the minimum point of that function. Fig. 3 shows the error as a function of just two weights and shows the green gradient vector. Note how going in the opposite direction of this gradient leads to the global minimum of the function. The backpropagation algorithm takes this idea into account to find the set of network parameters that minimize the training error. The development of backpropagation is detailed in Appendix A.

One of the models of neural networks that contributed most to the development of convolutional networks was developed by Fukushima (1975) and later perfected by the author in 1980 (FUKUSHIMA, 1980). These models, called Cognitron and Neocognitron,

Figure 3: Error function highlighting the direction of the gradient at one point



Source: Author

respectively, were designed to recognize visual patterns in images. The Neocognitron network, unlike its predecessor, was able to recognize patterns even if they were in different positions in the input image, using the concepts of simple and complex cells from the experiment of Hubel and Wiesel (1962).

Studying the receptive field of cells in the striated cortex of cats and monkeys, the researchers showed that individual neurons were activated only in the presence of specific patterns, such as edges in specific orientations. They also proposed a connection model with increasing order of specificity of the receptive fields, so that the cells of the lateral geniculate nucleus would pass information to simple cells, and from these to complex cells.

These concepts of hierarchical network and specialized components are also found in today's convolutional networks. However, the great difference in relation to the Neocognitron models is the backpropagation training process, which enabled the development of the first convolutional neural network, LeNet-5, by LeCun et al. (1998).

For applications with images, convolutional neural networks have achieved outstanding results in pattern recognition and detection tasks. Driven by the algorithm developed by Krizhevsky, Sutskever and Hinton (2012), for the *ImageNet* competition (DENG et al., 2009), these networks have drawn the attention of the Artificial Intelligence community to the area of deep learning.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (LECUN et al., 1989) are a specialized type of neural network capable of processing data that has a spatial structure, such as images. From the biological point of view, the CNNs were inspired specifically by the visual cortex, which has small regions with specialized cells in each area of the visual field. The idea of specialized components found in the Neocognitron can be found in the rationale for the CNNs. Simplistically, CNNs are neural networks that use the convolution operation in

place of the general matrix multiplication in at least one of its layers (GOODFELLOW et al., 2016). In general, convolution is an operation between two real functions. Its formal definition for the discrete case is given by

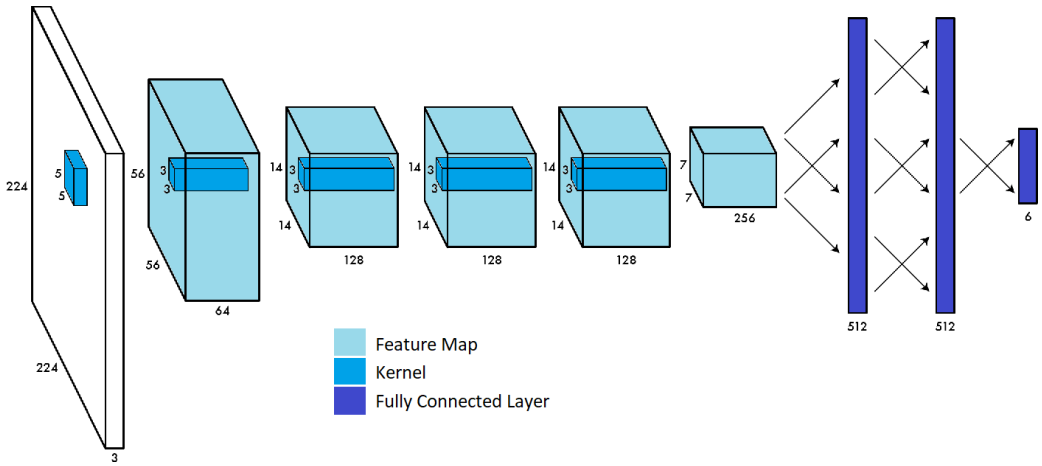
$$y[n] = h[n] * x[n] = \sum_{m=-\infty}^{\infty} x[n-m]h[m]. \tag{2.2}$$

In the context of convolutional networks, the first term of the Eq. 2.2 is called *input* and the second is called *kernel*. The output is usually called a *feature map* (GOODFELLOW et al., 2016). In machine learning it is common to deal with multidimensional data, that is, the input is usually a multidimensional vector (tensor) of data and the kernel is a tensor of parameters that must be adapted by the algorithm (GOODFELLOW et al., 2016). The infinite sum can be done under a finite number of elements since it is assumed that the functions involved in the convolution have a non-zero value only at the points where these elements were stored. In this way, Eq. 2.2 can be rewritten considering the convolution of a two-dimensional image I as input, with a kernel K, also two-dimensional, as Eq. 2.3.

$$S[i,j] = K[i,j] * I[i,j] = \sum_{m}\sum_{n} I[i-m, j-n]K[m,n] \tag{2.3}$$

Despite being made up of an infinity of representations, CNNs are usually built from two sets of layers: convolutional and fully connected. Fig. 4 exemplifies this CNN. The figure shows an image as input, with dimensions 224×224 and 3 channels (RGB). This image is processed with a convolution filter with a 5×5 kernel and depth 3, since the depths of the input and the kernel must be identical.

Figure 4: Usual Convolucional Neural Network with Fully Connected Layers



Source: Adapted from Redmon and Angelova (2015)

A Feature Map (FM) of dimensions $56 \times 56 \times 64$ is obtained as output since the convolution operation reduces the dimensions of the image and increases the depth of the FM. The depth increases because is recommended to use several convolution filters to

generate more FMs, which increases the learning capacity of the network. These operations are repeated many times (in Fig. 4 there are 5 convolutional layers) and other types of operations can be interspersed with the convolution in these layers, such as *pooling*, *batch normalization*, among others, as will be seen below.

At the end of the last convolutional layer, a $7 \times 7 \times 256$ FM is obtained. This FM is flattened so that it can be delivered to Fully Connected (FC) layers, assuming the dimension 1×12544. The operations that take place at the FC layers are identical to those of an MLP. In general, a CNN can be understood as the conjunction of convolutional layers, which act in the automatic extraction of relevant features of the image and fully connected layers, which use these characteristics to generate a classification or regression law.

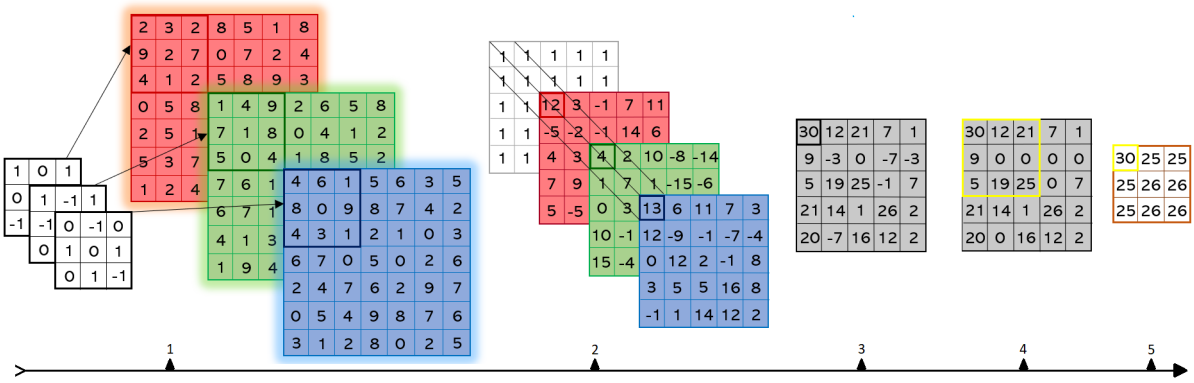
In addition to the convolution itself, a convolutional layer may contain other types of operations that modify the dimensions of the resulting FM. The pooling operation, for example, helps to make representations invariant to small translations of the input. This property can be useful in situations where the occurrence of a feature is more important than its location (GOODFELLOW et al., 2016). The computational cost is also balanced with the insertion of pooling, because, as the depth of the tensors tends to increase after the convolutional layers, it is convenient to reduce their spatial dimension (PONTI; COSTA, 2018).

Like a convolution filter, the pooling operation also has a spatial dimension and is associated with a number of steps. The most traditional form of this operation is max pooling (ZHOU; CHELLAPPA, 1988), in which the maximum value of a window is the only one kept.

The rectified linear unit (JARRETT et al., 2009), defined earlier, is the current recommendation of activation function in CNNs (GOODFELLOW et al., 2016). It has two main advantages. First, it decreases the network training time, since sigmoidal functions saturate, whereas ReLU is the identity function for positive values (PONTI; COSTA, 2018). Another key point is the fact that ReLU avoids the problem of vanishing gradient, that is, the gradient does not become zero during optimization, as with sigmoidal functions.

Fig. 5 schematically explains the operations involved in a convolutional layer, showing the convolution and pooling operations. Step 1 shows each channel of the input image and its respective 3×3 kernel. The result of the element-wise multiplication, after the kernel slide over the image, is shown in step 2. In step 3, we have the result of the sum of the 3 channels and a bias matrix. Subsequently, in step 4, there is the application of ReLU and the indication of a 3×3 max-pooling filter. The result, when the filter slides with jumps of 2 pixels, is shown in step 5, which also represents the resulting FM from the convolutional layer. As only 1 convolution filter is used in the layer, only one FM is obtained at the output.

Figure 5: All operations performed on a simple convolutional layer with max pooling

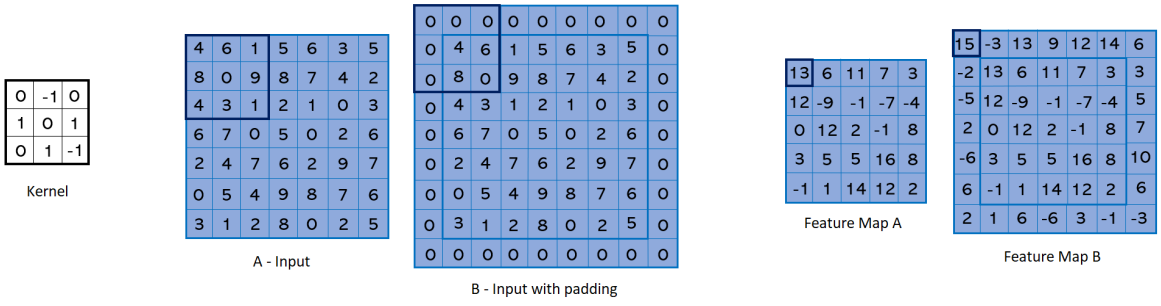


Source: Author

Another operation that can change the FM in a convolutional layer is padding. Through this technique, it is possible to control the size of the convolution kernel and the dimensions of the output independently. The padding is done with zeros (zero-padding) on the borders, making the dimensions of an FM $n \times n$ become $(n+2) \times (n+2)$ in the case of padding of just one line.

Fig. 6 demonstrates the idea for a 7×7 FM and a 3×3 kernel, without zero-padding in *A-Input*, resulting in the 5×5 *Feature Map A*. On the other hand, the same input with zero-padding, represented by *B - Input with padding*, results in the *Feature Map B*, with the same dimensions of the input (7×7). Padding allows the inclusion of several convolutional layers without reducing the FM dimension, making possible that more features be learned. However, this procedure should be done with caution, since relevant features that may be at the edges of the image are lost after convolution with many zeros (GOODFELLOW et al., 2016).

Figure 6: Zero-padding demonstration



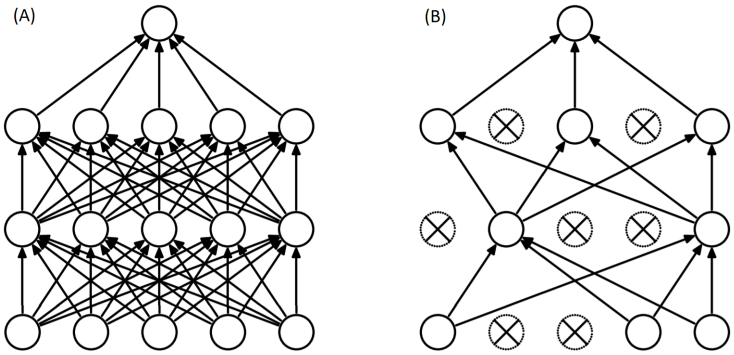
Source: Author

In addition to techniques that enable greater control and efficiency of the convolutional layers, other tools improve the performance of the FC layers, as well as the network as a whole. Dropout is one of those tools used to minimize overfitting, proposed

by Srivastava et al. (2014). The procedure consists of deactivating, with a probability p, the activation of neurons of the fully connected layers in the training phase. In the test phase, activations are reallocated with the same probability p, to compensate those that were disconnected in the training (PONTI; COSTA, 2018).

Fig. 7 illustrates the technique operation, in which (A) presents a traditional FC layer and (B) shows the case of an FC layer with dropout. By deactivating some neurons, the complexity of the functions is reduced since fewer activations are used. Besides, dropout can also work as a form of ensemble bagging, as shown by Goodfellow et al. (2016), because all possible networks to be formed from the withdrawal of neurons are trained. In other words, taking as an example the k-fold cross-validation technique, k networks are trained and the best one is selected by the method, whereas with dropout, a much larger number of networks are trained at a low computational cost.

Figure 7: Dropout in fully connected layers of a neural network



Source: Srivastava et al. (2014)

Batch normalization (BN) is another tool used to improve performance and also increase the stability of neural networks. The idea is to normalize the inputs of each layer in such a way that they have zero mean and unit variance. Through BN, networks train faster because, although the additional calculations of normalization slow down each iteration, they converge in a shorter time. It also makes it possible to use higher learning rates, another factor that reduces the convergence time, because, since gradients become smaller during backpropagation, higher learning rates increase training speed. BN is also able to reduce the network's sensitivity to its weights initialization, as described by Ioffe and Szegedy (2015).

One of the main training components of a convolutional neural network is its optimization algorithm since the gradient needs to be adjusted to be propagated across the entire deep network. As the algorithms used in deep learning are derived from the Stochastic Gradient Descent (SGD), following are some demonstrations of the adjustments made in the SGD to develop the optimization algorithms *Adam* (Adaptive Moment Estimation) (KINGMA; BA, 2014) and *Nadam* (Nesterov-accelerated Adaptive Moment Estimation)

(DOZAT, 2016; SUTSKEVER et al., 2013), used in this work. As a starting point, the weight adjustment law derived in Appendix A (Eq. A.7) is rewritten in a matrix and iterative way, and is given by:

$$W_{t+1} = W_t - \eta \nabla E(W). \tag{2.4}$$

SGD does not converge easily in regions close to local optima, oscillating considerably before reaching the optimum point. Momentum (POLYAK, 1964; QIAN, 1999) is a method used to accelerate the convergence of SGD by adding a σ fraction of the update vector from the previous iteration to the current one. This is done by using a term called momentum (ν_t), given in Eq. 2.5, so that the term is increased in dimensions whose gradients point in the same directions as the previous iteration and reduced in dimensions in which the gradients change direction. In this way, convergence happens faster since the oscillations are dampened (RUDER, 2016).

$$\nu_t = \sigma \nu_{t-1} + \eta \nabla E(W) \tag{2.5}$$

Nesterov Momentum (NESTEROV, 1983) is a way of giving the momentum term a form of anticipation. The gradient for this case is calculated based on an idea of where the weights will be in the future iteration. This prevents the algorithm from taking very large steps in updating the weights, which results in faster convergence. The momentum term for Nesterov's accelerated case is given by Eq. 2.6.

$$\nu_t = \sigma \nu_{t-1} + \eta \nabla E(W - \sigma \nu_{t-1}) \tag{2.6}$$

For both types of momentum, σ assumes values close to 0.9 and the update of the weights follows the rule expressed in Eq. 2.7.

$$W_{t+1} = W_t - \nu_t \tag{2.7}$$

Adam (Adaptive Moment Estimation) (KINGMA; BA, 2014) is a method that calculates an adaptive learning rate for each of the weights in the network. It does so by storing the average with exponential decay of the past gradients m_t, as well as the momentum (RUDER, 2016), and the average with exponential decay of the square of the past gradients (ν_t), just as Adadelta algorithm (ZEILER, 2012). These terms are given by Eqs. 2.8 and 2.9 respectively.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla E(W) \tag{2.8}$$

$$\nu_t = \beta_2 \nu_{t-1} + (1 - \beta_2)(\nabla E(W))^2 \tag{2.9}$$

The values of m_t and ν_t are initialized to zero, which makes these parameters very close to this value, especially in the initial steps. The authors then propose their corrected versions, given by Eqs. 2.10 and 2.11, where $\beta_1 = 0.9$, $\beta_2 = 0.999$.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.10}$$

$$\hat{\nu}_t = \frac{\nu_t}{1 - \beta_2^t}. \tag{2.11}$$

The updating of the weights is given by Eq. 2.12, where $\epsilon = 10^{-8}$. Kingma and Ba (2014) proves empirically in their work that Adam works well in practice and has better performance than other adaptive learning algorithms.

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{\nu}_t} + \epsilon} \hat{m}_t \tag{2.12}$$

Nadam (Nesterov-accelerated Adaptive Moment Estimation) (DOZAT, 2016), (SUTSKEVER et al., 2013) is basically the Adam algorithm with the Nesterov momentum. The first step in reaching Nadam's weight update rule is to expand Eq. 2.12 from the definitions 2.8 and 2.10, resulting in Eq. 2.13.

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{\nu}_t} + \epsilon} \left(\beta_1 \hat{m}_{t-1} + \frac{(1 - \beta_1)\nabla E(W)}{1 - \beta_1^t} \right) \tag{2.13}$$

The only change in the previous equation comes from Nesterov Momentum (NM) insertion strategy into the Adam algorithm, produced by Dozat (2016). In this work, after a change in the NM algorithm, the author came up with a way to optimize the Adam algorithm so that the value of the corrected momentum of the previous iteration is used to update only the current momentum, and not the gradients. So, the anticipation provided by the NM is applied, resulting in faster convergence. Placed in equation, the Nadam update rule is given by Eq. 2.13 with $\hat{m}_{t-1}$ replaced by $\hat{m}_t$, as Eq. 2.14 shows.

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{\nu}_t} + \epsilon} \left(\beta_1 \hat{m}_t + \frac{(1 - \beta_1)\nabla E(W)}{1 - \beta_1^t} \right) \tag{2.14}$$

For the implementation of deep learning algorithms is common to use development frameworks with specific and optimized tools that facilitate their design. The most used frameworks are *Tensorflow* (ABADI et al., 2016), *Keras* (CHOLLET et al., 2015), *Theano* (BERGSTRA et al., 2010) and *Caffe* (JIA et al., 2014).

Considered the most chosen option among developers, *Tensorflow* performs operations in the form of graphs that represents the flow of multidimensional data (tensors). On the other hand, *Keras* can execute commands from *Tensorflow* at a higher level, allowing the use of pre-developed training blocks, activation functions, convolutional layers, among others.

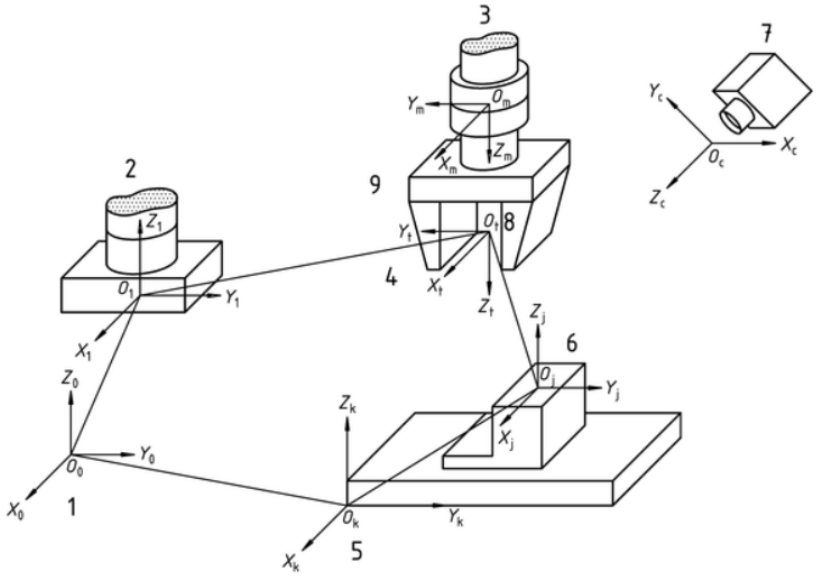
2.2 Robotic Manipulators

Robotic manipulators are kinematic chains composed of links and joints. These joints can be typically rotational or prismatic and are responsible for connecting two links i and $i+1$. The rotation axis of a rotational joint, or under which a prismatic joint moves, is called z_i and connects the links i and $i+1$. The joint variables can be called θ_i, for a rotational joint, or d_i, for a prismatic one, and represent the relative displacement between adjacent links (SPONG et al., 2006).

By associating frames to each item of a robotic system and specifying the geometric relationships between each of them, it is possible to obtain an accurate description of the position and orientation of the items arranged in the workspace, as well as to solve the problem of the manipulator's kinematics. Appendix B presents some description of spatial transformations between frames since this is necessary for the study of a manipulator and to understand the developed visual servoing.

Fig. 8 illustrates some associated frames in the workspace of a robot: {1}-World frame, {2}-Base frame, {3}-Mechanical interface frame, {4}-Tool frame, {5}-Task frame, {6}-Object frame, and {7}-Camera frame.

Figure 8: Associated coordinate system to each element of the robotic workspace



Source: International Organization for Standardization (2000)

The kinematics of a manipulator is the study of the relationships between the positions, speeds, and accelerations of its links. The forward kinematics of position is concerned with finding the relationship between the base and tool frames, T. To reach this relationship, it is necessary, first, to define coordinate systems associated with each link, assigning frames to them. This assignment can be done according to the rules of the Denavit-Hartenberg (DH) algorithm (SPONG et al., 2006), whose parameters are:

- a_i - distance between the links z_i e z_{i+1} measured on the axis x_i;
- α_i - angle between the links z_i e z_{i+1} measured on the axis z_i;
- d_i - distance between the links x_{i-1} e x_i measured on the axis z_i;
- θ_i - angle between the links x_{i-1} e x_i measured on the axis z_{i-1}.

With these parameters, it is possible to build the homogeneous transformation matrices that define the frame of the joint i with respect to the frame of the joint $i-1$. In this way, the problem of finding the manipulator's kinematics is divided into n sub-problems, where n is the number of joints in the manipulator. Each transformation between consecutive joints is given by the matrix of Eq. 2.15.

$$A_i = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sen\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sen\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

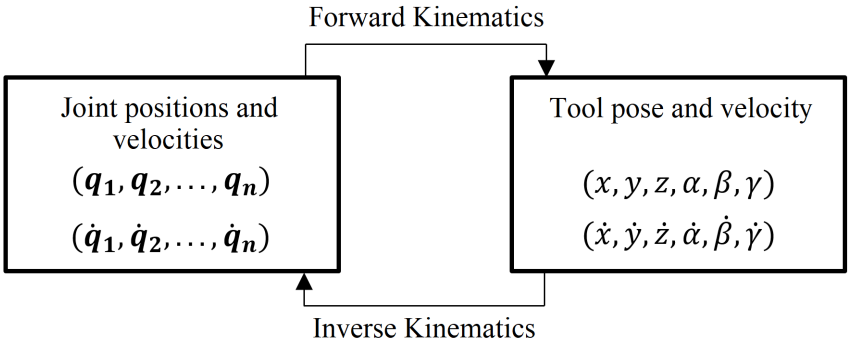
Once the transformation matrices associated with each joint have been found, the T matrix can be calculated by Eq. 2.16.

$$T = \prod_{i=1}^{n} A_i \tag{2.16}$$

The study of a manipulator's kinematics is divided into two problems: the location of the robot tool given the positions of its joints and its dual, that is, determining the positions of the joints given a certain position of the tool. These problems are called forward kinematics and inverse kinematics respectively. In this way, it is possible to define two variable spaces. The joint coordinate space (q), consisting of n dimensions, and the Cartesian space (r), which defines the pose of the tool from 3 positions and 3 orientations. This idea is outlined in Fig. 9.

The forward kinematics results from the development of the manipulator expressions, whereas the inverse kinematics deals with the set of methods to obtain the inverse relation. This is a problem whose analytical solution is not guaranteed and, at times, may not present a solution, or even present several solutions. In this way, it is common that solutions of practical interest be already developed.

Figure 9: Position and velocity kinematics



Source: Author

The relationship between the velocity of a robotic manipulator's joints and the velocity of its tool is given through its Jacobian. As in position kinematics, it is possible to map these velocities in the joint space and the Cartesian space. That is, if the objective is to find the velocity of the robot's tool from the velocity of its joints, the direct Jacobian is used. In order to discover the necessary velocities in each joint to reach a given tool velocity, the inverse Jacobian is used.

The Jacobian matrix is obtained from the differential analysis of the joint and the tool poses (CRAIG, 2009). To find its value it is necessary to have a description of the manipulator according to the parameters q_i, as in Eqs. 2.15 and 2.16. Deriving as a function of time and rewriting in matrix form, the Jacobian matrix that relates the joints and tool velocities is given by Eq. 2.17.

$$\begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix}^T = \begin{bmatrix} \frac{\partial f_x}{\partial q_1} & \cdots & \frac{\partial f_x}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_\gamma}{\partial q_1} & \cdots & \frac{\partial f_\gamma}{\partial q_n} \end{bmatrix} \begin{bmatrix} \dot{q}_1 & \dot{q}_2 & \cdots & \dot{q}_n \end{bmatrix}^T \tag{2.17}$$

For the case in which the velocity of the robot's tool is known, as in visual servoing, one must discover the velocities of the joints that guarantee the necessary velocity in the tool. This is the transformation from Cartesian space to joint space and is given by the inverse Jacobian, as Eq. 2.18 shows.

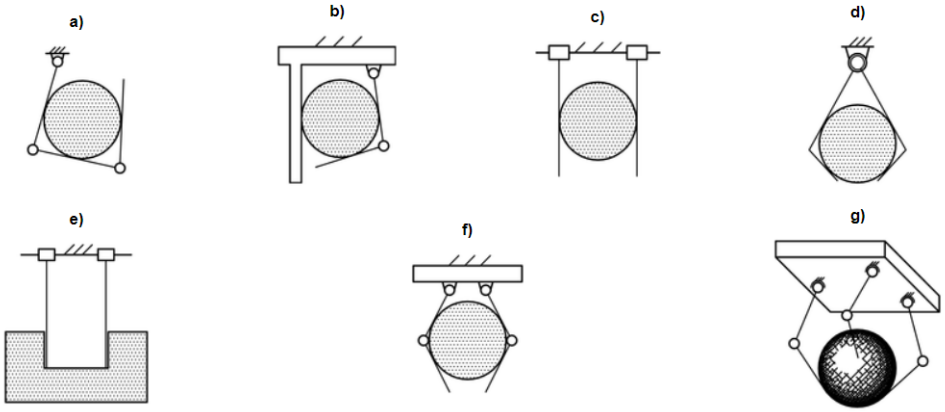
$$\begin{bmatrix} \dot{q}_1 & \dot{q}_2 & \cdots & \dot{q}_n \end{bmatrix}^T = J^{-1} \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix}^T \tag{2.18}$$

2.2.1 Robotic Grasping

Canonical works on robotic grasping (BICCHI; KUMAR, 2000; CUTKOSKY, 2012) present the concept of grasping on robots in a broad, general, or generic way. Others present only the mathematical definition (PRATTICHIZZO; TRINKLE, 2016). Therefore, for a succinct and purely conceptual definition of the term, the ISO standard is used.

According to ISO 14539:2000 (reviewed and confirmed in 2015) (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2000), three definitions that comprise the formal definition of what is considered robotic grasping. First, the word grip refers to the restrictions applied by the robot's end-effector to an object. Later the standard defines grasp as the restrictions applied to an object by means of grippers. And finally, define grasping as the movement of the grippers capable of applying restrictions to an object. For the sake of simplicity, robotic grasping is defined here as the ability of a robot's grippers to keep an object stable, independent of other external forces acting on the object. The meaning of grasp is the same as the standard, as it is considered to be the set of restrictions applied by the robot's grippers to achieve grasping. Some types of grippers are illustrated in Fig. 10.

Figure 10: Examples of grippers used for robotic grasping



Source: International Organization for Standardization (2000)

According to Birglen and Schlicht (2018), pneumatically operated parallel grippers appear as the most common in manufacturers' catalogs. The authors performed an extensive review of these catalogs and obtained a statistical analysis of manufacturers and models, as well as qualitative analysis based on some indexes, such as strength and power. The reasons why parallel grippers are the most commonly used are their ability to handle objects of different shapes and sizes and to efficiently perform the same tasks as other common grippers, such as the three-finger and the angular. In Fig. 10, the parallel grippers are represented by *c*.

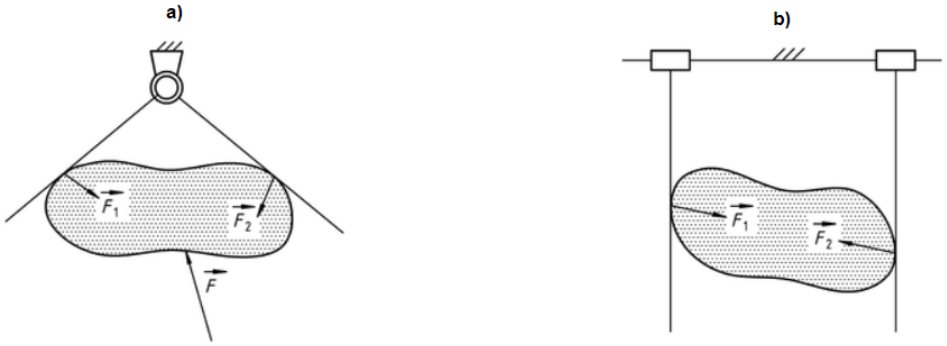
There are three main ways to perform robotic grasping. Two of them are based on *closure* and another on *caging*. Closure strategies aim to apply frictional forces through which the object does not move and caging, on the other hand, aims to just limit the movement of the object (RODRIGUEZ; MASON; FERRY, 2012).

Regarding closure methods, there are two possibilities of synthesis (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2000). *Form Closure Grasp* is a type of grasp in which the degrees of freedom are equal to or less than zero, regardless of

the frictional forces at the contact points. That is, only the grippers configuration is able to immobilize the object. In the *Force Closure Grasp*, the degrees of freedom are equal to zero only if frictional forces at the contact points are considered. That is, a force, usually friction, is required to immobilize the object.

The forces involved in the process of grasping are three: contact force, manipulation force and gripper force. In Fig. 11, the vectors $\vec{F}_1$ and $\vec{F}_2$ represent the contact forces. The manipulation force represents the vector sum of all contact forces on the object. In Fig. 11a this force is equal to $-\vec{F}$. Finally, the gripper forces, defined only for cases that involve two contact points and whose sum of forces and external moments is equal to zero, represent the magnitude of the contact force. In Fig. 11b, the magnitude of $\vec{F}_1 = \vec{F}_2$ is the gripper force.

Figure 11: Existing forces during the grasping process



Source: International Organization for Standardization (2000)

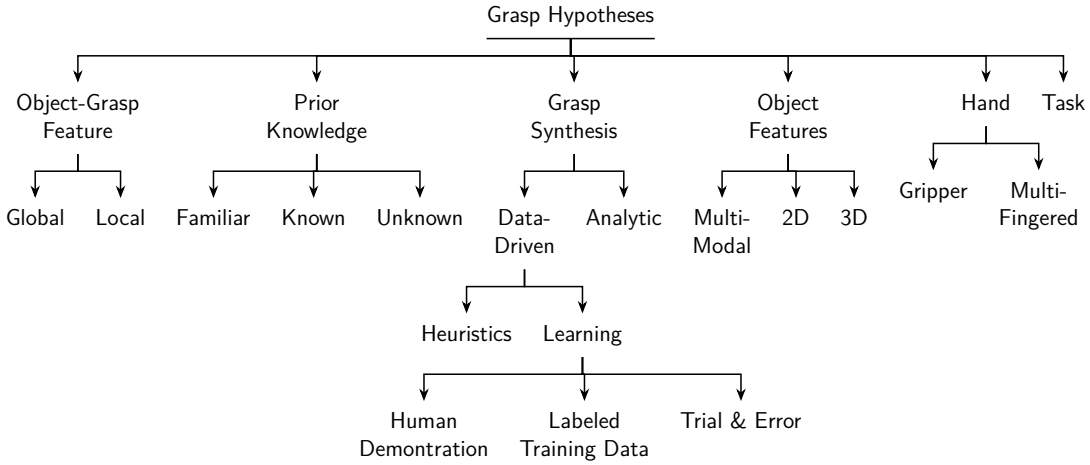
The last relevant concept in robotic grasping is stability, defined in ISO14539:2000 in two different ways. It can be considered the state in which an object maintains its initial position and orientation when a small change in its position occurs as a result of disturbing forces. Or the state in which the grippers always maintain contact with the object, without sliding, when subjected to disturbing forces.

The grasp stability depends on the gripper force and the correct choice of contact points (CASTRO et al., 1997). Usually, the control of the force is the last stage of the grasping process and is dependent on the correct choice of points. The task of classifying grasping as stable using common closure techniques, given an object and a set of contacts is called *grasp analysis*. On the other hand, discovering the set of contacts, given the objects and a set of restrictions is a task called *grasp synthesis*. For both tasks, quality measures are used (RUBERT et al., 2017) so that the robot decides how to grasp the object (LEÓN; MORALES; SANCHO-BRU, 2014).

2.2.1.1 Grasp Synthesis

There are infinite candidate grasps that can be applied to an object, so, as identified by Bohg et al. (2014), a good subset of grasp hypotheses is presented in Fig. 12. The robotic grasp problem is, thus, associated with the type of task that the robot will execute; with the features of the objects - if 2D, 3D or multi-modal feature, like images and sensory data, is used; what type of prior knowledge about the object to be grasped - known, unknown or familiar objects; if global or local features are used to find the grasp points; what type of hand is used - a gripper or a multi-fingered hand; and, finally, the grasp synthesis.

Figure 12: Aspects that influence the final set of grasp hypotheses



Source: Author, as presented in Bohg et al. (2014)

Grasp synthesis methods can, in general, be categorized as analytic or data-driven. Analytic methods, or geometrical ones, are those that construct force closure with a dexterous and stable multi-fingered hand with a certain dynamic behavior (SHIMOGA, 1996; BOHG et al., 2014). Grasp synthesis is formulated as a restricted optimization problem that uses criteria such as kinematic, geometric, or dynamic formulations to measure the properties described (SHIMOGA, 1996). Much of the research on robotic grasping, especially the first works, consolidated the theoretical framework of the area by means of analytic methods. Classic works were developed, for example, by Salisbury (SALISBURY; CRAIG, 1982) and Cutkosky (CUTKOSKY et al., 1989).

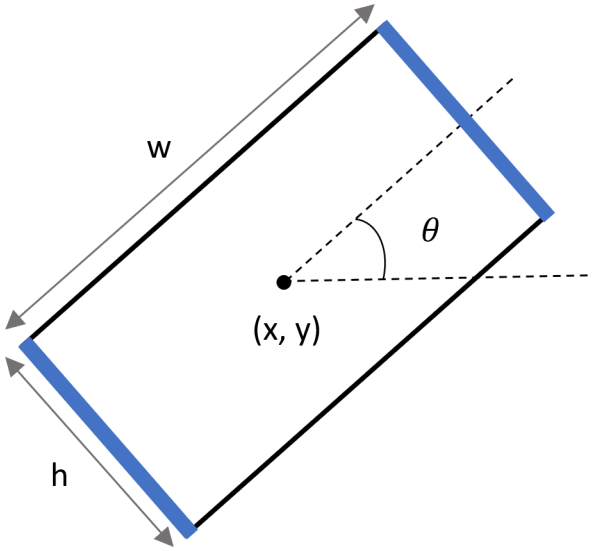
Analytic methods usually assume the existence of a precise geometric model of the object to be grasped, something that is not always possible. Besides, the surface properties or friction coefficients of the object, as well as its weight, center of mass, among others, may not be available (BOHG et al., 2014). A complete review of these methods can be found in the work of Bicchi and Kumar (2000).

Data-based methods, or empirical ones, are based on the search of candidates for

grasping and the classification of these candidates by some metric. This process generally assumes the existence of previous experience of grasping, provided by means of heuristics or learning (BOHG et al., 2014). That is, these methods require annotations of those candidates considered correct (RUBERT et al., 2017) to serve as a model for the algorithms used. Therefore, one must generate these annotations in some way, either employing real robots (LEVINE et al., 2018; PINTO; GUPTA, 2016), simulations (KAPPLER; BOHG; SCHAAL, 2015) or direct annotations in images (LENZ; LEE; SAXENA, 2015).

Among the works that use empirical methods, many deals with some kind of visual information in their algorithms. The main benefit of using images for grasp synthesis is its independence from 3D models of the analyzed objects. In practical scenarios, it is often difficult to obtain a complete and accurate 3D model of an object first seen using vision (SAXENA; DRIEMEYER; NG, 2008).

Figure 13: Representation of the robot's grippers using a grasp rectangle



Source: Author

Using only images, Jiang, Moseson and Saxena (2011) proposed a representation of the position and orientation of a robot's grippers, just before closing for grasping, from five dimensions. The scheme is illustrated in Fig. 13. In the image, (x, y) is the center of the oriented rectangle, w represents the opening of the grippers and h its size, that is, the edges shown in blue represent the grippers of the robot. Finally, there is the angle θ that represents the grippers' orientation. This five-dimensional representation is enough to encode the seven-dimensional representation of the grasp pose (JIANG; MOSESON; SAXENA, 2011) since a normal approximation to the image plane is assumed, so that the 3D orientation is given only by θ.

2.2.2 Robotic Vision

The geometric aspects of image formation and the foundations of visual servoing, one of the themes that most represent the specificity of robotic vision, are presented in this section.

2.2.2.1 Camera Parameters and $2D \leftrightarrow 3D$ Mapping

A camera can be modeled using its extrinsic and intrinsic parameters. The extrinsic parameters define the position and orientation of the camera frame related to a reference world frame by means of a rotation matrix and a translation vector. Intrinsic parameters are necessary to transform a point expressed on the camera frame into a point in the coordinates of the image, in pixels (TRUCCO; VERRI, 1998). The intrinsic parameters are represented by the focal length f, the principal point (o_x, o_y), which expresses the coordinates in pixel of the image center, and the effective size of the image pixels, (s_x, s_y).

These parameters are used to link the positions of points in the world with their corresponding points in the image. Given the homogeneous representation of a point in the world reference frame $[X, Y, Z, 1]^T$ and the set of extrinsic and intrinsic parameters, the perspective projection $[x, y, z]^T$ is given by the linear matrix equation of perspective projections (TRUCCO; VERRI, 1998), presented on Eq. 2.19.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -\frac{f}{s_x} & 0 & o_x \\ 0 & -\frac{f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.19}$$

The first matrix on the second term is the Intrinsic Parameters Matrix, M_{int}, and the second one is the Extrinsic Parameters Matrix, M_{ext}. The M_{ext} is calculated using the matrix representation of the translation vector and the rotations around axes x, y e z, as Eq. 2.20 shows.

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \end{bmatrix} R_z R_y R_x \right)^{-1} \tag{2.20}$$

Using the perspective projection $[x, y, z]^T$, the image coordinates (x_{im}, y_{im}), in pixels, are given by

$$\begin{bmatrix} x_{im} \\ y_{im} \end{bmatrix} = \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix}. \tag{2.21}$$

The opposite problem, that is, obtaining points in the world frame $[X, Y, Z]^T$ from its corresponding points expressed in image coordinates $[x_{im}, y_{im}, 1]$, cannot be resolved in the general case, as depth information is lost in the $3D \rightarrow 2D$ transformation. However, for cases in which the camera parameters are known and it is known that all points are at the same depth, it is possible to do the $2D \rightarrow 3D$ transformation, and recover the X and Y coordinates in the world.

The Eqs. 2.22 and 2.23 show how it is possible to recover these points taking into account the conditions above and Eqs. 2.19 and 2.20.

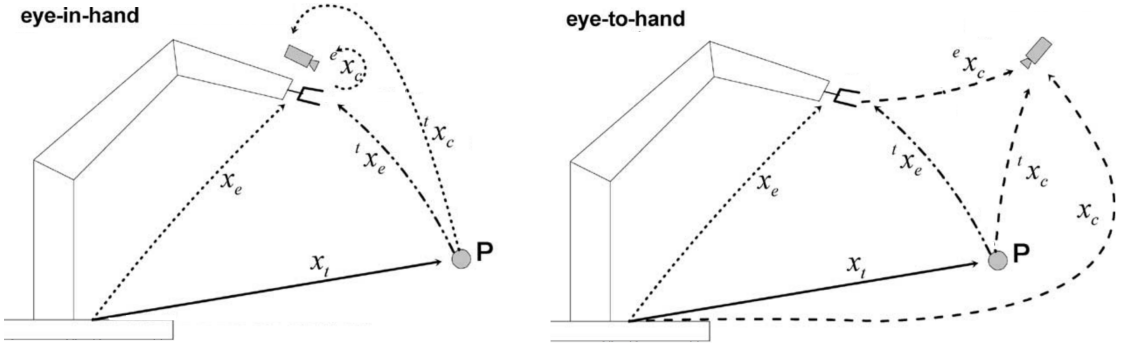
$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \left(\begin{bmatrix} -\frac{f}{s_x} & 0 & o_x \\ 0 & -\frac{f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \right)^{-1} \begin{bmatrix} x_{im} \\ y_{im} \\ 1 \end{bmatrix} \tag{2.22}$$

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \frac{X'}{Z'} \\ \frac{Y'}{Z'} \end{bmatrix} \tag{2.23}$$

2.2.2.2 Visual Servoing Foundations

Visual servo control refers to the use of computer vision data to control the movement of a robot (CHAUMETTE; HUTCHINSON, 2006). The data can be obtained by a camera mounted directly on a manipulator robot (*eye-in-hand*), in which case the movement of the robot induces the movement of the camera, or it can be fixed in the workspace (*eye-to-hand*), so that it can observe the movement of the robot in a stationary configuration (Fig. 14).

Figure 14: Eye-in-hand and eye-to-hand representations



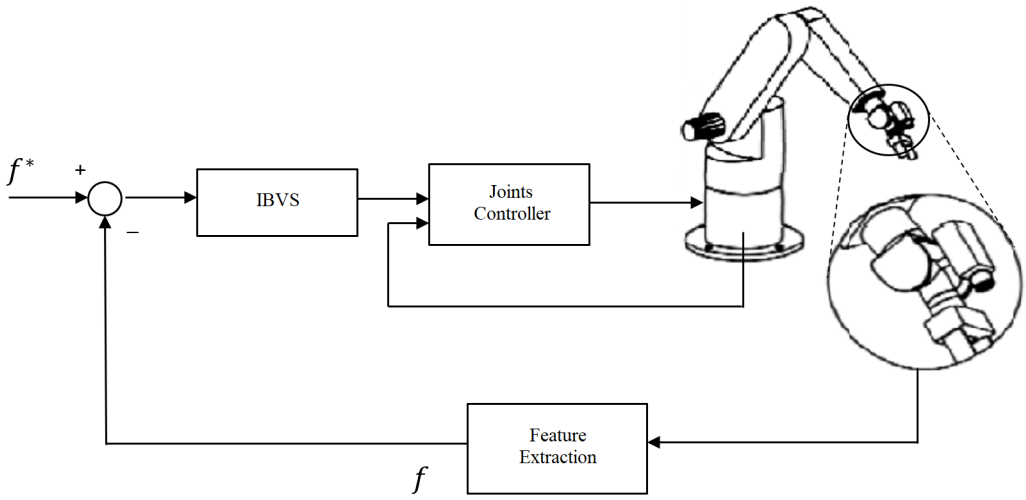
Source: Adapted from Muis and Ohnishi (2005)

The first control system that encompasses visual information in its loop was developed in the work of McCarthy et al. (1968). In this system, the task of processing visual information and the movement of the robot were dissociated and designed in an open-loop way, in an approach known as "static look-and-move". Thus, the systems were

highly sensitive to any modeling errors, such as camera calibration or errors related to the dynamics of the robot, and external disturbances.

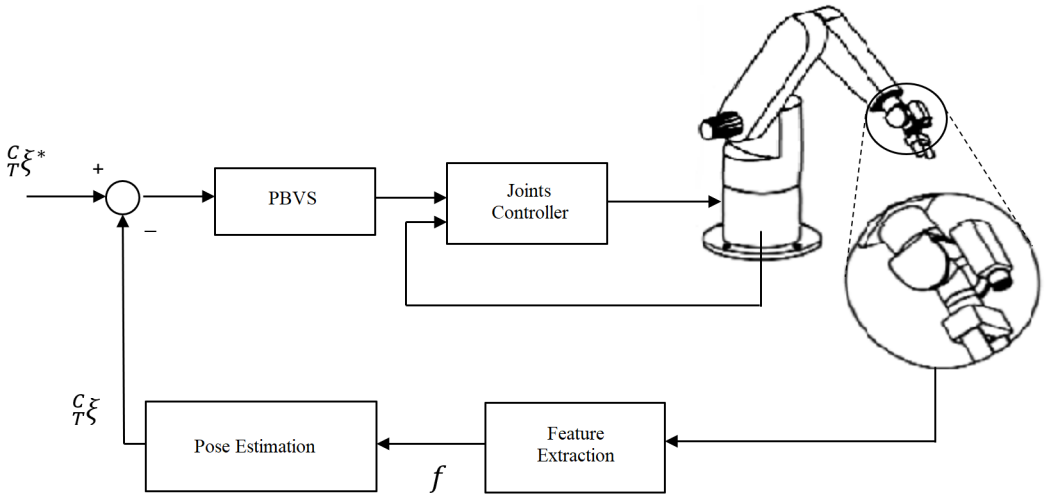
The first work that integrates visual information in the feedback loop was developed by Shirai and Inoue (1973). However, the term visual servoing to identify this type of control was coined only with Hill (1979). The classical Visual Servoing (VS) can be classified according to the nature of the visual information provided to the control system (MUÑOZ, 2011) into Image-Based Visual Servoing (IBVS) or Position-Based Visual Servoing (PBVS). This division was proposed by Sanderson (1980) and, as shown by Figs. 15 and 16, the main difference is in estimating the error in the Cartesian space or in the image features space. Later, the $2\frac{1}{2}$D approach, that combines advantages of both image-based and position-based control schemes was proposed by Malis, Chaumette and Boudet (1999).

Figure 15: Image-Based Visual Servoing (IBVS)



Source: Adapted from Garcia et al. (2009) and Corke (2017)

Figure 16: Position-Based Visual Servoing (PBVS)



Source: Adapted from Garcia et al. (2009) and Corke (2017)

Another way of classifying visual servoing concerns the form of inputting the control signal to the robotic system. As presented by Bachiller, Cerrada and Cerrada (2004), if the control architecture uses the vision system control signal as input to the joint level controller of the robot, it is referred to as *dynamic look-and-move*. On the other hand, if the visual controller is used to directly control the robot's joints, it is called *direct visual servo*. The vast majority of visual servoing applications is of the dynamic look and move type. The difference between vision systems and the servo loop rates might cause problems of stability on the robot (PALMIERI et al., 2012), which makes it impracticable to apply direct visual servo in real scenarios.

2.2.2.3 Classical Dynamic Look-and-move Visual Servoing

The error signal to be minimized in a classical vision-based control system can be described by Eq. 2.24, as established by Chaumette and Hutchinson (2006). The $\mathbf{m}(t)$ vector represents measurements in the image, such as the coordinates of some points of interest. These measurements are used to obtain k features, $\mathbf{s}(\mathbf{m}(t), \mathbf{a})$, where $\mathbf{a}$ is a set of additional information regarding the system, such as the camera's intrinsic parameters or the 3D model of the objects. And $\mathbf{s}^*$ represents the expected values of the features (CHAUMETTE; HUTCHINSON, 2006).

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(t), \mathbf{a}) - \mathbf{s}^* \tag{2.24}$$

This feature vector $\mathbf{s}$ can be considered as the image as a whole in an approach called Direct Visual Servoing (DEGUCHI, 2000; COLLEWET; MARCHAND, 2011). This technique does not require anymore feature extraction nor tracking, however, it has a small convergence compared to classical techniques (BATEUX et al., 2018), so it is still common to use designed features to build $\mathbf{s}$.

Once $\mathbf{s}$ is selected, it is common to design a velocity controller. For this, the relation between the time variation of $\mathbf{s}$ and the velocity of the camera $\mathbf{v}_c$ is necessary. This relation is given by:

$$\dot{\mathbf{s}} = \mathbf{L}_s \mathbf{v}_c, \tag{2.25}$$

where $\mathbf{L}_s \in \mathbb{R}^{k \times 6}$ is the interaction matrix.

The variation of the error in time can be obtained from Eqs. 2.24 and 2.25, which leads to

$$\dot{\mathbf{e}} = \mathbf{L}_e \mathbf{v}_c, \tag{2.26}$$

where $\mathbf{L}_e = \mathbf{L}_s$.

For controlling the robot through velocity input, it is common to use a proportional controller that guarantees the exponential decrease of the error, that is,

$$\dot{\mathbf{e}} = -\lambda \mathbf{e}. \tag{2.27}$$

Finally, using the Eqs. 2.26 and 2.27, the control law is defined as

$$\mathbf{v}_c = -\lambda \mathbf{L}_e^{+} \mathbf{e}, \tag{2.28}$$

where $\mathbf{L}_e^{+}$ is the Moore-Penrose pseudo-inverse for cases where $\mathbf{L}_e$ is not square. In this particular case, the control law takes the form $\mathbf{v}_c = -\lambda \mathbf{L}_e^{-1} \mathbf{e}$. However, according to Chaumette and Hutchinson (2006), in real visual servoing systems it is impossible to know perfectly $\mathbf{L}_e^{+}$, which requires its approximation, so the control law is given by

$$\mathbf{v}_c = -\lambda \hat{\mathbf{L}_e^{+}} \mathbf{e}. \tag{2.29}$$

The choice of $\mathbf{s}$ and, consequently, the estimation of $\hat{\mathbf{L}_e^{+}}$ depends on the type of control used, that is, IBVS or PBVS. To keep the theory short and focus on what is relevant in the methodology developed, only a specific case of PBVS is developed below.

PBVS uses the camera's position and orientation related to some frame to define $\mathbf{s}$ (CHAUMETTE; HUTCHINSON, 2006). In its classic form, PBVS needs the intrinsic parameters of the camera and the 3D model of the observed object so that it can measure its position from the image. Thus, this information makes up the vector $\mathbf{a}$ of Eq. 2.24 and $\mathbf{s}$ is defined from the parameterization used to represent the camera position.

Thus, $\mathbf{s}$ can be defined as $(\mathbf{t}, \theta\mathbf{u})$, where $\mathbf{t}$ is the translation vector and $\theta\mathbf{u}$ is the angle-axis representation of the rotation matrix.

This type of representation admits that any rotation around three axes can be characterized as a single rotation θ around an axis $\mathbf{u}$. As presented in Diebel (2006), these parameters are given by

$$\theta = cos^{-1}\left(\frac{tr(R)-1}{2}\right), \tag{2.30}$$

$$\mathbf{u} = \frac{1}{2sen\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}. \tag{2.31}$$

where R is the rotation matrix and r_{ij} is the element at line i and column j of the same matrix.

If $\mathbf{s}$ is defined as the translation of the current camera frame related to the frame of the desired pose ${}^{c^*}\mathbf{t}_c$, then,

$$\mathbf{s} = ({}^{c^*}\mathbf{t}_c, \theta\mathbf{u}). \tag{2.32}$$

Thus, $\mathbf{s}^* = 0$, $\mathbf{e} = \mathbf{s}$ and the iteration matrix is given by

$$\mathbf{L}_e = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_{\theta u} \end{bmatrix}, \tag{2.33}$$

where,

$$\mathbf{L}_{\theta u} = \mathbf{I}_3 - \frac{\theta}{2}[\mathbf{u}]_\times + \left(1 - \frac{sinc\theta}{sinc^2\frac{\theta}{2}}\right)[\mathbf{u}]_\times^2. \tag{2.34}$$

Once the k dimension of $\mathbf{s}$ is equal to six, $\mathbf{L}_e$ has an inverse and it is given by

$$\hat{\mathbf{L}_e^{-1}} = \begin{bmatrix} \mathbf{R}^T & \mathbf{0} \\ \mathbf{0} & \theta\mathbf{u} \end{bmatrix}. \tag{2.35}$$

So, the control law expressed in Eq. 2.29, can be rewritten using the Eq. 2.35 as the Eq. 2.36, in which the translational and rotational movements are decoupled.

$$\mathbf{v}_c = \begin{bmatrix} -\lambda \mathbf{R}^{T\ c^*}\mathbf{t}_c \\ -\lambda\theta\mathbf{u} \end{bmatrix} \tag{2.36}$$

3 RELATED WORK

In this chapter, a review of related work is presented taking into account the main themes addressed in this dissertation, *i. e.*, robotic grasping and visual servoing. Since both are notably vast themes, only those works that have a clear correlation with the method proposed is exposed in more detail, while other techniques are only cited as illustration.

Du, Wang and Lian (2019) assume that the vision-based robotic grasping system is composed of four main steps, namely target object localization, object pose estimation, grasp detection and grasp planning. Localization refers to finding the target object in the workspace and pose estimation refers to estimating the rotation and translation of this object related to a reference frame. Grasp detection is a stage in which the grasp configuration is calculated, that is, the grasping points, orientation, gripper opening, among others. Grasp planning refers to the final stage in which the manipulator finds the best path to the object and closes its grippers, keeping the object stable.

In this dissertation, target object localization and object pose estimation are subtasks of the grasp detection stage and are processed through deep learning, in an approach called end-to-end grasp detection (DU; WANG; LIAN, 2019). Object localization and object pose estimation are problems with their own literature (DU et al., 2020) and will not be addressed here, so greater emphasis is placed on grasp detection strategies.

The grasp planning stage is often also performed using already implemented algorithms, since they basically represent the problem of path planning and/or inverse kinematics, extensively studied in robotics. However, in some systems this stage is widely explored. Levine et al. (2018) approach the grasp detection and motion planning phase jointly through reinforcement learning, in which successful grasps are treated as reward functions. Other works optimize only the grasp planning, assuming that the grasp points are already available from the detection stage. In this way, Rai et al. (2017) apply a DMP-based method and Amor et al. (2012) approach imitation learning and learning from demonstration techniques for planning, whereas Kwiatkowski and Lipson (2019) use reinforcement learning to find the optimized path to the grasp points.

Here we consider grasp planning as an internal problem of the robot's inverse kinematics, with no restrictions on the movement of the robot to the object, except those related to singularities. Regarding the detection phase, as presented in Chapter 2, the early methods of grasp detection (synthesis), known as analytic methods, rely on the geometric structure of the object to be grasped and have many problems with execution time and force estimation. Besides, they differ in many ways from data-driven methods, so, in the sequence, only data-driven grasp detection methods that use visual information

are addressed, since this is the methodology addressed here.

In Section 3.2, variants of the classic visual servoing are discussed, such as those that consider only pixel intensity to obtain the control law. Some models based on classical machine learning methods, as well as those that use deep learning algorithms, are presented in more detail. Finally, works that deal with the problem of grasping and visual servoing in a joint manner, as in this work, are presented in Section 3.3 to complete the state of the art review.

3.1 Vision-Based Grasp Detection

One of the first works that uses only vision to obtain grasp points was developed by Saxena, Driemeyer and Ng (2008). The authors propose a learning algorithm that does not require the 3D model of the object, predicting points at which the object should be grasped using only its image. A probabilistic supervised algorithm is used to find good grasp points in a pair of images and triangulation is performed to find the 3D position in which the grasp will be performed. The authors also innovate by taking inspiration from the work of Goodale et al. (1991), in which is shown a dissociation between recognizing objects and grasping them. Thus, the authors suppose that an intelligent system capable of recognizing a target object, will not necessarily have an easier time grasping it.

However, in the opposite direction, Nikandrova and Kyrki (2015) show that in many applications, the grasp points found should not only be stable, but also applicable to a specific task. To this end, the authors create a probabilistic model capable of finding points on an object through previous experiences with objects belonging to the same category. Likewise, Veres, Moussa and Taylor (2017) assume that knowing the properties of an object and having an idea of how similar objects can be grasped, can improve the generation of grasp points.

Still regarding information that can help in determining grasp points, Murali et al. (2018) propose a system capable of grasping new objects without any visual sensor, relying only on tactile information. In another approach, Eppner and Brock (2015) explore environmental constraints. The authors present a system that interacts with the environment and transforms the constraints into a control input to obtain robustness in the grasp of thin objects on a table. Kroemer et al. (2010) demonstrate a two-stage system that uses reinforcement learning to plan grasping and imitation learning in conjunction with reactive control to perform the action.

In a different way, Gualtieri and Platt (2017) do not seek to add information to determine grasps, but to find the best way to obtain the visual data. So, the authors explore the relationship between viewpoint and grasp detection performance. Using a real robot, the authors execute experiments in which the grasp pose is known in advance and

the algorithm tries to find a viewpoint that maximizes the chance of the grasp detector localize it more precisely.

Another challenge related to the data used in the training of algorithms for grasp detection is its acquisition, so some authors explore the issue of data availability. As obtaining real data in robotics is expensive, many works use data from simulation to evaluate their algorithms (KAPPLER; BOHG; SCHAAL, 2015; MAHLER et al., 2016; JOHNS; LEUTENEGGER; DAVISON, 2016), since this is a way of obtaining a large amount of data at low cost and in a short time.

By building datasets with thousands of 3D models of objects, the Dex-Net series bring innovations in the way grasp is detected. In the first work, Dex-Net 1.0 (MAHLER et al., 2016), the authors build a dataset of 10,000 3D models and more than 2.5 million grasping points for parallel grippers. An algorithm based on Multi-Armed Bandits and Continuous Correlated Beta Processes (GOETSCHALCKX; POUPART; HOEY, 2011) is also proposed for grasp detection. In the second version of the work, Dex-Net 2.0 (MAHLER et al., 2017), the authors generate a new dataset from the first one and propose a CNN to predict the points. Finally, in the last work, Dex-Net 3.0 (MAHLER et al., 2018), the authors collect a third dataset to train a new CNN, that deals with vacuum grippers. All of the datasets were gathered in simulation.

However, the transition from simulation to the real world generally suffers from the "reality gap" and results in a fragile grasp, as presented by Bohg et al. (2014). So, Bousmalis et al. (2018) seeks to overcome the difficulty of generalizing the simulation to real world through Domain Adaptation (BEN-DAVID et al., 2007), and Tobin et al. (2018) use simulated grasp data by randomizing rendering in the simulator using Domain Randomization (TOBIN et al., 2017).

According to James et al. (2019), however, domain adaptation needs a large amount of unlabelled real-world data and domain randomization can waste modeling power. Then, they present a Randomized-to-Canonical Adaptation Network, which translates randomized rendered images into their equivalent canonical version, with no need for real data. To obtain the grasp pose, reinforcement learning is used and, at the test phase, the real-world image is also transformed into its canonical version so that the algorithm deals with the same modality of data used in the training.

On the other hand, some works do acquire real data through self-supervision, which eases the human work involved in the dataset creation. Jang et al. (2017) use a self-supervised method so that a robot can autonomously collect a large self-labeled dataset. They also use label propagation to build a large object classification dataset in a semi-supervised way. For designing the method, they take inspiration from the two-stream hypothesis of human vision (UNGERLEIDER; HAXBY, 1994) in which visual reasoning can be decomposed into a dorsal stream that reasons about space and a ventral stream

that reasons about object identity. They train a network with two specialized streams and demonstrate that it can outperform single-streams approaches for the grasping task.

Pinto and Gupta (2016) also use self-supervision to build a grasp dataset from 50k tries and 700 robot hours. They innovate by addressing grasp regression as 18 binary classification problems and by using multi-stage learning. The network trained with the random trial dataset is further used as a basis for fine-tuning in each new trial of the second phase of training, using data aggregation techniques (ROSS; GORDON; BAGNELL, 2011). This is one of the first works that bring big data to the field of robotics, but do not achieve great accuracy during real tests, presenting a success rate of 66% and 73% for novel and previously seen objects, respectively.

Another work that approaches grasp detection as classification and also uses aggregation techniques to increase data is presented by Wang et al. (2016). The authors use three kinds of modal information (RGB, depth and normal surface) to infer a set of spatial features of objects. As an initial step, they perform RANSAC and Extended Graph-Based Segmentation to discriminate the objects in the workspace and use the length, width and height of the segmented parts of the image to find which objects are graspable. Finally, using the rectangular representation, reinforcement learning, and the spatial features, they achieve 81.8% accuracy on Cornell Grasping Dataset (CGD), with a prediction speed of 7.1 fps, which places them very far from the current state of the art.

Park and Chun (2018) also approach grasp detection as a classification problem and use the idea of multi-stage learning applied in a Spatial Transform Network. They evaluate their algorithm in the CGD achieving 89.6% accuracy and a prediction speed of 43.5 fps, both below the current state of the art.

Motivated by the idea that deep learning methods for predicting grasps are largely influenced by the choice of CNN architecture, Asif, Tang and Harrer (2018a) also propose a multi-stage method called EmsembleNet. The network consists of two steps, namely grasp generation and grasp evaluation. In the first step, four different grasp representations are obtained (regression, joint regression-classification, segmentation, and heuristics) and then the network produces confidence scores for every representation and chooses the best one.

Following the same idea of network optimization, Hossain, Capi and Jindai (2017) propose a model based on Genetic Algorithm to automate some Deep Belief Network (DBN) hyper-parameter choices. However, the authors did not provide performance measures for their network and the best grasp detection has low priority since DBN is used only for localization and recognition of the grasped objects.

Employing other types of machine learning algorithms and/or feature extraction for the grasp detection problem, Liang et al. (2019) address it directly from point clouds using geometric information and heuristics, Ren et al. (2017) perform a fusion of sensory data

and training acceleration with a hierarchical method, Trottier, Giguere and Chaib-draa (2017) use dictionary learning and sparse representations and Wei et al. (2017) explore extreme learning machines.

Since the vast majority of the works presented above approaches robotic grasping as a regression problem using data from visual sensors, Gualtieri et al. (2016) propose a series of innovations that result in better performance of these methods. On the other hand, Johns, Leutenegger and Davison (2016) establish the idea of grasp function, as opposed to the regression strategy of just one grasp pose. The authors implement a CNN capable of predicting the score of each possible grasp pose and then add a term of uncertainty of position and orientation to this learned function, taking into account the geometry of the object and its depth image. Thus, the model predicts robust grasps, considering the uncertainties involved in its execution.

This work approaches the rectangular representation presented in Chapter 2 to obtain the best place where two parallel grippers can grasp an object, based only on its image. This representation was developed by Jiang, Moseson and Saxena (2011) as opposed to the grasping point previously used to detect grasps, since, according to the authors, these configurations only partially represent the grippers and are, therefore, sub-optimal.

The images required for the algorithm design were obtained through the Cornell Grasping Dataset (JIANG; MOSESON; SAXENA, 2011). Such dataset is composed of several images of domestic objects and their respective ground truth grasp rectangles.

Lenz, Lee and Saxena (2015) were one of the first to test their algorithms in this dataset and to use deep learning for grasp detection. Although they did not obtain a relevant accuracy, it encouraged other researchers to investigate grasp problem using the rectangle representation.

Redmon and Angelova (2015), for example, also used the CGD to fine-tune an AlexNet model (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) using the RGD channels of the images. The authors propose two ways of performing the grasp rectangle regression, namely direct regression, and multigrasp detection. The direct regression model presupposes the existence of only one object in the scene, whereas the multigrasp model divides the input image into parts and assumes the existence of a grasp rectangle in each of them. In this way, more than one rectangle can be predicted by the network, which does so through a map of probabilities associated with each part of the image.

Kumra and Kanan (2017) search for the grasp rectangle in the CGD using two pre-trained ResNets (HE et al., 2016) executed in parallel. Each of them processes one type of information, namely the RGB components and the depth component. The results obtained surpass those of Redmon and Angelova (2015) and, although the joint networks have a higher number of parameters, the authors report a prediction time of 103 ms.

Guo et al. (2017) use the idea that visual and tactile data are complementary in the task of robotic grasping to design their deep network. They use the rectangular representation to detect grasps from the CGD images and evaluate them, according to their stability, using information from the tactile sensors. However, this information has very little effect on accuracy, resulting in approximately 50% accuracy for cases where only visual information is available and for cases where both types of data are available, when evaluated on a real robot. Regarding the evaluation on the CGD (using only visual data), the authors obtain a higher accuracy than that of Kumra and Kanan (2017), relying on recent advances in object detection, specifically, the Faster R-CNN designed by Ren et al. (2015).

Also based on the Faster R-CNN, Chu, Xu and Vela (2018) developed a system where the identification of the grasp points is divided into two stages, the first being responsible for detection and the second by the classification of the rectangle's orientation. This network, from the object detection point of view, is a two-stage detector whereas single-stage detectors perform direct regression of the rectangle. So, although the authors achieve good detection results in CGD, the grasp detection runs at only 8 fps.

Zhou et al. (2018) also developed a system inspired by Faster R-CNN. However, instead of varying sizes and proportions of scale, the authors propose to vary the angles of the rectangle, which, according to them, are more important for the definition of the grasp rectangle. In this way, the authors achieve accuracy close to 100%. In addition, the built network is single-stage, but the prediction speed is increased by only 0.2 fps compared to Chu, Xu and Vela (2018).

Morrison, Corke and Leitner (2018) present a generative neural network (GG-CNN) that detects grasps, and an associated quality index, in all pixels of the input image, so, a fully convolutional architecture can be used. The authors use the CGD to transform the rectangular representation into parameterized images of Grasp Quality, Angle and Width, and use them for training. For testing, the network is evaluated in a real scenario with a prediction speed of 52,6 fps. However, the authors do not present the network efficiency in the CGD.

Following the same approach, Gu, Su and Bi (2019) reparameterize the instances from the CGD into three images representing the ground truth grasp and use a fully convolutional architecture (AGN) as well. They introduce a mechanism of attention in the network so that the model focuses on salient features and, thus, reaches state-of-the-art results in the dataset. However, the authors do not make it clear which set of the dataset is used to obtain this result. Furthermore, the developed network is slightly slower than GG-CNN and no real tests are presented.

Ghazaei et al. (2018) also attempt to use a lightness fully convolutional architecture, so they transform the rectangular representation into grasp belief maps using Gaussian

kernels. In this way, it is possible to deal with the ambiguities present in the detection of only one grasp pose. They evaluate their method on the CGD and achieve competitive results, but do not incorporate the experiment in a real robot and do not show whether the prediction speed of 17.9 fps does, in fact, ensure real-time execution.

Asif, Tang and Harrer (2018b) create a dataset whose ground truths are available as pixel-wise annotations for grasp affordances. The authors use a new approach called Dilated Dense Fire on the encoder of the developed network, GraspNet, and implement it in a low power Nvidia Jetson X1 in real-time. In the CGD, they achieve competitive results in the order of 90%.

Differently from pixel-wise, Zhang et al. (2018) use Regions of Interest (ROI) to extract features instead of the entire image. First, the ROIs are generated and submitted to an adaptive pooling, then each of the ROIs is processed with the grasp detector. Using the CGD, the authors obtain more than 93% accuracy with a prediction speed of 25,16fps.

Using Hierarchical Cascaded Forests (HCF), Asif, Bennamoun and Sohel (2017) create a method for joint object recognition, categorization, and grasp detection. They create a new way of representing RGB-D data called Structural Embedding so that HCF is computed in different levels of an image hierarchy. However, the authors do not provide information on prediction speed and its accuracy in the CGD is far behind the current best result.

Recently, Chen, Huang and Meng (2019) stipulate a new way of evaluating accuracy on the CGD, using the concept of grasp path instead of direct comparison with a ground truth rectangle. However, even if this strategy increases the accuracy of the proposed method, the results obtained in the dataset are at least ten percentage points behind the top results.

This work aims to simplify the convolutional neural network used to predict grasp points. So, when trained, it does not occupy large memory space with its parameters and can be quickly executed without penalizing accuracy. Thus, making it possible to use the algorithm in real-time robotic systems.

The amount of data-driven methods for grasping is large enough to be addressed in a vast review, so one can suggest reading the work of Bohg et al. (2014). For a review and comparative analysis of several reinforcement learning methods applied to vision-guided grasping, the work of Quillen et al. (2018) is suggested. Finally, for an extensive review of deep learning methods in robotic grasp detection, one can resort to the work of Caldera, Rassau and Chai (2018).

3.2 Visual Servo Control

As seen in Chapter 2, classic Visual Servoing (VS) strategies demand the extraction of visual features as an input to the control law. These features must be correctly selected since the robustness of the control is directly related to this selection. In this sense, many works assume the availability of this information to design the controller (ESPIAU; CHAUMETTE; RIVES, 1992; MOHTA; KUMAR; DANIILIDIS, 2014; WILSON; HULLS; BELL, 1996). However, the use of projected features requires manual engineering for each new system and, according to Lee, Levine and Abbeel (2017), it avoids the exploitation of statistical regularities that make the developed VS system more flexible.

Assuming that notable convergence and stability issues can appear if the VS is performed in a region away from the projected operating point (CHAUMETTE, 1998), the classic VS techniques mainly deal with controller convergence and stability analysis. Thus, they are concerned with the gain that provides a good tradeoff between these two requirements, with the estimation of the interaction matrix and other parameter tunning, failing to learn features (SAMPEDRO et al., 2018).

Still relying on projected features, some studies sought to investigate the applicability of machine learning algorithms in VS. The first one was developed by Miller (1987), in which a multi-layer perceptron was used to estimate the interaction matrix. Other authors also sought to automate the interaction matrix estimation using an MLP (HASHIMOTO et al., 1992; WEI; HIRZINGER, 1999; RAMACHANDRAM; RAJESWARI, 2003) or support vector machines (PANDYA; KRISHNA; JAWAHAR, 2015).

The first step towards the independence of feature extraction and tracking was taken with the development of Direct Visual Servoing (DVS) by Deguchi (2000). The technique has been improved recently so that it can better deal with nonlinearities present in the minimized cost function and convergence domain (CARON; MARCHAND; MOUADDIB, 2013; SILVEIRA; MALIS, 2012; SILVEIRA, 2014). However, although the control law is obtained directly from pixel intensity, information about the objects is still necessary and the complexity of the image processing algorithms tend to compromise their use.

In order to create a control system that uses neither projected features nor metric information about objects, and that is still robust to imaging conditions, Silveira and Malis (2012) developed a new DVS model. The authors explore the projective and geometric parameters that relate the current and desired image using the pixel intensity, however, the system has initialization issues since obtaining these parameters is computationally expensive.

In a later work (SILVEIRA, 2014), the authors present three optimization methods to try to overcome computational complexity. However, cheaper methods do not have good convergence properties and generally need prior knowledge of the system. Furthermore,

when evaluated in the real world, the algorithms converge only with reduced gains and the cheaper method does not converge.

The developed controllers, however, were one of the few that exploit non-metric information, since it is difficult to find a control error that is diffeomorphic to the camera pose (SILVEIRA, 2014). In addition, the controller is robust to inaccuracy in camera calibration and image noise. However, the camera's intrinsic parameters are still necessary and image processing is still a barrier.

In this sense, the most recent VS techniques have explored deep learning algorithms to simultaneously overcome the problems of feature extraction and tracking, generalization, prior knowledge of the system and, in some cases, processing time. The main algorithms are of the deep reinforcement learning type (ZHANG et al., 2015; SHI et al., 2016; LEE; LEVINE; ABBEEL, 2017; FINN; LEVINE, 2017; SAMPEDRO et al., 2018; SADEGHI et al., 2018; SADEGHI, 2019).

The first work that demonstrated the possibility of generating a controller from raw-pixel images, without any prior knowledge of configuration, was developed by Zhang et al. (2015). The authors use a Deep Q-Network to perform a target reaching task controlling 3 joints of a robot by means of deep visuomotor policies. The training was done in simulation and failed to be confronted with real images, however, when the camera images were replaced by synthetic images, the robot was controlled.

In a similar approach, Lee, Levine and Abbeel (2017) use a predictive bilinear model and Q-iteration to focus on reducing errors in the object of interest in a car tracking task. However, they do not guarantee the generalization of the model trained by reinforcement learning for different objects, only for the appearance of the target, changes in viewpoint and occlusions. In the same way, Sadeghi et al. (2018) used randomized simulation and a small amount of real-world data of trajectories performed during a reaching task to train a deep neural network augmented with recurrent connections. The model can be applied to new objects, but unlike default VS, the query image is not the desired image that the camera should see, but rather an object that the robot should reach.

Other works that follow the reinforcement learning approach use deterministic police gradients to design a new IBVS (SAMPEDRO et al., 2018) or Fuzzy Q-Learning, relying on feature extraction (SHI et al., 2016), to control multirotor aerial robots. A complete survey of the methods developed for VS using reinforcement learning, classical techniques and other learning algorithms can be found in the works of Kragic, Christensen et al. (2002), Chen, Li and Kwok (2011) and Sun et al. (2018).

In a different approach, some works that explore deep learning in visual servoing do so through convolutional neural networks. Although Reinforcement Learning (RL) appears to be the best approach to the problem, by mapping joint angles and camera

images directly to the joint torques, CNNs can also be trained to perform end-to-end visual servoing. In addition, the generalization power achieved by CNNs is superior to that of RL, since the parameters learned by RL are specific to the environment and task (SAXENA et al., 2017).

Saxena et al. (2017) developed a CNN trained to perform visual servoing in different environments, without knowledge of the scene's geometry. To this end, the authors trained a network, based on the FlowNet architecture (DOSOVITSKIY et al., 2015), with the 7-Scenes dataset (GLOCKER et al., 2013), which has several images taken in sequence through transformations in the camera. Thus, having the current image I_c and the desired image I_d, the network is able to predict the homogeneous transformation that relates the camera pose in I_d and the camera pose in I_c. The authors performed tests on a quadcopter and achieved good results in both indoor and outdoor environments.

Bateux et al. (2018), like the previous authors, developed a CNN capable of predicting the transformation that occurred in a camera through two images. Some essential differences are in the architecture used, in the robot operated and, mainly in the dataset used. The networks are based on fine-tunning of the AlexNet and VGG (SIMONYAN; ZISSERMAN, 2014) networks, and the operated robot is a 6 DoF manipulator. The authors developed their own dataset using just one image. Starting from virtual cameras, it was possible to generate thousands of images and their associated transformations based on homography.

The work also addresses control in cases of a 3D target object, as well as cases in which the object is seen for the first time by the system. In both the control converge, with errors in the order of 1 mm, even in situations of lighting variation and occlusions. In this way, the authors demonstrate that CNNs have robustness and good generalization capacity for the task of visual servoing in 3D objects, even if trained only with images, *i. e.* without depth information.

In order to carry out the control, both works use the relative pose predicted by the network as input of a PBVS, according to Eq. 2.36. This approach is the most similar to the one developed in this dissertation, as presented in Chapter 4.

3.3 Joint Application

Most of the approaches for the robotic grasping task perform one-shot grasp detection and can not respond to changes in the environment. Thus, the insertion of visual feedback in grasp systems is desirable since it makes it robust to perceptual noise, object movement, and kinematic inaccuracies (VIERECK et al., 2017). However, even with modern hardware, classical approaches require a deterrent amount of time for closed-loop applications, and only perform adjustments of force, without visual feedback (LAMPE;

RIEDMILLER, 2013).

Thus, some works started to include a stage of visual servoing to deal with possible disturbances during grasp execution, with a survey given by (KRAGIC; CHRISTENSEN et al., 2002). However, VS is applied in specific cases in these works, in which full knowledge about the camera and the object to be grasped are available, and the grasp detection stage is performed just once.

In this way, learning visual representations for perception-action (PIATER et al., 2011) that adhere to reactive paradigm and that generates a control signal directly from sensory input, without high-level reasoning (LAMPE; RIEDMILLER, 2013), can help in closed-loop grasping. So, some authors developed variants of VS specifically for reaching and grasping tasks, using uncalibrated camera (PIEPMEIER; MCMURRAY; LIPKIN, 2004), calculating the visual-motor Jacobian without knowledge of the system (SHADEMAN; FARAHMAND; JÄGERSAND, 2010), or using policy iteration and imitation learning techniques (RATLIFF; BAGNELL; SRINIVASA, 2007; STULP et al., 2011).

One of these works, developed by Lampe and Riedmiller (2013), uses reinforcement learning to determine the outcome of a trajectory and if it is the target state of the visual servoing. That is, if the robot will reach a final position in which closing the grippers results in a successful grasp. In addition, the authors propose a grasp success prediction that anticipates grasp success based on current visual information, to avoid multiple attempts of grasping.

However, because they are applied specifically to certain objects and because they still rely on some kind of prior knowledge, these systems do not have full capacity for generalization. Thus, more recently, a significant number of works have explored the use of deep learning as an approach to the problem of closed-loop grasping.

In this sense, Levine et al. (2016b) propose a grasp system based on two components. The first part is a prediction CNN that receives an image and a motion command as input and outputs the probability that, by executing such a command, the resulting grasp will be satisfactory. The second component is the visual servoing function. This uses the prediction CNN to choose the command that will continuously control the robot towards a successful grasp. Separating the hand-eye coordination system between components, it is possible to train the grasp CNN using standard supervised learning and design the control mechanism to use the network prediction so that grasp performance is optimized. The resulting method can be interpreted as a form of deep reinforcement learning (LEVINE et al., 2016b).

Their work also explores, in addition to the developed grasping method, the relationship between the amount of data available for training the network and its performance. To this end, the authors carry out experiments on large robot platforms, collecting a total

of 800K grasp attempts without any human intervention, during 2 months of experiments.

Aiming to improve this work, Viereck et al. (2017) develop a system based on CNN to learn visuomotor skills using depth images in simulation. The learned controller is a function that calculates the *distance-to-nearest-grasp* so that it can react to changes in the position and orientation of objects. Starting from simulation, and mounting the depth sensor near the robot end-effector, the authors manage to adapt the algorithm to the real world, avoiding the two months of training experience performed by Levine et al. (2016b). However, the developed CNN calculates the distance in relation to a grasp pose given a priori. This grasp pose is obtained using the algorithm developed by Pas and Platt (2017), so that errors in the grasp detection can induce errors in the control.

Later, Levine et al. (2018) extended the previous work (LEVINE et al., 2016b), collecting another 900K grasp attempts using a different robotic platform. The goal was to determine the ability to transfer learning between robots and how data from a different set of robots can aid learning. However, as the grasping task was learned end-to-end, directly generating motor torques from images, transfer to other robots and scenes cannot be done without large amounts of extra training data (MORRISON; CORKE; LEITNER, 2019).

Inspired by the work of Viereck et al. (2017) and Levine et al. (2018), which receive actions as inputs to the network, Wang et al. (2019) developed a Grasp Quality CNN that relates these actions to the images that the robot would see when executing them, using homography. Thus, from a dataset of only 30K grasp trials, collected in a self-supervised manner, the authors remove the need to generalize over different actions and predict the grasp success metrics in a large number of poses.

Finally, in a way closer to the methodology developed in this work, Morrison, Corke and Leitner (2019) developed a system for closed-loop grasping in which the grasp detection and visual servoing are not learned in a joint manner. The authors use a Fully CNN to obtain the grasp points and apply a position-based visual servoing to make the pose of the grippers match the predicted grasp pose.

In contrast, in the system presented in this work, the prediction of the grasp network is not used during the control and, in addition, the visual servoing is also learned and performed by a CNN. Control and grasp detection run simultaneously, so the control is done to keep the object in the camera's field of view and to keep the robot in a position that favors the grasp execution. All details are presented in the next section.

4 METHODOLOGY

Robotic manipulation systems must overcome some challenges to be applicable in the real world. First, they must be able to generalize to a large number of objects, whether they are known or unknown. They must also be able to react to changes in the environment and deal with non-static objects. Finally, its processing cannot be time-prohibitive, but performed in real-time, as humans do.

To address the first challenge, a convolutional neural network is designed and trained for grasp detection on a large number of objects. To avoid the one-shot grasp and consider changes in the workspace reactively, a visual controller is designed. In addition, for the controller to satisfy the first generalization condition, it is learned by a CNN capable of end-to-end visual servoing. The real-time execution of the algorithms is guaranteed since the developed networks are, by design, both light and fast.

The purpose of the VS is to guide the manipulator, through comparisons between the image continuously obtained by the camera and a reference image, to a position where the robot has a full view of the object so that the grasp detection conditions are met. Thus, the application of the method encompasses all situations in which a robotic manipulator, with a camera mounted in eye-in-hand mode, must pursue and grasp an object.

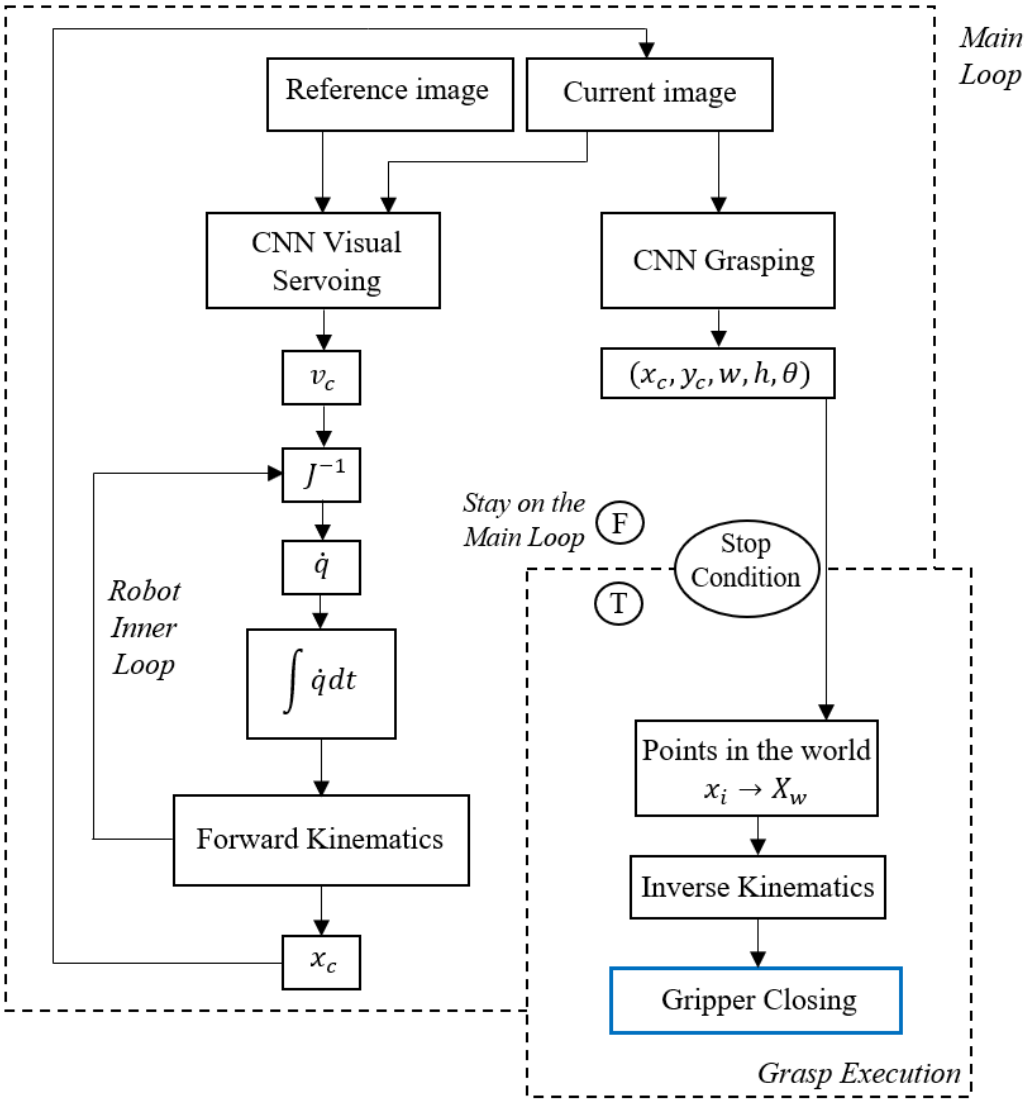
The system consists of three phases: design stage, test stage, and operational stage. The first is based on designing the architectures and training of the CNNs, as well as gathering and processing the datasets. In the second phase, offline results are obtained using validation sets and evaluated based on their accuracy, speed and application domain. The third phase involves the implementation of the trained networks on a robot to assess their adequacy in real-time and real-world applications.

The requirement for the system operation in the operational phase is to obtain an image of the target object *a priori*, which will be used as setpoint by the VS. The control loop is executed as long as the L1-norm of the control signal is greater than a certain threshold. The operational phase is outlined in Fig. 17.

A single reference image is presented to the system as one of Visual Servoing CNN's inputs. The image obtained by the camera at the present time serves as the second input in this network and as the input for the grasp CNN. Both networks run continuously, as the grasp CNN predicts the rectangle in real-time for monitoring purposes and the VS network executes real-time control of the robot's pose.

The VS CNN predicts a velocity signal, which must be multiplied by a proportional gain λ, to be applied in the camera. The robot's internal controller performs all necessary calculations to find the joints velocities that guarantee the predicted velocity in the camera.

Figure 17: Schematic summary of the proposed dynamic grasp system



Source: Author

At each loop execution, the current image is updated according to the robot's current position. This loop is repeated until the L1-norm of the control signal is below some pre-defined threshold, so that the current image is close enough to the reference.

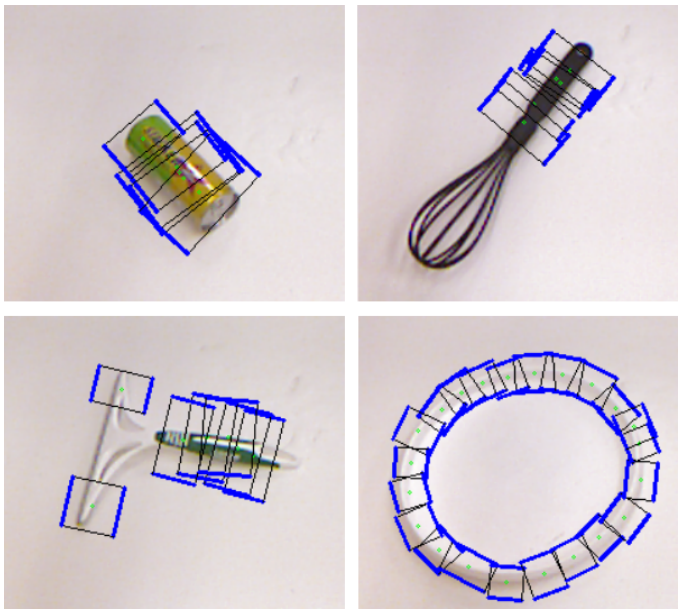
When the stop condition is satisfied, the prediction of the grasp network is mapped to world coordinates. Then, the robot executes the inverse kinematics to reach the predicted points and close the grippers. Since no depth information is considered and mapping to world coordinates does not allow it to be found, the robot approaches the object up to a height of $Z = 23$ cm, by default.

The methodology used in all phases of the proposed system is presented below. Much of the efforts of this work, however, is in the design phase, which is described in greater detail. All network illustrations presented in this section were designed using the open software *PlotNeuralNet*, developed by Iqbal (2018).

4.1 Cornell Grasping Dataset

The Cornell Grasping dataset is used to train the grasp network. It is a set of 885 images and associated point clouds of 240 common household objects, which would potentially be found by a personal robot. These objects have the appropriate size and shape to be grasped by a robotic arm equipped with parallel grippers capable of opening up to 4 inches. These dataset features make it possible to apply the trained agent not only in domestic environments, but also in any situation where the objects of interest can be grasped by parallel grippers. Fig. 18 shows some instances of the dataset.

Figure 18: Examples of objects found in the Cornell Grasping Dataset and their respective grasp rectangles. The blue edges represent the potential poses of the grippers for grasping.



Source: Author (using instances of the dataset)

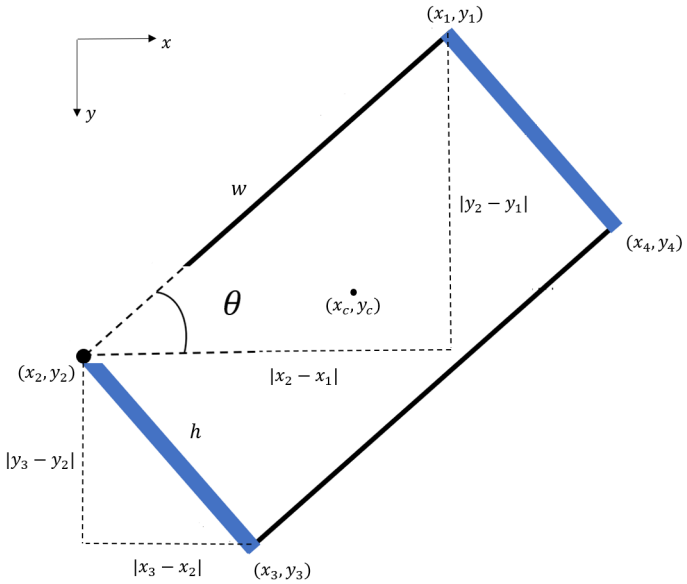
To obtain the data, a Kinect sensor mounted in the robot's end-effector was used. During the data acquisition, the positions of the robot were chosen so that the manipulator can grasp the object from a path normal to the image plane.

For encoding the dataset ground truths, the grasp rectangles were compiled into a text file with $4n$ lines, with n being the number of rectangles associated with the image. In this way, four lines are used to define each rectangle as they represent the x and y coordinates of the four vertices listed in counterclockwise order.

However, this way of expressing the grasp rectangles is unusual for training a convolutional neural network. Thus, it is necessary to extract the values that define them in another way. Based on the vertices, five parameters are obtained for their representation, according to the methodology found in Jiang, Moseson and Saxena (2011), introduced in Chapter 2.

Fig. 19 and the equations that follow show how the dataset is reparameterized into (x, y, w, h, θ).

Figure 19: Rectangle encoding using its vertices coordinates



Source: Author

The x_c and y_c parameters, which represents the x and y coordinate of the rectangle's center point, respectively, can be obtained from:

$$x_c = \frac{\sum_{i=1}^{4} x_i}{4}, \qquad y_c = \frac{\sum_{i=1}^{4} y_i}{4}. \tag{4.1}$$

The grippers opening, w, and height, h, are calculated from the vertices as follows:

$$w = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \qquad h = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}. \tag{4.2}$$

Finally, θ, which represents the orientation of the grippers relative to the horizontal axis, is given by:

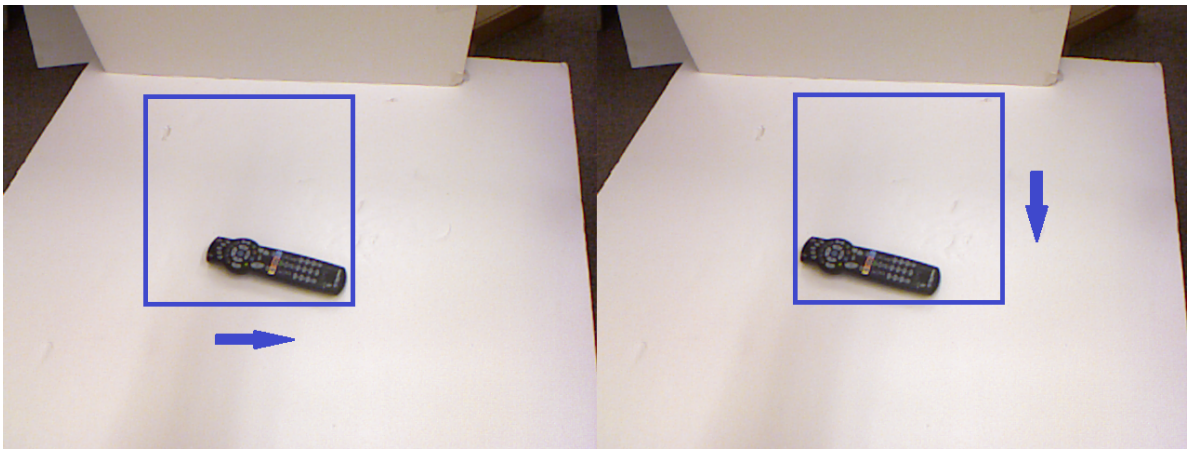
$$\theta = \text{atan2}(y_2 - y_1, x_2 - x_1). \tag{4.3}$$

4.1.1 Data Augmentation

Data augmentation (TANNER; WONG, 1987) can be used when the dataset available for training is small compared to the number of parameters that must be adjusted in the network. The strategy considered is based on the assumption that the greater the amount of visual information in the network's input image, the better its generalization capacity. Thus, instead of simply assigning horizontal and vertical shifts to the images, a

320×320 cropping window slides in the image under the restriction that no part of the object is left out of the representation. Fig. 20 demonstrates the idea.

Figure 20: Cropping window used to generate new images



Source: Author

It should be noted that, unlike a classification problem, the generation of new images for regression problems makes necessary adaptations on labels. Thus, the ground truth values for the original 640×480 images were adjusted for each image generated in proportions 320×320 and adjusted again as the images were scaled to 224×224. In order to crop the image without part of the object being outside the representation, the values of the labeled grasp rectangles were taken into account. Thus, the x and y displacements of the window were limited by the minimum x and y coordinates of the associated grasp rectangles, respectively.

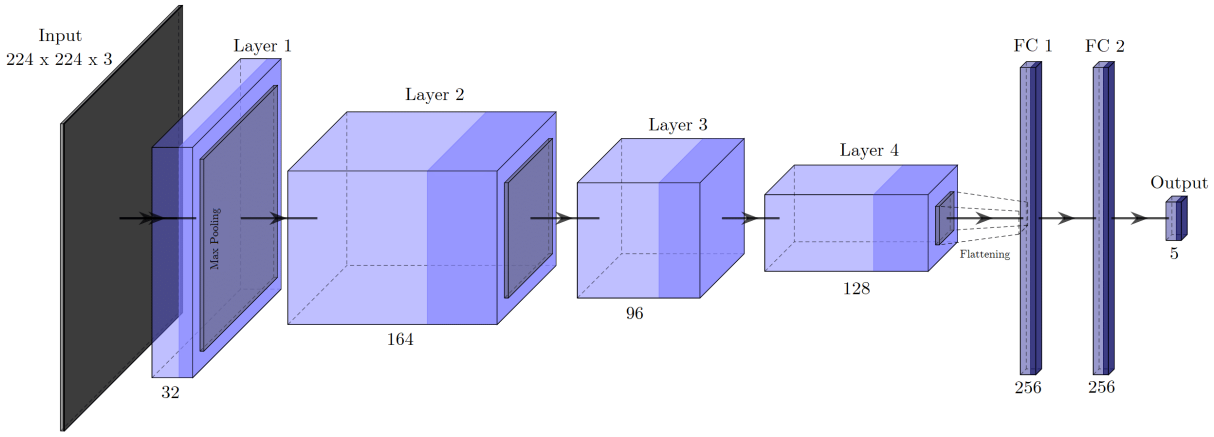
Applying data augmentation, the number of images available for training the network is increased by more than 335 times, resulting in 297,338 instances generated from just 885 images. In addition to data augmentation, the only pre-processing performed on the images is the normalization of the RGB channels.

4.2 Grasping Network Architecture

The convolutional network architecture illustrated in Fig. 21, inspired by the very simple and yet powerful Alexnet architecture (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), is proposed for grasp detection. The network receives as input a $224 \times 224 \times 3$ image, in which 3 means RGB channels, without any depth information. The first layer is composed of 32 convolution filters with dimensions 3×3 and the second layer contains 164 of these filters. In both, the convolution operations are performed with stride 2 and zero-padding, and are followed by batch normalization and 2×2 max pooling.

The third layer contains 96 convolution filters in which the convolutions are performed with stride 1 and zero-padding, and followed only by batch normalization. The

Figure 21: Convolutional neural network architecture for grasping



Source: Author

Table 1: Output Dimensions and Number of Parameters of the Grasping CNN

Layer	Output dimensions	Parameters
Convolutional Layer 1	$112 \times 112 \times 32$	896
Normalization	$112 \times 112 \times 32$	128
Max Pooling	$56 \times 56 \times 32$	0
Convolutional Layer 2	$28 \times 28 \times 164$	47,396
Normalization	$28 \times 28 \times 164$	656
Max Pooling	$14 \times 14 \times 164$	0
Convolutional Layer 3	$14 \times 14 \times 96$	141,792
Normalization	$14 \times 14 \times 96$	0
Convolutional Layer 4	$12 \times 12 \times 128$	110,720
Max Pooling	$6 \times 6 \times 128$	0
Flattening	1×4608	0
Fully Connected 1	1×256	1,179,904
Fully Connected 2	1×256	65,792
Output	1×5	1,285
Total		1,548,569

fourth, and last, convolutional layer is composed of 128 filters performed with stride 1 and followed by 2×2 max pooling.

After the last layer of convolutions, the resulting feature map is flattened in an one-dimensional vector with 4,608 elements. This vector is further delivered to two fully connected layers with 256 neurons each. Between these layers, a dropout rate of 50% is considered during training.

Finally, the output layer consists of 5 neurons corresponding to the (x, y, w, h, θ) values that encode the grasp rectangle. In all layers, the activation function used is ReLU, except in the output layer, in which the linear function is used.

Table 1 shows the evolution of the data dimensions and the number of parameters

involved in each operation. The implemented network is noticeably smaller than others described in the literature for solving the grasping problem. The outcomes of this choice are discussed in Chapter 5.

4.2.1 Training and Evaluation

The network is implemented in Python through the Keras framework, with backend in Tensorflow, in the Ubuntu 16.04 operating system. The hardware used is an Intel Core i7-7700 processor with 8 cores of 3.6GHz and an Nvidia GPU GeForce GTX 1050 Ti. The training parameters of the network are presented in Table 2.

Table 2: Training parameters for Grasping CNN

Cost function	Mean Squared Error
Optimization algorithm	Nadam
Learning rate	0.002
Weight initialization	Xavier (GLOROT; BENGIO, 2010)
Bias initialization	0
Number of epochs	30
Batch size	144

The dataset is divided into training, validation, and test sets, and the number of samples in each of them is presented in Table 3. Another division performed in the dataset comprises the type of images contained in the above mentioned sets. In the case called *Image-Wise* the figures are randomly organized into sets. In the case *Object-Wise*, all images containing the same object are placed in the same set.

Table 3: Grasping dataset division

Set	Training	Validation	Test	Total
Image-Wise	250,211	2,527	44,600	297,338
Object-Wise	251,951	2,545	42,842	297,338

The evaluation of the result is made with the following metric, as used by default in related works of grasp detection by means of the rectangle representation:

- the difference between the angle θ of the predicted rectangle (r_p) and the correct rectangle (r_c) must be within 30 degrees, and
- the Jaccard index of the predicted rectangle relative to the correct one must be greater than 0.25. This index represents the percentage of overlap between the rectangles, according to the following formula:

$$J(r_p, r_c) = \frac{r_p \cap r_c}{r_p \cup r_c} > 0.25. \tag{4.4}$$

Since each object has more than one labeled rectangle, the network prediction is considered correct if the predicted rectangle satisfies the two constraints for at least one of the labels. The reason for using half the limit commonly used for object detection (0.5) is that the labeled rectangles are not exhaustive. Besides, in practical scenarios (not applicable in precision industry), a rectangle with a Jaccard index of 0.25 and 30 degrees offset in orientation generally enables a successful grasp (REDMON; ANGELOVA, 2015).

4.3 Visual Servoing Dataset

To train a model that can perform visual servoing on different target objects, without the need to design features, it is necessary to have a dataset that efficiently captures the attributes of the environment in which the robot operates, be representative of the VS task and diverse enough to ensure generalization. To this end, the data is collected by the robot itself (Kinova Gen3 - see Section 4.5) in a way that approximates the self-supervised approach. Human interventions are associated with the assembly of the workspace and the setup of the robot, which involves determining a reference pose from which the robot captures the images and labels them.

The robot is programmed to assume different poses from a Gaussian distribution centered in the reference pose, with different standard deviations. This approach is inspired by the work of Bateux et al. (2018), which do the same, but using virtual cameras and homography instead of a real environment. The reference pose (mean of the distribution) and the sets of Standard Deviations (SD) assumed by the robot are presented in Table 4.

Table 4: Gaussian distribution parameters to build the VS dataset

Dimension		Mean	Standard Deviation		
			High	Mid	Low
Position (meters)	x	0.288	0.080	0.030	0.010
	y	0.344	0.080	0.030	0.010
	z	0.532	0.080	0.030	0.005
Orientation (degrees)	α	175.8	5.0	2.0	1.0
	β	-5.5	5.0	2.0	1.0
	γ	90.0	5.0	2.0	1.0

The SD choices take into account the expected displacement values that the robot must perform during the VS. In this way, the images obtained from a high SD help the network to understand the resulting changes in the image space when a large displacement is made by the robot. The instances obtained from a low SD enable the reduction of the error between the reference image and the current one when they are very close, for a good precision in steady state. The mean SD values help the network to reason during most of the VS execution. Two dataset instance examples and their respective label are illustrated in Fig. 22.

Figure 22: Instance examples from the VS dataset. Generated from a Gaussian distribution with mean in the reference pose $[x, y, z, \alpha, \beta, \gamma]$: (0.228, 0.344, 0.532, 175.8, -5.5, 90.0) (position in meters, orientation in degrees).

(a) Image taken from camera in pose (0.326, 0.356, 0.503, 178.0, 1.1, 91.5)

(b) Image taken from camera in pose (0.258, 0.207, 0.402, -175.8, -23.0, 87.2)

Source: Author

Concerning the choice of objects and their distribution in the workspace to build the dataset, two factors were considered. First, the network must learn the features that make it possible to correlate a reference image to the current one, therefore, the greater the number of visual cues in the dataset images, the better the network's learning ability. Thus, some scenes comprise a large number of objects, with varied structures, scattered across the table (as in Fig. 22). Other scenes are constructed so that the network learns the task in which it will be applied, that is, grasping. In this way, only one object is placed on the table, so that little visual information is available, but a good representation of the task is obtained.

In addition to single and multiple objects, two other types of scenes were considered. A framed map with a reflective surface, so that the network is robust to distortions in the captured image, and sets of books so that the network learns relevant features in flat objects. Brightness changes were considered during the dataset generation, since it was captured at different times of the day, without the concern of keeping the luminosity constant. In addition, shadows in random regions of the image were considered, since the environment was not controlled and susceptible to intermittent transit of people. The only post-processing performed on the obtained images is the exclusion of those that do not contain any part of the object. Table 5 shows the dataset composition in details.

After obtaining the data, the dataset is structured in the form $(I, [x, y, z, \alpha, \beta, \gamma])$, in which I is the image, and $[x, y, z, \alpha, \beta, \gamma]$ is the associated camera pose when this image was captured. In order to use this dataset to train a Position Based VS neural network, two images and the relative pose between them must be considered. Then, each instance of the processed dataset takes the form $(I_d, I_c, {}^d\mathbf{H}_c)$, in which I_d is a random instance chosen as the desired image, I_c is another instance chosen as current image, and ${}^d\mathbf{H}_c$ is the

Table 5: VS dataset: Composition, labels generation and division

Description	Composition				Generation	Division	
	Scenes	High SD	Mid SD	Low SD		Training	Validation
Multiple objects	4	767	630	600	361,315	289,052	18,065
Single objects	13	877	910	780	161,625	129,300	8,081
Books	2	208	210	180	62,935	50,348	3,146
Framed map	1	140	140	120	50,470	40,376	2,523
Total	**20**	**1,992**	**1,890**	**1,680**	**636,345**	**509,076**	**31,817**

transformation that relates the current frame to the desired camera frame. This is done by expressing each pose, represented by translations and Euler angles, in an homogeneous transformation matrix form ($^0\mathbf{H}_d$ and $^0\mathbf{H}_c$), and then obtaining $^d\mathbf{H}_c = {}^0\mathbf{H}_d^{-1}\,{}^0\mathbf{H}_c$.

Finally, for the network to be, in fact, a controller, the intention is that its prediction is directly the velocity signal of the camera, *i. e.* the control signal. So, the data structured as $(I_d, I_c, {}^d\mathbf{H}_c)$ is transformed to $(I_d, I_c, \mathbf{v_c})$, in which $\mathbf{v_c}$ is the proportional camera velocity. The proportional term is used because the λ gain is not considered in determining the labeled velocity, and must be tunned *a posteriori*, during the control execution. The velocity $\mathbf{v_c}$ is obtained from $^d\mathbf{H}_c$ using Eqs. 2.30, 2.31 and 2.36 (not considering λ).

The final number of instances generated for network training and validation is given by Eq. 4.5.

$$N_{ins} = \sum_{i=1}^{20} h_i l_i + m_i l_i + C_{l_i,2} + C_{m_i,2} \tag{4.5}$$

In this equation, N_{ins} is the total number of generated instances in the form $(I_d, I_c, \mathbf{v_c})$, i is the considered scene (since I_d and I_c must be from the same scene) and h_i, m_i and l_i are, respectively, the number of images obtained from a high, medium and low standard deviation. $C_{l_i,2}$ and $C_{m_i,2}$ is the total number of combinations of two between the images obtained with low and medium standard deviations, respectively, as given by Eq. 4.6.

$$C_{n,p} = \frac{n!}{p!(n-p)!} \tag{4.6}$$

These I_d and I_c choices were made to ensure that there is overlap between the considered images. Thus, combinations between High SD and Mid SD images, as well as combinations of two in the High SD set, were not considered. The details of this generation and the division of the final dataset are presented in Table 5.

The number of instances in the training and validation sets for each type of scene is estimated. Only the total values of each set are correct, as the division is made considering 80% and 5% of the entire dataset. The remaining 15% were not used.

4.3.1 7-Scenes Dataset

The 7-scenes dataset (BRACHMANN et al., 2016) is a set of images for pose estimation, so the label is represented by a homogeneous transformation matrix describing the absolute pose related to some reference frame when the camera took the image. Some examples of a sequence in one of the scenes of the dataset are shown in Fig. 23.

Figure 23: Example instances from the chess sequence of the 7-scenes dataset

Source: Author (using instances of the dataset)

Using the same approach as in the generation of the dataset using the images collected by the robot, 2,414,400 $(I_d, I_c, \mathbf{v_c})$ instances were generated from the 7-scenes images. The overlap was guaranteed since each image was combined considering only the 60 subsequent ones. Part of the dataset is used to pre-train the VS CNN and to analyze the effects of this pre-training and later fine-tuning, with the built dataset, in terms of accuracy and training speed.

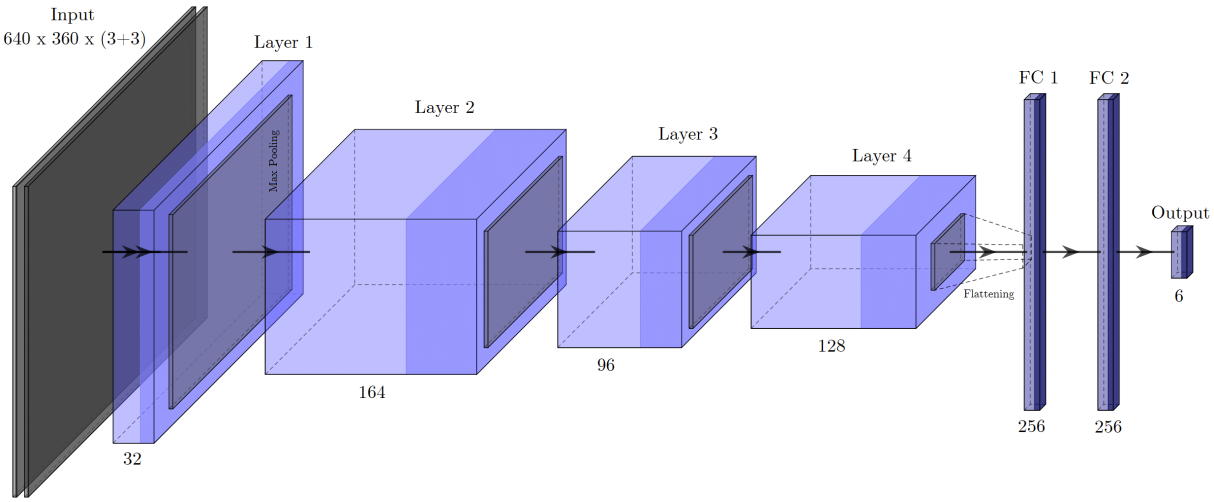
4.4 Visual Servoing Network Architectures

Unlike grasping, the network designed to perform visual servo control of a robotic manipulator receives two images as input and must regress six values, which can also be categorized into two outputs, considering linear and angular camera velocities. In this way, four network architectures are designed and evaluated according to the ability to process two inputs and two sets of outputs. As one of the architectures is used to create two models, by replacing an association operator, in the end 5 models are presented to approach the VS task.

All networks, however, are generated from the remarkably simple structure of the grasping network. In this way, the number of filters and the operations considered in each convolutional layer, the number of FC layers and the respective number of neurons, as well as the dropout rate are similar to the grasping network and common to all models developed. All networks also receive two image inputs in the format $640 \times 360 \times 3$ and outputs the 6-dimensional velocity vector without any intermediate step.

The first model for the VS task, called *Model 1 - Direct Regression*, is presented in Fig. 24. It is basically identical to the grasping network, except for the inclusion of max-

Figure 24: Model 1 - Direct regression



Source: Author

pooling in the third convolutional layer and for the different input dimensions, which causes the same proportional difference on the feature maps. The inputs are simply concatenated and the data flow inside the network is always ahead, without any association or division of feature maps.

The second model, called *Model 2 - Task-specific Regression*, is presented in Fig. 25. The network inputs are concatenated and the third set of feature maps is processed by two separate layers sequences. Thus, the 6D velocity vector is predicted by the network in the form of two 3D vectors. Specifically, this structure consists of a shared encoder and two specific decoders - one for linear velocity and another for angular velocity.

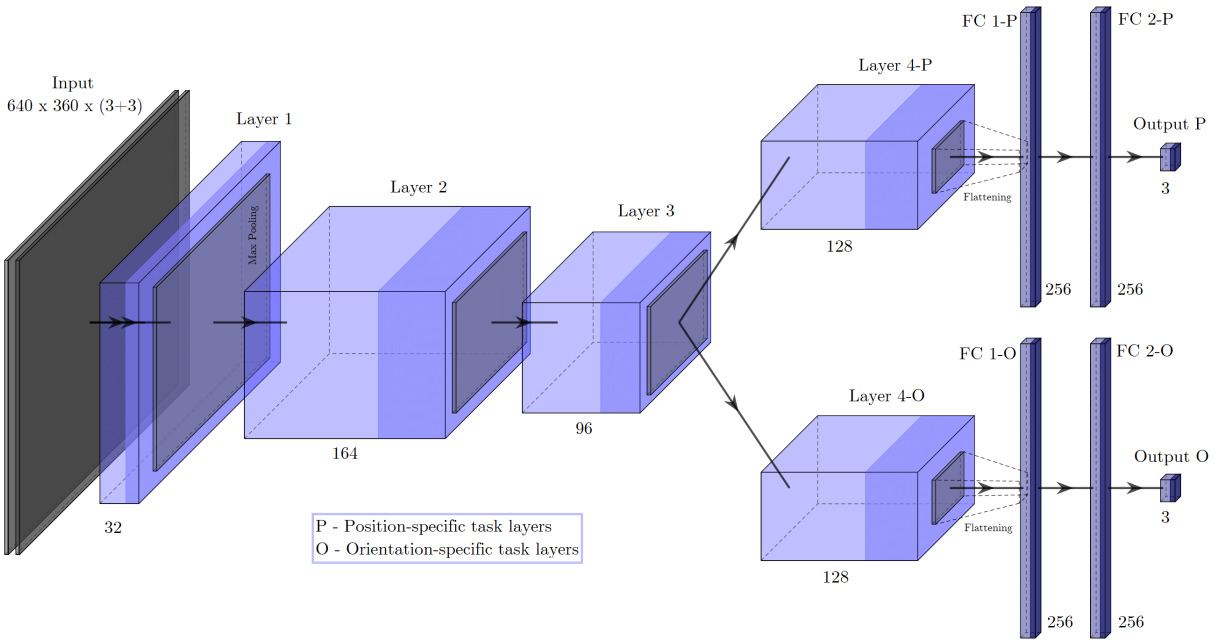
In this way, the particularities of each domain can be learned by the network at the same time that its similarities can help in generalization. This structure is known as a multi-task network and, as presented by Kendall, Gal and Cipolla (2018), the cues from one task can be used to regularize and improve generalization of another one, by using inductive knowledge transfer. The intention is to evaluate how the dissociation of the velocity control problem in two specific tasks of linear and angular velocities control, can affect the network accuracy.

The third and fourth models, called *Model 3 - Direct Regression from Concatenated Features* and *Model 4 - Direct Regression from Correlated Features*, are created using the architecture presented in Fig. 26. Unlike architecture 2, this structure has two encoders and one decoder, which makes it possible to obtain high-level representations of each image before associating them. Two different association operators (Σ) were considered: a simple concatenation, which defines Model 3, and a correlation layer, used in the work of Dosovitskiy et al. (2015), which defines Model 4.

Model 3 simply considers the concatenation of the feature maps resulting from the

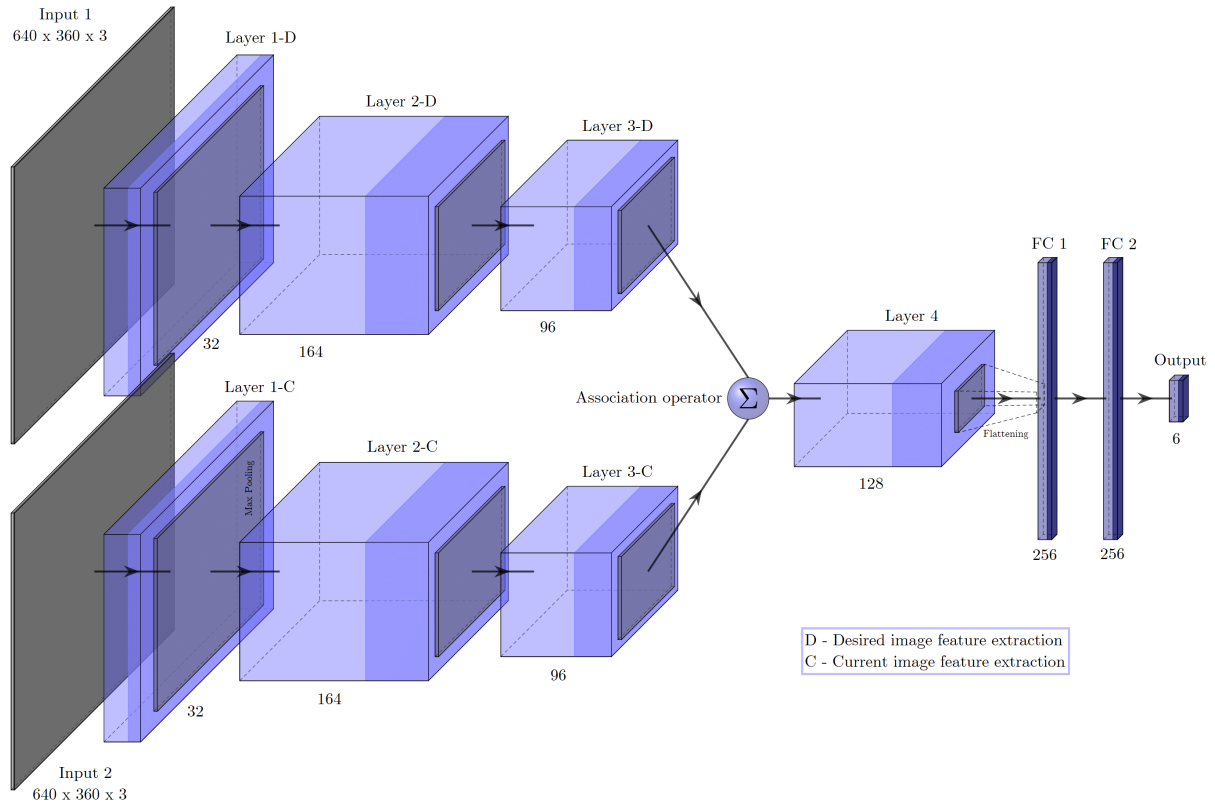
third convolutional layer, so that the input to the fourth layer is twice as deep. Model 4, on the other hand, has a correlation layer that aids the network to find correspondences

Figure 25: Model 2 - Task-specific regression



Source: Author

Figure 26: Models 3 and 4 - Direct regression from associated features



Source: Author

between the feature representations of each image. The original correlation layer, part of the optical flow network FlowNet (DOSOVITSKIY et al., 2015), is defined by Eq. 4.7.

$$c(\mathbf{x_1}, \mathbf{x_2}) = \sum_{\mathbf{o} \in [-k,k] \times [-k,k]} \langle \mathbf{f_1}(\mathbf{x_1} + \mathbf{o}), \mathbf{f_2}(\mathbf{x_2} + \mathbf{o}) \rangle \tag{4.7}$$

In the equation, $\mathbf{f_1}$ and $\mathbf{f_2}$ are two feature maps, whose correlation of two patches centered in $\mathbf{x_1}$ in the fist map and $\mathbf{x_2}$ in the second map is $c(\mathbf{x_1}, \mathbf{x_2})$ for a square patch of size $K := 2k + 1$. However, comparing all patch combinations makes the training intractable, so the authors propose modifications in the equation which results in a more simple correlation layer, adapted from the definition of Rocco, Arandjelovic and Sivic (2019), as Eq. 4.8.

$$c(i, j) = \mathbf{f_1}(i, j)^T \mathbf{f_2}(i + k_1, j + k_2) \tag{4.8}$$

This equation says that at spatial location (i, j) at the correlation map contains all the similarities between spatial location (i, j) at $\mathbf{f_1}$ and all spatial positions in the proximity of (i, j) at $\mathbf{f_2}$, given by $k_1, k_2 \in [-k + 2, k - 2]$. Thus, the resulting correlation map has the same dimensions as the input feature maps, with a depth equal to k^2, where k is the size of the square neighborhood considered. The size considered in the implemented architecture is 18 with stride 2, which is the same as using size 9, thus resulting in a correlation map with 81 feature maps. Table 6 shows the output dimensions and number of parameters of each layer for all developed models.

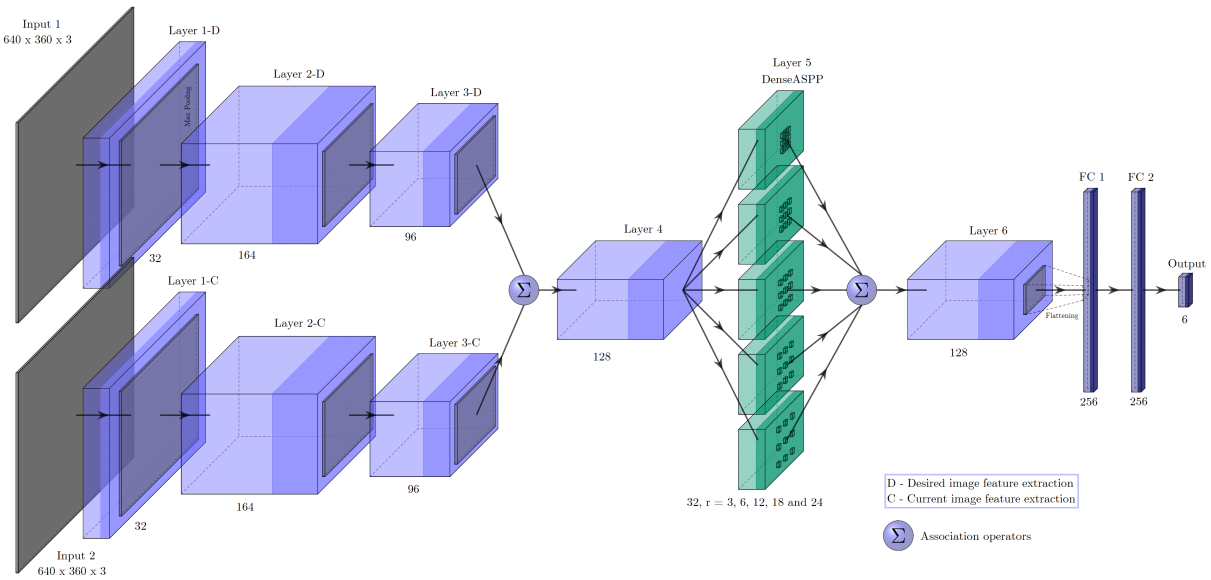
The last model considered, called *Model 5 - Direct Regression from Associated Features with ASPP*, is presented in Fig. 27.

This structure is identical to Architecture 3 up to the fourth convolutional layer, using concatenation as the association operator. Then the resulting feature map, instead of being flattened for FC layers, is first processed by a Dense Atrous Spatial Pyramid Pooling (ASPP) module followed by another convolutional layer.

Atrous convolution (CHEN et al., 2014) is able to achieve a larger receptive field using a dilated kernel with zeros in appropriate positions. In this way, a feature map produced from atrous convolution can be as the same size as the input and can encode higher-level semantics, without the cost of more trainable parameters (YANG et al., 2018). However, the atrous-convolved feature map is built from a single scale, which motivated the creation of an ASPP module (CHEN et al., 2017), a method to concatenate atrous-convolved feature maps with different dilation rates, to encode multiscale information and eventually improve accuracy.

Later, a variation of the ASPP module, called DenseASPP, was proposed by Yang et al. (2018). It uses dense connections to make the output of each layer of atrous convolution

Figure 27: Model 5 - Regression from associated features with ASPP



Source: Author

available to all layers with different dilation rates ahead. Thus, unlike ASPP, which needs high dilation rates to achieve larger receptive fields and ends up losing its modeling power, DenseASPP can obtain larger receptive fields, without kernel degradation, using only reasonable dilation rates (YANG et al., 2018).

In the architecture proposed, dilation rates of 3, 6, 12, 18 and 24 are used. The resulting feature map is processed by a last convolutional layer with max-pooling to reduce its dimensions before flattening.

All models were developed based on the assumption of simplicity and the reduced number of parameters, as can be seen in Table 6. Only models 4 and 5 have more complex operations, which are still used to process feature maps without large dimensions and contribute little in decreasing the network lightness.

4.4.1 Training and Evaluation

The networks are implemented in Python through the framework Keras, with backend in Tensorflow, in the Ubuntu 16.04 operating system. The hardware used is an Intel Core i7-7700 processor with 8 cores of 3.6GHz and an Nvidia GPU GeForce GTX 1050 Ti. The training parameters of the networks are presented in Table 7.

For evaluation, the mean squared error is used as the performance metric, according to Eq. 4.9, where n is the number of instances in the validation set, $\mathbf{v_c^l}$ is the labeled camera velocity and $\mathbf{v_c^p}$ is the network prediction.

$$E_{mse} = \frac{1}{n}\sum_{k=1}^{n}(\mathbf{v_c^l} - \mathbf{v_c^p})^2 \tag{4.9}$$

Table 6: Output dimensions and number of parameters of the VS CNN models

Model 1					
Layer	Output dimensions	Parameters	Max Pooling	$20 \times 11 \times 96$	0
Convolutional Layer 1	$320 \times 180 \times 32$	1760	Convolutional Layer 4	$18 \times 9 \times 128$	110,720
Normalization	$112 \times 112 \times 32$	128	Max Pooling	$9 \times 4 \times 128$	0
Max Pooling	$160 \times 90 \times 32$	0	Flattening	1×4608	0
Convolutional Layer 2	$80 \times 45 \times 164$	47,396	Fully Connected 1	1×256	1,179,904
Normalization	$80 \times 45 \times 164$	656	Fully Connected 2	1×256	65,792
Max Pooling	$40 \times 22 \times 164$	0	Output	1×6	1,285
Convolutional Layer 3	$40 \times 22 \times 96$	141,792			
Total					1,549,690

Model 2					
Layer	Output dimensions	Parameters	Max Pooling	$20 \times 11 \times 96$	0
Convolutional Layer 1	$320 \times 180 \times 32$	1760	2× Convolutional Layer 4	$18 \times 9 \times 128$	221,440
Normalization	$112 \times 112 \times 32$	128	2× Max Pooling	$9 \times 4 \times 128$	0
Max Pooling	$160 \times 90 \times 32$	0	2× Flattening	1×4608	0
Convolutional Layer 2	$80 \times 45 \times 164$	47,396	2× Fully Connected 1	1×256	2,359,808
Normalization	$80 \times 45 \times 164$	656	2× Fully Connected 2	1×256	131,584
Max Pooling	$40 \times 22 \times 164$	0	Output 1	1×3	771
Convolutional Layer 3	$40 \times 22 \times 96$	141,792	Output 2	1×3	771
Total					2,906,106

Model 3					
Layer	Output dimensions	Parameters	2× Max Pooling	$20 \times 11 \times 96$	0
2× Convolutional Layer 1	$320 \times 180 \times 32$	1792	Concatenate	$20 \times 11 \times 192$	0
2× Normalization	$112 \times 112 \times 32$	256	Convolutional Layer 4	$18 \times 9 \times 128$	221,312
2× Max Pooling	$160 \times 90 \times 32$	0	Max Pooling	$9 \times 4 \times 128$	0
2× Convolutional Layer 2	$80 \times 45 \times 164$	94,792	Flattening	1×4608	0
2× Normalization	$80 \times 45 \times 164$	1312	Fully Connected 1	1×256	1,179,904
2× Max Pooling	$40 \times 22 \times 164$	0	Fully Connected 2	1×256	65,792
2× Convolutional Layer 3	$40 \times 22 \times 96$	283,584	Output	1×6	1542
Total					1,850,286

Model 4					
Layer	Output dimensions	Parameters	2× Max Pooling	$20 \times 11 \times 32$	0
2× Convolutional Layer 1	$320 \times 180 \times 32$	1792	Correlation Layer	$20 \times 11 \times 81$	0
2× Normalization	$112 \times 112 \times 32$	256	Convolutional Layer 4	$18 \times 9 \times 128$	93,440
2× Max Pooling	$160 \times 90 \times 32$	0	Max Pooling	$9 \times 4 \times 128$	0
2× Convolutional Layer 2	$80 \times 45 \times 164$	94,792	Flattening	1×4608	0
2× Normalization	$80 \times 45 \times 164$	1312	Fully Connected 1	1×256	1,179,904
2× Max Pooling	$40 \times 22 \times 164$	0	Fully Connected 2	1×256	65,792
2× Convolutional Layer 3	$40 \times 22 \times 32$	94,528	Output	1×6	1542
Total					1,533,358

Model 5					
Layer	Output dimensions	Parameters	Dense ASPP Layer, r=3	$18 \times 9 \times 160$	26,976
2× Convolutional Layer 1	$320 \times 180 \times 32$	1792	Dense ASPP Layer, r=6	$18 \times 9 \times 224$	29,664
2× Normalization	$112 \times 112 \times 32$	256	Dense ASPP Layer, r=12	$18 \times 9 \times 320$	34,016
2× Max Pooling	$160 \times 90 \times 32$	0	Dense ASPP Layer, r=18	$18 \times 9 \times 448$	40,544
2× Convolutional Layer 2	$80 \times 45 \times 164$	94,792	Dense ASPP Layer, r=24	$18 \times 9 \times 32$	132,224
2× Normalization	$80 \times 45 \times 164$	1312	Convolutional Layer 6	$16 \times 7 \times 128$	36,992
2× Max Pooling	$40 \times 22 \times 164$	0	Max Pooling	$8 \times 3 \times 128$	0
2× Convolutional Layer 3	$40 \times 22 \times 32$	94,528	Flattening	$1 \times 3,072$	0
2× Max Pooling	$20 \times 11 \times 32$	0	Fully Connected 1	1×256	786,688
Concatenate	$20 \times 11 \times 64$	0	Fully Connected 2	1×256	65,792
Convolutional Layer 4	$18 \times 9 \times 128$	73,856	Output	1×6	1542
Total					1,420,974

For purposes of illustration, predictions of all models for the same validation instance are implemented as a velocity command in the robot. No feedback is considered

Table 7: Training parameters for the VS networks

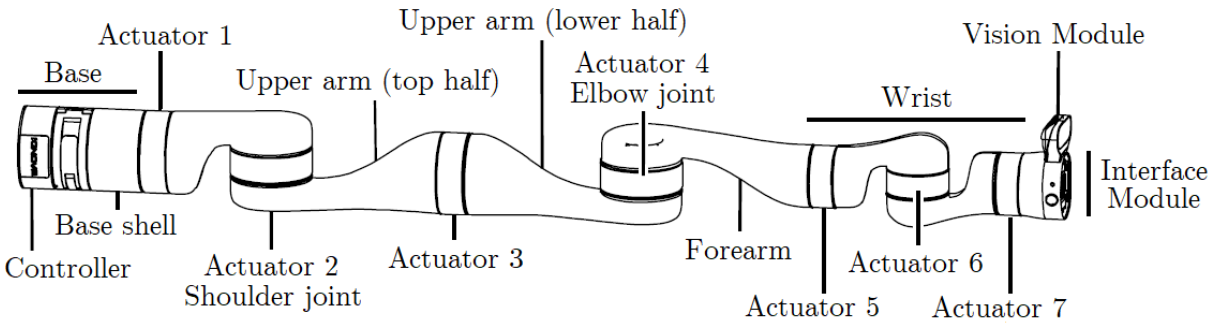
Cost function	Mean Squared Error
Optimization algorithm	Adam
Learning rate	0.001
Weight initialization	Xavier
Bias initialization	0
Number of epochs	10
Batch size	32 (24 for Model 5)

since the intention is only to show the effect of applying the velocity predicted by the network in open-loop, so that the mean squared error of this prediction is better understood in terms of the difference between poses.

4.5 KINOVA Gen3 7 DoF Robotic Manipulator

To build the visual servoing dataset and evaluate the performance of the developed algorithms, the Kinova Gen3 7 DoF robotic manipulator is used. The robot has a built-in vision module and is equipped with a two-finger parallel gripper. Its main components are shown in Fig. 28.

Figure 28: Kinova Gen 3's main components: 7 degrees-of-freedom, 1 kHz controller, vision module with color and depth sensors, and interface module compatible with Robotiq 2F-85



Source: Adapted from Kinova Robotics (2019)

The dimensions and Denavit-Hartenberg parameters, indicating the relationship between the base frame (reference) and the tool frame, are presented on Annex A. The robot has numerous features, such as wired controller, web monitoring interface, cartesian and null space admittance control, singularity and collision avoidance, among many others. The most relevant specifications and capabilities for the considered task are detailed in Table 8, whilst others can be found in the User Guide (KINOVA ROBOTICS, 2019).

The Kinova software framework and application development platform, KORTEX, makes it possible to programmatically configure and control the robot according to the

Table 8: Gen 3's relevant specifications and capabilities (KINOVA ROBOTICS, 2019)

Specification	**Description**
Camera	Omnivision OV5640 - 1280 × 720 @ 15/30 fps
Mass	8.2 kg (without gripper)
Payload	3.1 kg (mid-range continuous, with gripper)
Maximum reach	902 mm
Control	100 Mbps Ethernet for real-time 1kHz control
Actuators readings available	Current, temperature, voltage, torque, position and velocity
Tool readings available	Pose and twist
Gripper readings available	Temperature, voltage, current, position and velocity
High-level control	Cartesian twist and wrench, and joint speed

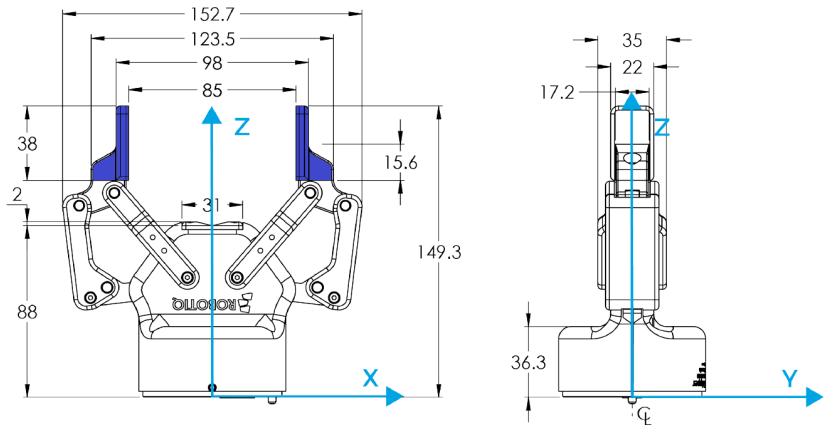
particularities of a grasping system. The API is used in Python language to send Cartesian pose, camera velocity, and gripper commands, as well as to take sensor readings and have access to the camera stream.

The robot is controlled using Python 3.6.7 in the Windows 10 operating system. The KORTEX API version used in the development of the algorithms is 2.0.0 and the firmware versions for all actuators and vision module is 2.0.1-72. The trained networks are implemented in the same environment, and the hardware available is an Intel Core i7-7500U processor with 2.9GHz and an Nvidia GPU GeForce 940MX.

4.5.1 Robotiq 2F-85 Gripper

To execute grasp, the robot is mounted with a Robotiq 2F-85 parallel gripper, illustrated in Fig. 29. The gripper has two articulated fingers with two phalanxes each, which can engage up to five points of contact, and can adapt to the shape of the object since the fingers are under-actuated. So, picking the same object could result in either an encompassing grasp or a parallel grasp, but only the last is considered in this work,

Figure 29: Robotiq 2F-85 Gripper's dimensions with fingertips highlighted in blue



Source: Adapted from Robotiq Inc. (2019)

Table 9: Gripper's mechanical specifications (ROBOTIQ INC., 2019)

Specification	**Description**
Gripper opening	85 mm
Maximum height	162.8 mm (closed)
Mass	925 g
Grasp force	20 to 235 N
Finger speed	20 to 150 mm/s
Position repeatability	0.05 mm
Position resolution	0.4 mm

meaning that if the gripper performs a successful encompassing grasp, it is considered as a failure anyway.

The gripper control interface is encapsulated by the KORTEX API, so it is not necessary to have two different environments to actuate on the robot and the gripper. The main mechanical specifications are presented in Table 9.

4.6 Robot Configuration

Once the networks are trained, the robot is ready to be controlled, the communication with KORTEX API is adjusted and the workspace is established, it is possible to implement the grasping system in the robot. As shown in Fig. 17, visual servoing and grasp detection are continuously executed until the control signal norm is below a threshold. Only the VS network prediction is actually sent to the robot during the execution of the main loop, whereas the grasp network prediction is simply shown to the user, until the stop condition is satisfied.

The velocity command applied to the robot's camera during the control is given by Eq. 4.10, where $net_{VS}(\mathbf{I_d}, \mathbf{I_c})$ is the network prediction considering the current image obtained by the camera, $\mathbf{I_c}$, and the reference image, $\mathbf{I_d}$, and λ is the proportional gain, tunned as 0.05 and 2 for linear and angular velocities respectively. This velocity signal is applied in the tool reference frame.

$$\mathbf{v}_c = -\lambda net_{VS}(\mathbf{I_d}, \mathbf{I_c}) \tag{4.10}$$

When the control is interrupted and the grasp is executed, the grasp network prediction is used to guide the robot to the target object. The coordinates of the rectangle's central point are given by the first two values of the vector predicted by the network, as Eq 4.11 shows.

$$(x_c, y_c) = net_G(\mathbf{I_c})[1, 2] \tag{4.11}$$

These coordinates are expressed in pixels and, as presented in subsection 2.2.2.1, they must be mapped to the world reference using the Eqs. 2.22 and 2.23. Replacing the values of the robot's camera intrinsic parameters, Eq. 2.22 is rewritten as

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \left(\begin{bmatrix} -1297 & 0 & 620.9 \\ 0 & -1298 & 238.2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \right)^{-1} \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}, \tag{4.12}$$

in which the extrinsic parameters matrix, within parentheses, can be obtained by removing the third column and fourth row (since depth information is lost and the points are on the same plane) of

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} R_z(\gamma)R_y(\beta)R_x(\alpha) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.05639 \\ 0 & 0 & 1 & -0.12275 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right)^{-1}, \tag{4.13}$$

where $R_z(\gamma)R_y(\beta)R_x(\alpha)$ is the matrix representation of tool's orientation as a sequence of three Euler angles using $z-y-x$ Tait-Bryan extrinsic convention (Eq. 4.14). And the last matrix represents the camera frame related to the tool frame.

$$R_z(\gamma)R_y(\beta)R_x(\alpha) = \begin{bmatrix} C\gamma C\beta & C\gamma S\beta S\alpha - C\alpha S\gamma & S\gamma S\alpha + C\gamma C\alpha S\beta & 0 \\ C\beta S\gamma & C\gamma C\alpha + S\gamma S\beta S\alpha & C\alpha S\gamma S\beta - C\gamma S\alpha & 0 \\ -S\beta & C\beta S\alpha & C\beta C\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

Finally, the points that the robot's gripper must reach are given by $X = \frac{X'}{Z'}$, $Y = \frac{Y'}{Z'}$ and Z fixed at 23mm. The gripper opening is given by Eq. 4.15, where ξ is an empirically obtained factor considering the gripper's maximum opening and the scale of the command signal, where 1 means fully closed. The fourth element of the network prediction is not used since the gripper has a fixed height of 22mm, and lastly, the orientation is given by Eq. 4.16.

$$w = 1 - \xi net_G(\mathbf{I_c})[3] \tag{4.15}$$

$$\theta = 90 - net_G(\mathbf{I_c})[5] \tag{4.16}$$

5 RESULTS AND DISCUSSION

This chapter presents all the results obtained with the developed algorithms and analyzes them in two different ways - considering the tasks of grasp detection and visual servoing in isolation and the joint application in the dynamic grasping system.

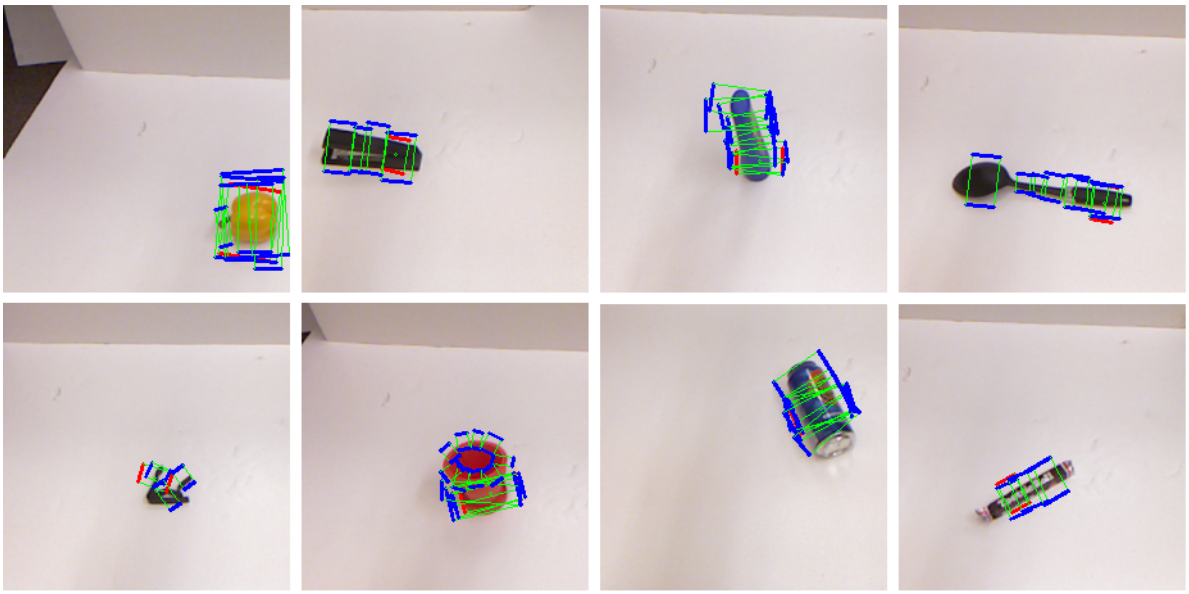
5.1 Offline Results

This section presents the results achieved for grasping and visual servoing from the dataset validation sets developed for each task.

5.1.1 Grasp Detection

The visual results of the developed grasp detection algorithm are illustrated in Fig. 30. It is possible to note the network's ability to predict a grasp rectangle consistent with the object's shape even if it requires a small gripper opening, or choose from several options of grasp points.

Figure 30: Grasp network prediction (red) and labeled rectangles (blue)

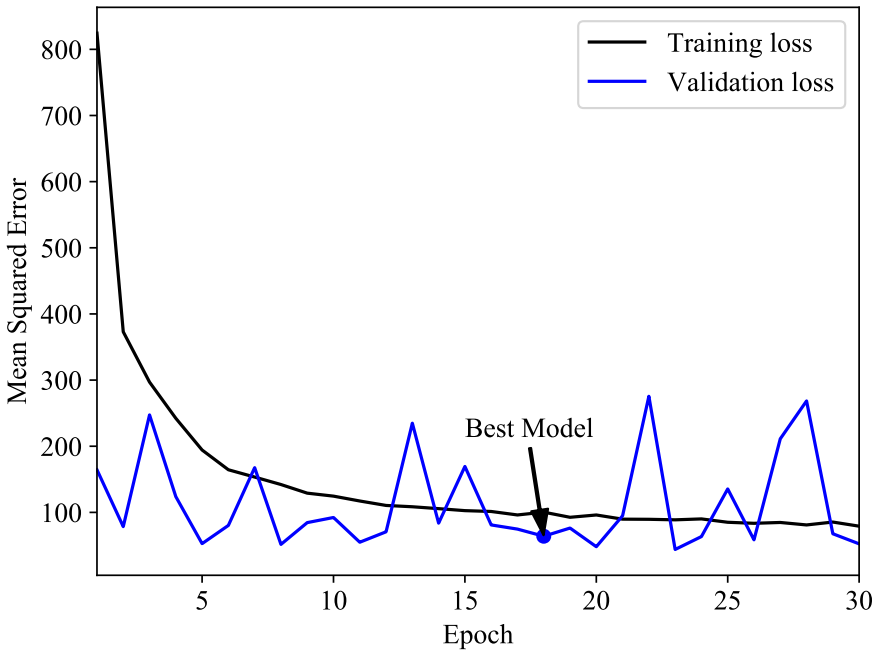


Source: Author

Several grasp rectangles are associated with each image of the dataset. Therefore, the way training is conducted must be chosen between using all of them as labels or using only one rectangle, chosen at random, as labeled grasp. In the first case, we have a more reliable representation of the different grasp possibilities in each object. However, for a direct regression network, the prediction tends to the mean of the labeled rectangles, since only one can be predicted. The consequence of this choice involves, for example,

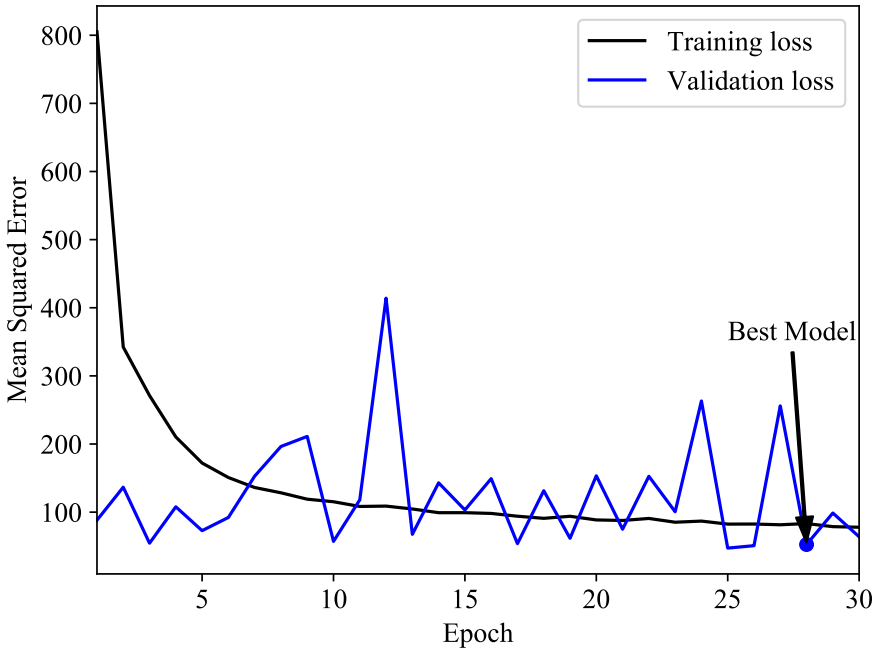
the prediction of a grasp rectangle in the middle of a circular object, since its labels are arranged on its circumference. This setting makes it impossible for the object to be grasped, therefore only one rectangle is chosen, at random, as label.

Figure 31: Training and validation losses using image-wise division (validation ×10)



Source: Author

Figure 32: Training and validation losses using object-wise division (validation ×10)



Source: Author

The networks took an average of one hour per epoch to be trained. As the initial-

ization of the network weights plays an important role in the gradient descent convergence, five training sessions were carried out for each division (image-wise and object-wise). Considering that each session is composed of 30 epochs, the total time spent on training is 300 hours. The graphics of the two training sessions that led to the best models, according to the evaluation metric presented in Chapter 4, are illustrated in Figs. 31 and 32.

It can be seen that the training error evolves as expected, with more significant falls in the first epochs and subsequent accommodation, but always assuming decreasing values. The validation error, however, does not follow any default, over-fitting or under-fitting pattern, and are much larger than training error. This behavior can be understood from the fact that only one rectangle is chosen as label.

During the validation stage of each training epoch, the network prediction is compared with just one labeled rectangle. In other words, even though the network prediction is a valid grasp rectangle, it cannot be said that it coincides with the one chosen as label. So, the validation error assumes a random behavior, even though the network is obtaining generalization capacity.

For the choice of the best model, which is used in the operational phase, as well as in the statistical survey of the test data, the metric used is the Jaccard index plus the difference in orientation. Therefore, the mean squared error of the validation set is not really relevant for the analysis of the network.

The network weights are recorded at each epoch, thus enabling the test data to be evaluated by the resulting parameterized network after each epoch. During this evaluation, the prediction is compared with all labeled rectangles, to avoid the problem of matching between prediction and label, which occurs in the validation set.

Table 10 presents the accuracy obtained in this work and compares it with those found by related works. The result of 94.8% accuracy in the image-wise division is the third best among the listed works and the 86.9% percentage for object-wise is in sync with the publications, except for the works of Chu, Xu and Vela (2018) and Zhou et al. (2018).

In general, it is seen that the network is quite efficient given the simplicity of its architecture. With only four convolutional layers and two FC layers, similar or higher accuracy can be achieved when compared with works that use significantly larger networks. This property leads to the main benefit of the proposed method: prediction speed. Moreover, many of the listed works also use depth information in addition to RGB or replacing one of the color channels, making their application inflexible to scenarios where such information cannot be obtained.

The work with the lowest reported prediction time is the one developed by Park and Chun (2018), with 23 ms, whereas the network presented here has mean prediction time in the order of 13.5 ms. The high prediction speed makes it possible to use the network

Table 10: Prediction accuracy and speed on the Cornell Grasp Dataset

Approach	*Image-Wise*	*Object-Wise*	*Speed* (fps)
Jiang, Moseson and Saxena (2011)	60.5%	58.3%	0.02
Lenz, Lee and Saxena (2015)	73.9%	75.6%	0.07
Redmon and Angelova (2015)(1)	84.4%	84.9%	13.15
Redmon and Angelova (2015)(2)	88.0%	87.6%	13.15
Wang et al. (2016)	81.8%	- .	7.10
Asif, Bennamoun and Sohel (2017)	88.2%	87.5%	-
Kumra and Kanan (2017)	89.2%	88.9%	16.03
Oliveira, Alves and Malqui (2017)	86.5%	84.9%	-
Guo et al. (2017)	93.2%	89.1%	-
Asif, Tang and Harrer (2018b)	90.2%	90.6%	41.67
Park and Chun (2018)	89.6%	-	43.48
Chu, Xu and Vela (2018)	96.0%	96.1%	8.33
Zhou et al. (2018)	**97.7%**	**96.6%**	8.51
Zhang et al. (2018)	93.6%	93.5%	25.16
Ghazaei et al. (2018)	91.5%	90.1%	17.86
Chen, Huang and Meng (2019)	86.4%	84.7%	-
Proposed	94.8%	86.9%	**74.07**

in a real-time scenario where a robot deals with non-static objects. It is understood that, considering that most of the works exposed in Table 10 use more powerful GPUs than ours, the results found come primarily from the applied data augmentation strategy, thus enabling the use of a much simpler and faster network.

For comparison of efficiency and prediction time, the convolutional layers of the proposed network were replaced by a ResNet-50 v2 (HE et al., 2016). This network is recognized for its performance in ImageNet by reducing the challenge top-5 error by just over 3%. The result was attributed to the considerable increase in the network depth, built through residual blocks (HE et al., 2016).

The ResNet presented efficiency similar to the one found in the Image-Wise division (95.44%), but had a noticeably lower efficiency in the Object-Wise division, reaching a result slightly higher than half of that found by the proposed network (46.92%).

Due to the larger number of trainable parameters, approximately 49 million, ResNet tends to lose the generalization capacity assessed by the Object-Wise division. Although it was pre-trained on ImageNet, it was not able to reduce overfitting.

In the temporal analysis, ResNet 50 took 2 hours per epoch in the training, twice as the proposed network. Regarding the prediction time, it is noted that the high number of parameters has no proportional impact on the obtained result. However, our proposed network can still operate at approximately 3 times the speed of ResNet 50 (25.64 fps).

Regarding the difference found in the IW and OW divisions, it can be said that

the OW result expresses not only the adaptation of the network to objects never seen (which is the purpose of this division) but also to the different positions of these objects, which is the purpose of the IW division. This happens because, unlike other works that use data augmented for training and original data for testing, this work deals with a much larger test set, since a whole new dataset is generated offline from the original Cornell Grasping and only then, divided. In this way, the OW test set has images of objects never seen, but also several images that show them in different positions, causing the accuracy to naturally fall.

5.1.2 Visual Servoing

The five VS networks took an average of 6.18 hours per epoch to be trained. Unlike the grasp network, only one training session was considered for each model, due to time constraints. Considering that each session is composed of 10 epochs, the total time spent on training is nearly 309 hours. But using techniques that make the network less sensitive to weights initialization, such as batch normalization, and the use of the Xavier method to assign its values, increase the reliability that the single training session will converge.

As the visual servoing dataset is much larger than the grasp dataset, and the input images have higher resolution (640×360 to 224×224), making it impossible to use bigger batch sizes (32 compared to 144), the training time of one epoch is greatly affected. This has nothing to do with the complexity of the architectures, which are simple enough to achieve the prediction speeds exposed in Table 11.

Table 11: Prediction speed of the VS CNN models

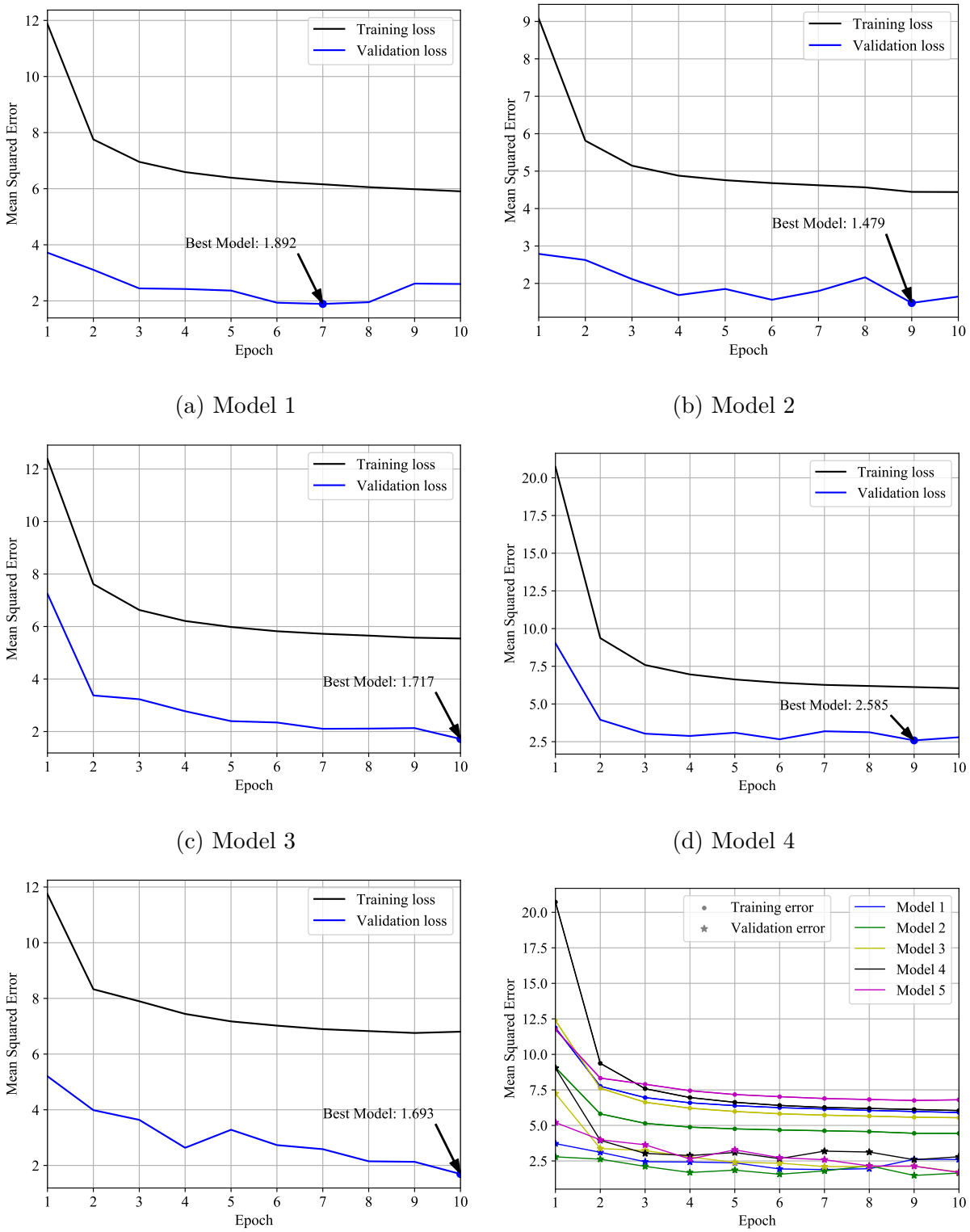
Model 1	86.77 fps
Model 2	82.49 fps
Model 3	70.12 fps
Model 4	59.86 fps
Model 5	56.35 fps

The speeds exposed in the table were obtained from the average of the prediction speed of each model for the validation data. The training and validation errors at each training epoch, as well as a comparison of the five training sessions, are shown in Fig. 33.

The mean squared error is used both as cost function and evaluation metric. In this way, the results obtained by each model in the validation set can be taken from the image. All networks were able to assimilate the features of the input images that make it possible to regress the velocity values so that these features are abstract enough to ensure generalization. Thus, in all models, the validation error is below the training error.

As can be seen in Fog. 33f, Model 2 is the one that leads to the best offline results in the visual servoing task, demonstrating that the dissociation between linear and angular

Figure 33: Training and validation losses for the five VS CNN models



(a) Model 1

(b) Model 2

(c) Model 3

(d) Model 4

(e) Model 5

(f) Models comparison

Source: Author

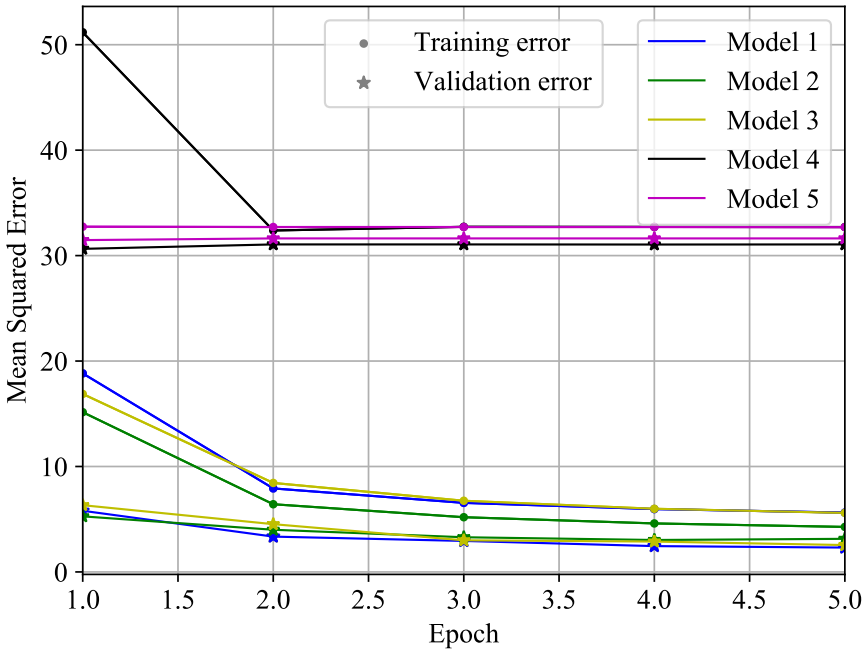
velocities and multi-task learning can, in fact, boost performance. Model 4, on the other hand, had the worst performance, which suggests that a correlation layer designed for the

optical flow task does not produce significant features for the visual servoing task.

Concerning pre-training in the 7-scenes dataset, 10 training epochs were considered for each model, plus five fine-tuning epochs in the developed dataset. The results of the fine-tuning stage, shown in Fig. 34, show that the performance of the networks is deteriorated, considering those that were trained from scratch. In addition, models 4 and 5 did not converge, presenting constant MSE, since the network learned to predict the average of the labeled velocities.

This behavior can be understood by the fact that the 7-scenes contain images of a camera that moves freely in an environment, unlike the camera in the eye-in-hand configuration. So, during the fine-tuning stage, the networks were not able to take advantage of the learning acquired with the 7-scenes and had to make major adjustments to the weights. Furthermore, according to Cavallari et al. (2019), the poses in the 7-Scenes dataset are slightly inaccurate due to tracking drift with the KinectFusion used.

Figure 34: Training and validation losses using pre-trained weights on 7-scenes dataset



Source: Author

The concept of offline results in visual servoing refers only to the difference between the predicted and labeled velocities. To get a better idea of how the MSE expresses the quality of the designed networks, the prediction of each model for an instance of the validation set is presented in Table 12. Thus, it is possible to make a quantitative comparison between the predicted and the labeled velocity values.

Going further, it is possible to use these predictions to illustrate the robot's behavior when these speeds are applied to the camera. First, the robot is positioned in the pose equivalent to the one it had when it captured the current image (I_c) of the validation

Table 12: Labeled and predicted velocities using two input images from the validation set, in which the current and desired ones are obtained at poses [0.344, 0.326, 0.372, -179.089, -3.201, 91.907] and [0.276, 0.390, 0.411, 177.734, -4.326, 90.252], respectively.

Prediction ($\times 10^{-2}$)	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{\alpha}$	$\dot{\beta}$	$\dot{\gamma}$
Label	-6.756	6.617	3.415	5.731	-1.928	-2.904
Model 1	-6.533	5.085	1.388	4.949	-2.232	-3.141
Model 2	-5.687	5.536	2.175	5.489	-2.420	-3.593
Model 3	-5.355	7.007	2.625	6.492	-2.363	-1.751
Model 4	-6.104	5.798	4.011	6.032	-1.450	-3.802
Model 5	-4.625	4.972	3.741	3.907	-0.374	-2.484

instance. Then the speed command is applied (without feedback, visual or any other type), considering $\lambda_{lin} = 0.1$ and $\lambda_{ang} = 10$, until the L1-norms of the linear and angular velocities are greater than the L1-norms of the previous iteration. When this condition is met, the applied speed is zero and the robot stops moving.This is not the actual visual servoing, which is performed in closed-loop, but rather an illustration of how the velocity predicted by the network can affect the robot's pose.

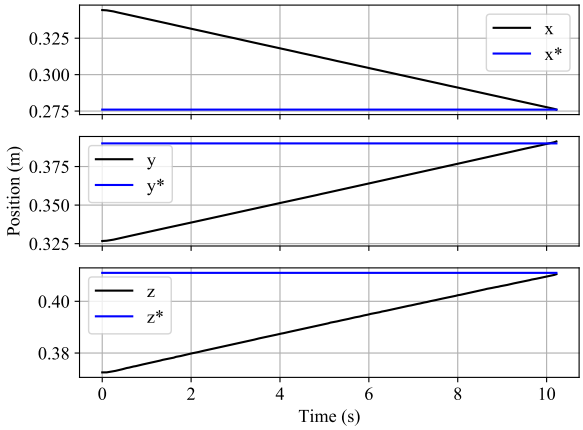
The pose of the robot is read throughout the process and illustrated graphically in Figs. 35 and 36. The best scenario is represented by the application of the labeled speed (Figs. 35a and 36a), in which the robot's final position and orientation exactly match its pose when the desired image (I_d) of the validation instance was captured.

The robot's final orientation is quite noisy at the end of the movement. This is because the linear and angular velocities are analyzed separately for the stop condition. In all cases, except for Model 5 (Figs. 35f and 36f), the stop condition for the angular velocity is reached before the linear velocity condition, so the linear motion that the robot still performs generates noise when reading its orientation.

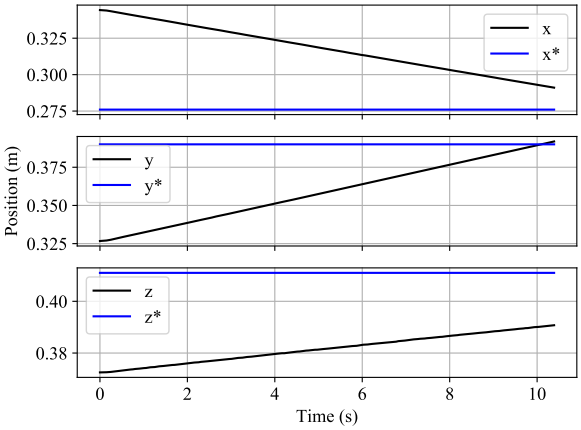
It must be said that these results may be different according to the chosen λ, which interferes both in the convergence time and in the final positioning of the robot. However, as the same experimental conditions were kept for all models, it is possible to make a comparative analysis of efficiency. The largest positioning error for Model 1 is in z, which differs by 21mm from the expected. The biggest error in Model 2 is also in z, of only 9mm. Model 3 has an error of 16mm in y. Model 4, although it has the largest MSE among the models, is the one that achieves the least positioning error, of 8mm in z. This may indicate that, although in general the values predicted by Model 4 are more distant from the expected values, they are proportional, which plays a fundamental role in visual servoing. Finally, Model 5 produces a 15mm z error.

Concerning orientation, the convergence of control occurs when the line changes its slope. Motion from that point on is just noise, which would be suppressed if there was feedback. The angular displacement that the robot must make in this validation instance

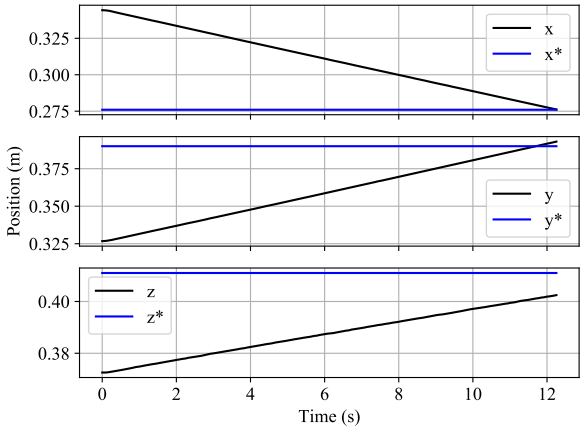
Figure 35: Evolution of the robot's position in time (black) and its desired value (blue), when the network prediction (weighted by $\lambda_{lin} = 0.1$ and $\lambda_{ang} = 10$) for some instance of the validation set is sent to the robot, which executes it without feedback.



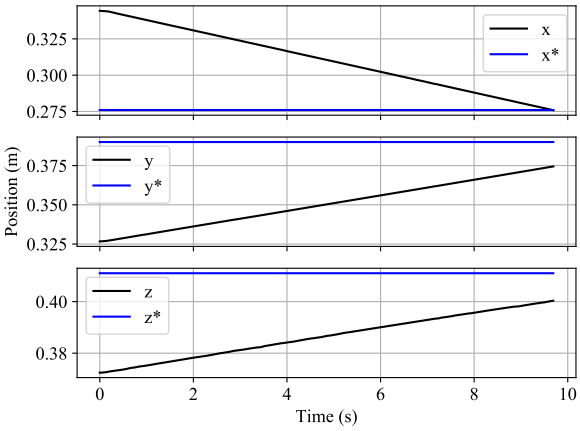
(a) Labeled linear velocity



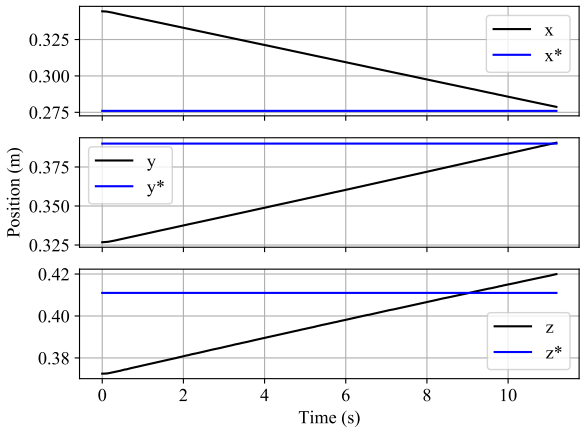
(b) Model 1 prediction



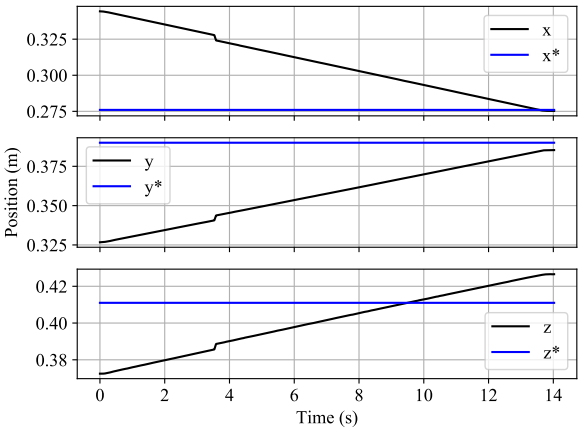
(c) Model 2 prediction



(d) Model 3 prediction



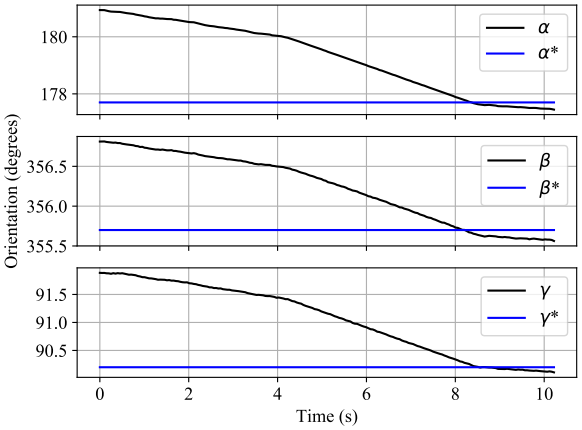
(e) Model 4 prediction



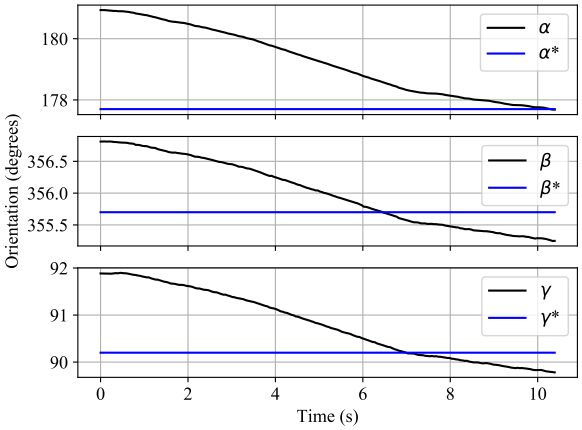
(f) Model 5 prediction

Source: Author

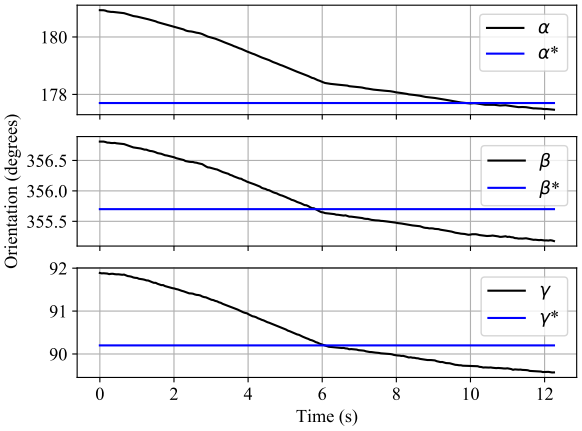
Figure 36: Evolution of the robot's orientation in time (black) and its desired value (blue), when the network prediction (weighted by $\lambda_{lin} = 0.1$ and $\lambda_{ang} = 10$) for some instance of the validation set is sent to the robot, which executes it without feedback.



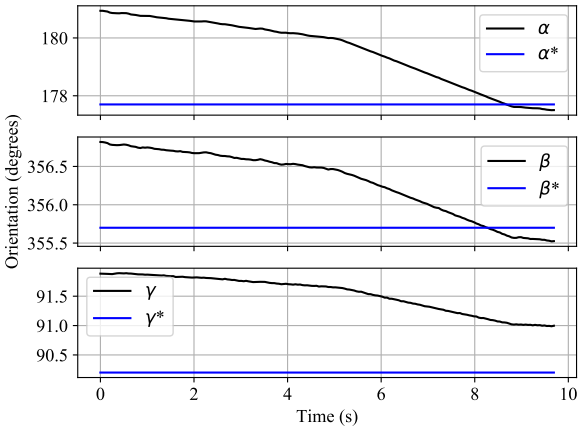
(a) Labeled angular velocity



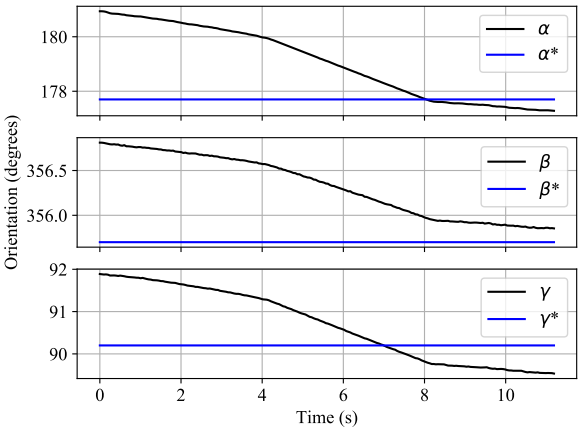
(b) Model 1 prediction



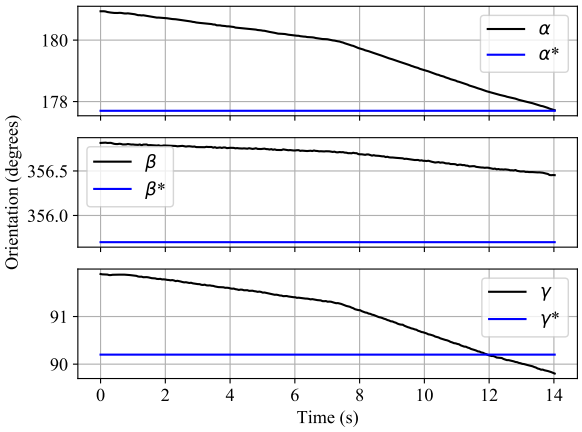
(c) Model 2 prediction



(d) Model 3 prediction



(e) Model 4 prediction



(f) Model 5 prediction

Source: Author

is not so significant, but it still shows that the networks, especially Model 2, can achieve accuracy of less than 1 degree in all directions.

Figs. 35 and 36 clearly show how well adapted the networks are for the visual servoing task, as they show the difference between the final pose achieved by the robot and the desired one. However, the figures do not accurately express the robot's behavior when the networks are used as actual controllers, in a closed-loop way, with visual feedback. Furthermore, although they demonstrate generalization for tuples of images not seen in training, they do not demonstrate generalization for unseen objects. These two scenarios are explored in subsection 5.2.2.

5.2 Online results

This section presents the achieved results when the developed algorithms are implemented in the Kinova Gen3. So, suitability for the real world and time constraints can be assessed.

5.2.1 Grasp Detection

The grasp detection results are presented visually, showing the predicted grasp rectangle in the image acquired by the robot for different objects. Once the rectangle is correctly predicted, the grasp can be executed considering the $2D \rightarrow 3D$ mapping derived in Chapter 4. The network application scenario is quite different from that represented in the Cornell dataset, primarily due to the different objects, but also due to the considerable difference in luminosity. Fig. 37 shows eighteen detection examples.

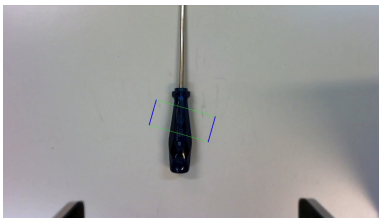
In general, the figure makes it evident that the predicted rectangles (grippers indicated in blue) are consistent with the shape of the target objects and can adapt to their sizes and orientations. Some particular cases, however, deserve more attention.

The objects represented in Figs. 37a and 37e are the easiest for the network to detect grasps, as they are large in length and small in thickness. Most of the objects that are commonly handled follow this pattern and therefore have a large number of instances in the Cornell dataset.

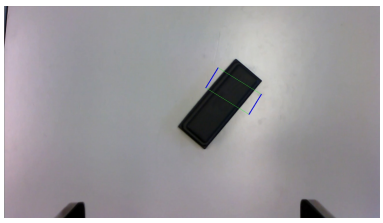
The chalkboard eraser (Fig. 37b) and pencil case (Fig. 37f) are not as simple as the previous objects because they are thicker. The network needs to reason whether the object is so large that it exceeds the represented gripper opening. This same situation occurs with circular objects, as illustrated by the detections in the hand massage ball (Fig. 37g) and anti stress ball (Fig. 37h), in which the smaller ball is easier to detect. However, circular objects can also confuse the network due to the infinite possibility of orientations.

The masking tapes represented in Figs. 37c and 37i also exemplify the thickness problem. But the great challenge regarding these objects is with the ring format. Often

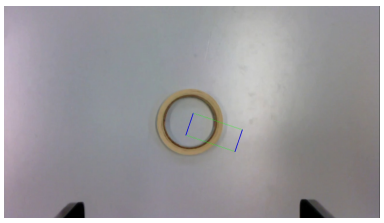
Figure 37: Network prediction for different objects first seen by the robot (Source: Author)



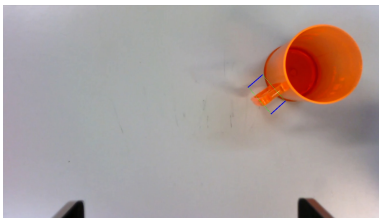
(a) Screwdriver



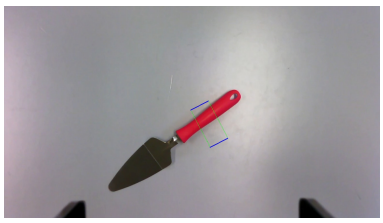
(b) Chalkboard eraser



(c) Thin masking tape

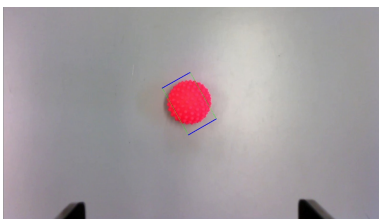


(d) Cup



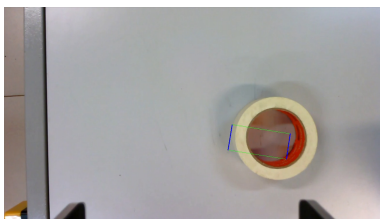
(e) Pie knife

(f) Pencil case

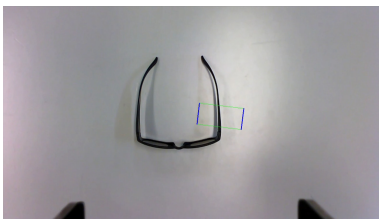


(g) Hand massage ball

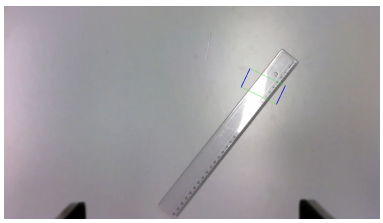
(h) Anti stress ball



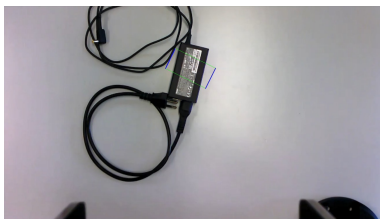
(i) Thick masking tape



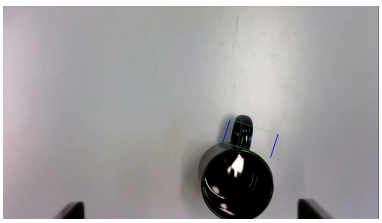
(j) 3D glasses



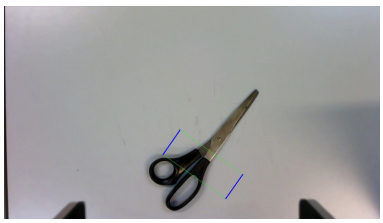
(k) Ruler



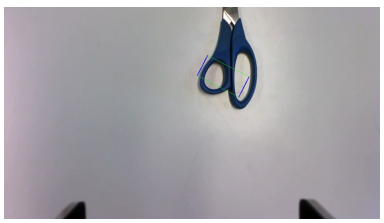
(l) Laptop charger



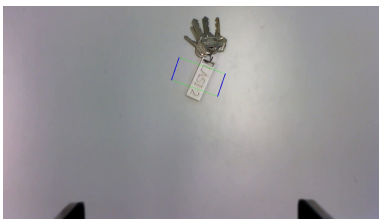
(m) Mug



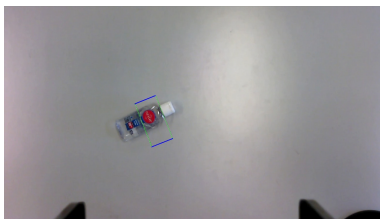
(n) Scissors 1



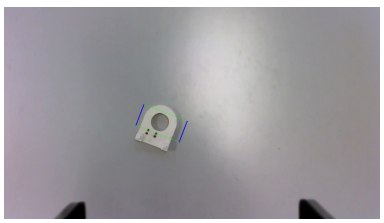
(o) Scissors 2



(p) Keychain + Bunch of keys



(q) Hand sanitizer



(r) Random polymer device

the network may reason that the middle of the object is a constituent part of it, especially if the object and the background have similar colors. Besides, the prediction of a grasp rectangle in the middle of a circular object (because of the arrangement of its labels), is just surpassed because only one label is used during training.

Cups and mugs can also offer a challenge to the network, as they can be caught both by the edges or by the handles. In both Figs. 37d and 37m, the network predicts a rectangle in the handles, which, especially in the case of Fig. 37m, can be justified by the prominence of the handle and the fact that the edges are not so distinguishable.

The ruler (Fig. 37k) and hand sanitizer (Fig. 37q) are transparent and, especially in the case of the ruler, reflective, however, the network manages to distinguish them. The scissors in Figs. 37n and 37o and the device in Fig. 37r, test the network's ability to reason on scale. The first scissor is further away from the camera, so the network prefers to predict a large rectangle, whereas the second scissor is closer, and the network can have a better understanding of how big the hole between the handles is, so it can make a more skillfully grasp. The device in 37r exemplifies the same situation, however, whether near or far, the network always prefers the larger opening, since it understands that the gripper does not fit in the hole.

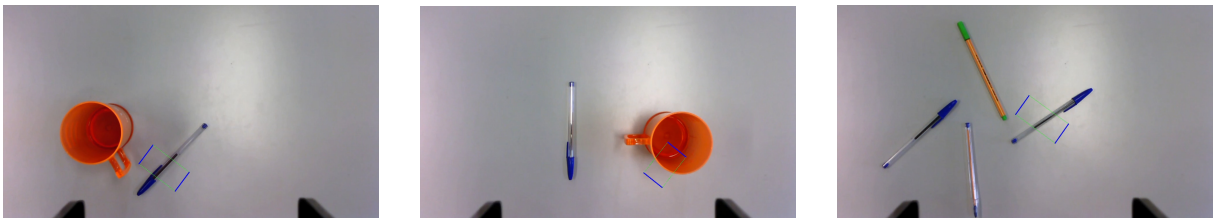
To end the discussion about Fig. 37, the objects in Figs. 37j, 37l and 37p test the network's ability to find the best grasp location, amid a large number of possibilities. It is possible to note that the network makes wise choices, especially regarding the laptop charger, when choosing the part of the object that is most likely to result in a stable grasp.

A final analysis of the network involves its behavior when the input image has multiple objects, as presented in Fig. 38. The network is designed to deal with only one object, which can be seen as a limitation, or as a strategy to avoid post-processing of choice between the predicted rectangles. The important thing is to note that the presence of two or more objects does not confuse the network so that it makes the prediction somewhere between them, but, on the contrary, it chooses one of the objects as a target. The images 38a and 38b show that this choice is made at random. This attribute is important because, although the application space involves only one object, the network operates normally if another object enters the camera's field of view.

5.2.2 Visual Servoing

To test the networks in a closed-loop visual servoing scenario, as well as assess their robustness to lighting changes (global or local), their ability to generalize to first-seen objects and to ascertain whether the prediction speed ensures real-time operation, these trained networks were implemented to control the Kinova robot. The experiment setup for all models begins at pose [0.276, 0.390, 0.411, 177.734, -4.326, 90.252], where the reference image is taken, and then the robot is repositioned at pose [0.344, 0.326, 0.372 , -179,089,

Figure 38: Grasp network prediction when confronted with multiple objects



(a) Cup and pen A (b) Cup and pen B (c) Pens

Source: Author

-3,201, 91,907], where the control process starts. At each iteration, a new current image I_c is considered by the network. The target object is a scientific calculator and the network models considered are those that achieve the smallest validation errors in the offline stage.

The results of the robot's positioning and orientation, as well as the values of the control signal over time are shown in Figs. 39 to 43 and Table 13, considering $\lambda_{lin} = 0.05$ and $\lambda_{ang} = 2$. The gains chosen are less than those used in the offline experiment, as the network must deal with a noisy scenario. In the final dynamic grasp experiment, the convergence time becomes a relevant factor and is better considered by increasing λ.

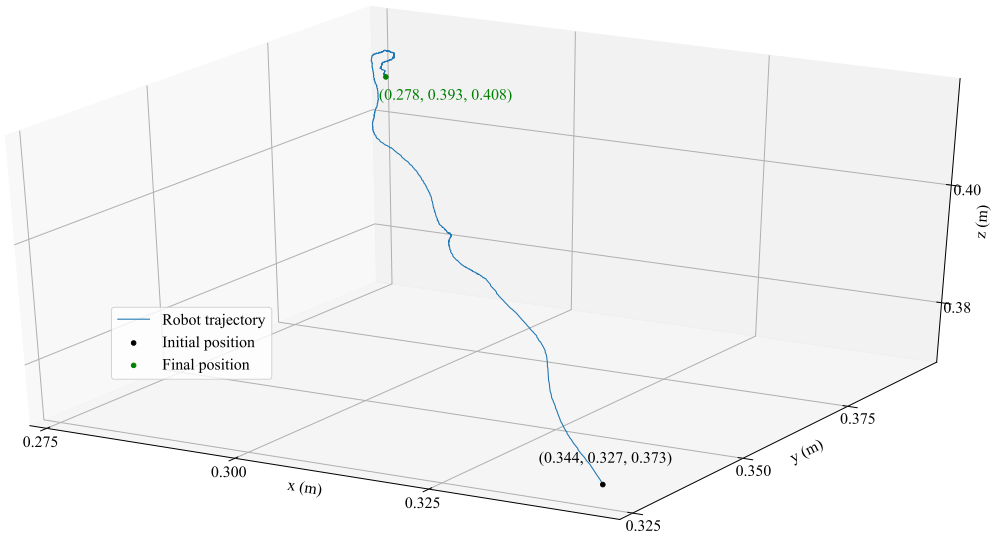
Table 13: Difference between achieved and desired poses using visual servoing with the developed CNN models

Error	x	y	z	α	β	γ
Model 1	1.3 mm	2.3 mm	4 mm	0.56 deg	0.29 deg	0.57 deg
Model 2	26.2 mm	16.2 mm	3 mm	1.20 deg	0.14 deg	0.58 deg
Model 3	1.1 mm	8.4 mm	31.8 mm	0.75 deg	0.15 deg	1.86 deg
Model 4	44.5 mm	3.6 mm	31.1 mm	0.11 deg	0.61 deg	0.95 deg
Model 5	0.3 mm	15.5 mm	69.9 mm	0.82 deg	0.49 deg	0.46 deg

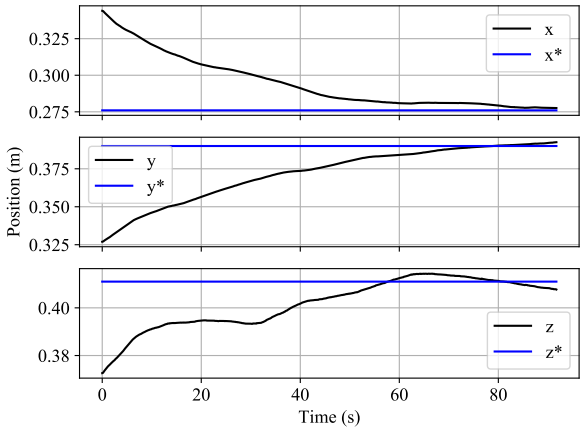
In general, it is possible to note that all networks have more difficulty in inferring the necessary velocity in z. This can be seen both in the final positioning error and in the behavior of the control signal $\dot{z}$, more oscillatory than the others.

The efficiency of the networks must be analyzed assuming that the target object was not an instance of training (it was considered only among other objects, not as a single object). Thus, the final errors of positioning and orientation express the network's ability to generalize a task that, in the literature, is almost always performed starting from a strong modeling and a priori knowledge of the object and camera parameters. Then, a VS controller designed using prior knowledge will obviously outperform one learned from scratch (LAMPE; RIEDMILLER, 2013). The intention is to show that this VS CNN can be applied in the field of autonomous manipulation in situations in which knowledge about the process is scarce or difficult to generate.

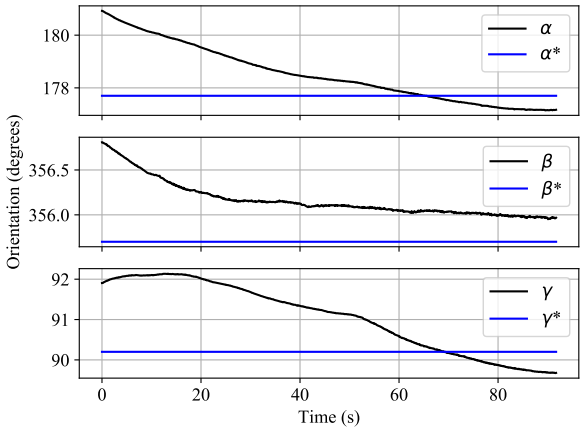
Figure 39: Online results when the robot is controlled by the Model 1 CNN



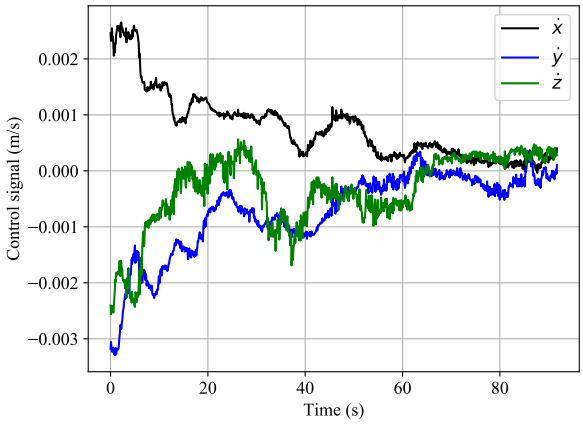
(a) Trajectory performed by the robot during visual servoing



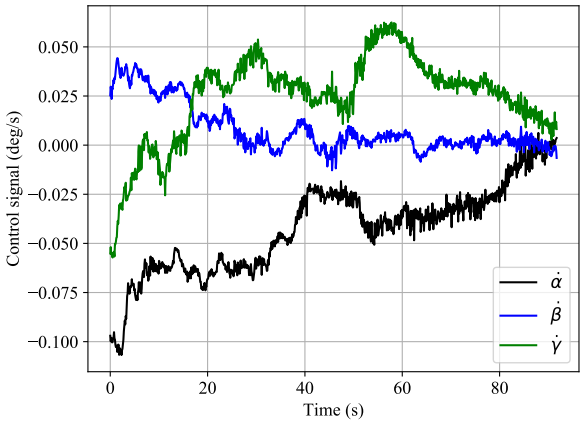
(b) Position in time (setpoint in blue)



(c) Orientation in time (setpoint in blue)



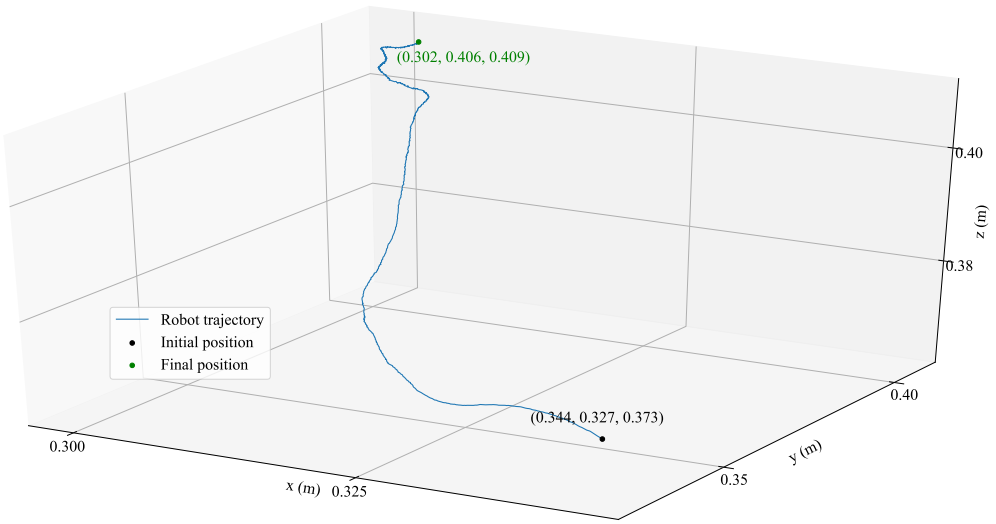
(d) Linear velocity in time



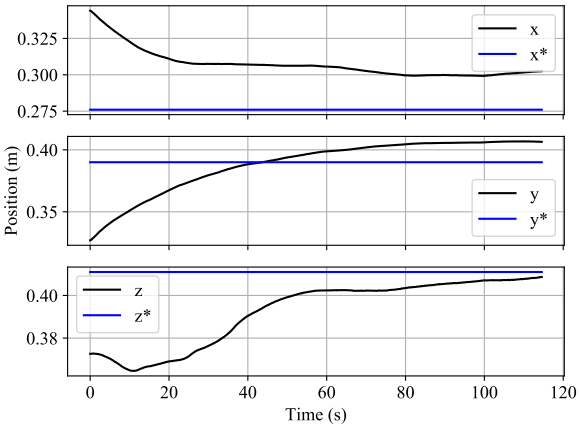
(e) Angular velocity in time

Source: Author

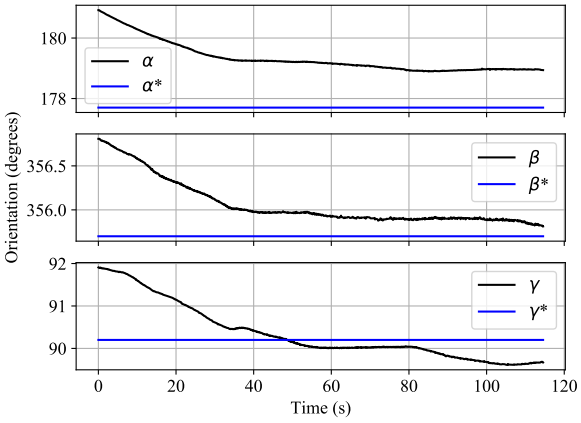
Figure 40: Online results when the robot is controlled by the Model 2 CNN



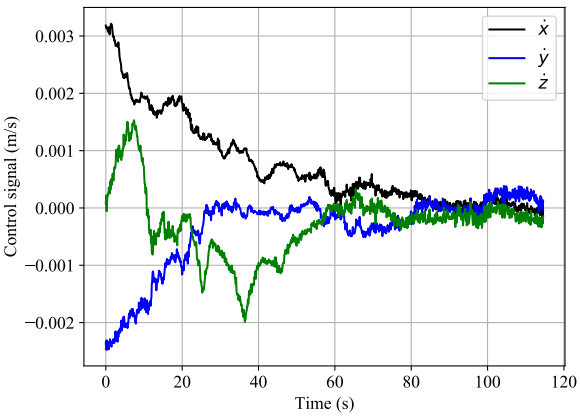
(a) Trajectory performed by the robot during visual servoing



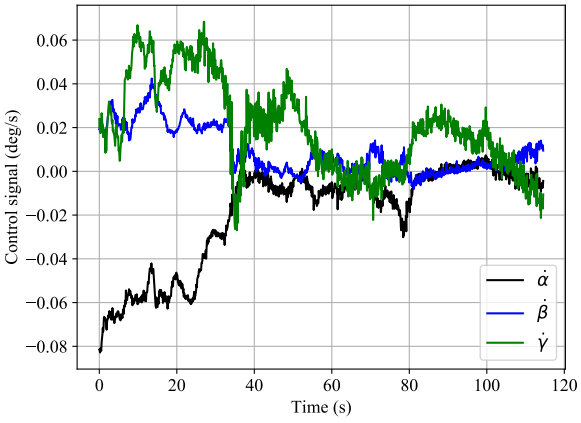
(b) Position in time (setpoint in blue)



(c) Orientation in time (setpoint in blue)



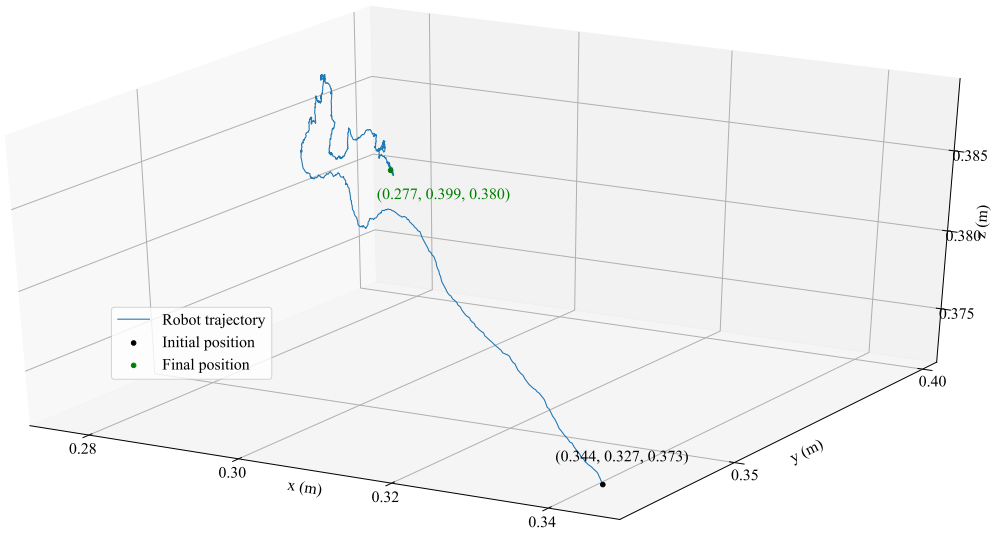
(d) Linear velocity in time



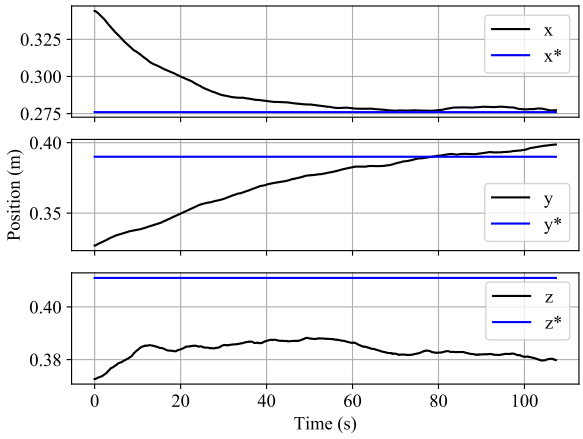
(e) Angular velocity in time

Source: Author

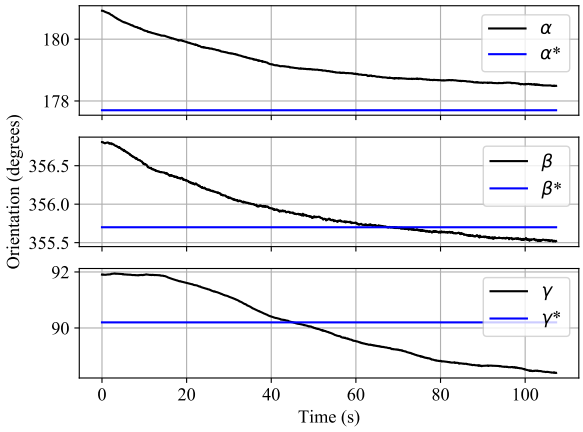
Figure 41: Online results when the robot is controlled by the Model 3 CNN



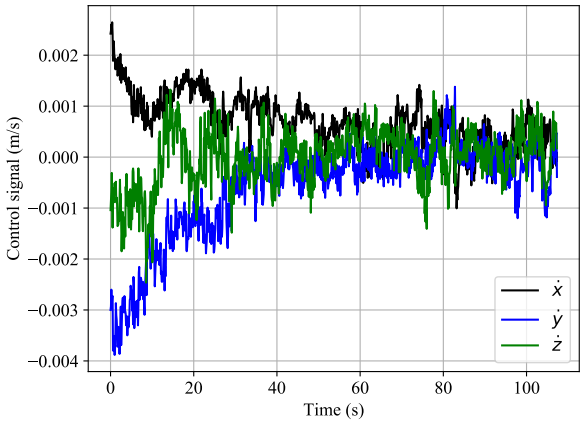
(a) Trajectory performed by the robot during visual servoing



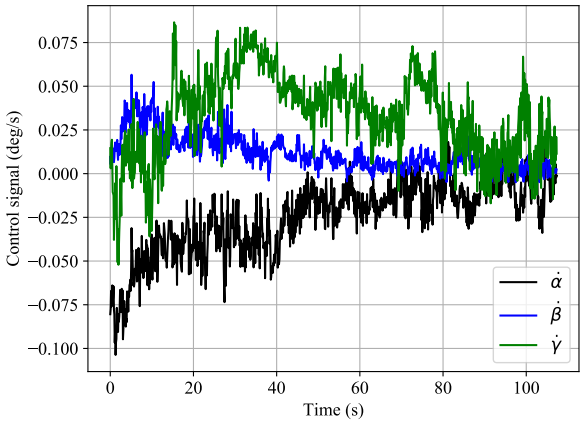
(b) Position in time (setpoint in blue)



(c) Orientation in time (setpoint in blue)



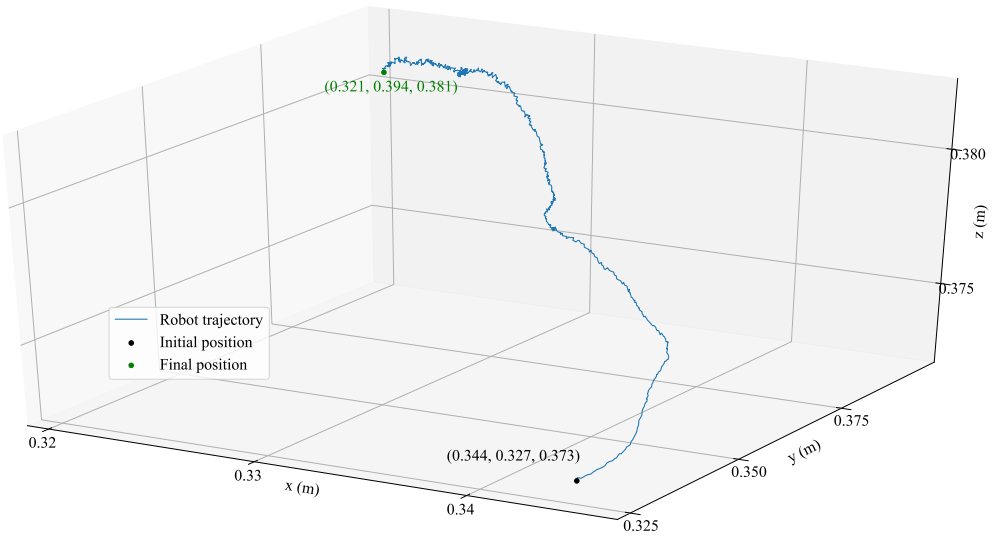
(d) Linear velocity in time



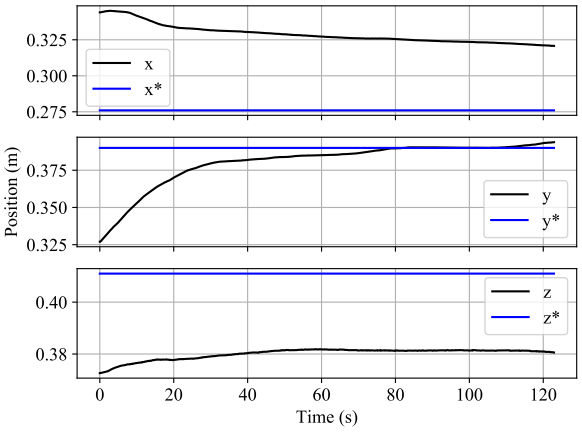
(e) Angular velocity in time

Source: Author

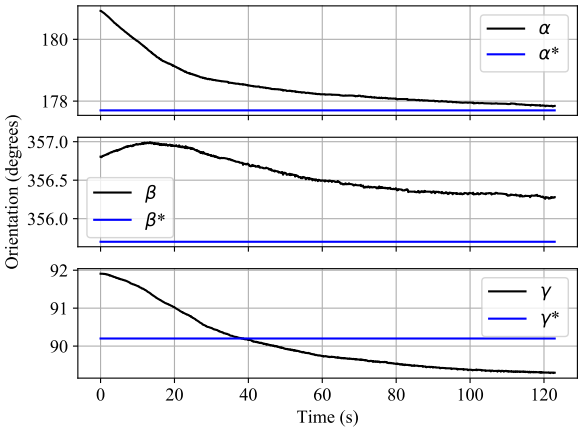
Figure 42: Online results when the robot is controlled by the Model 4 CNN



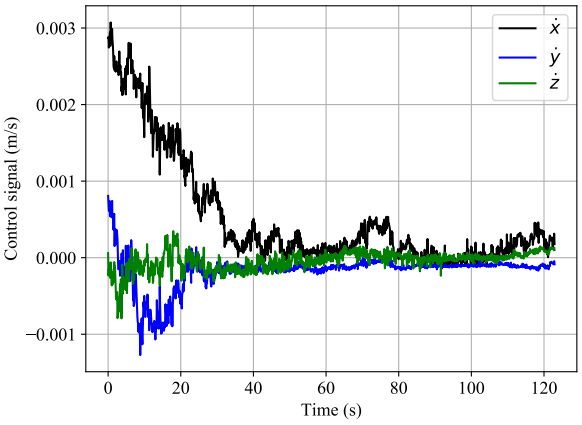
(a) Trajectory performed by the robot during visual servoing



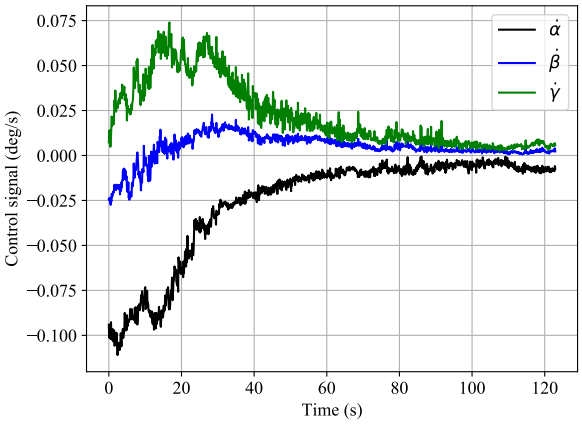
(b) Position in time (setpoint in blue)



(c) Orientation in time (setpoint in blue)



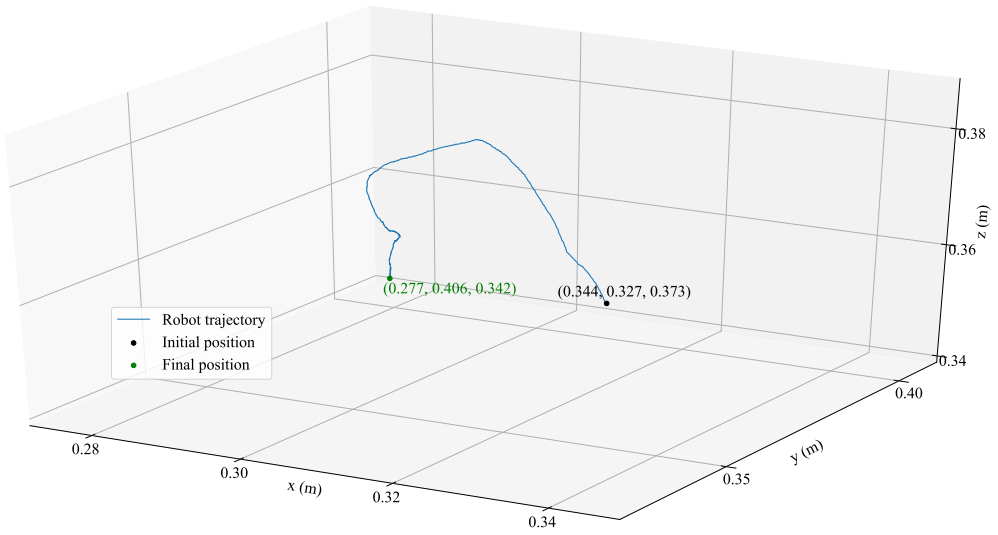
(d) Linear velocity in time



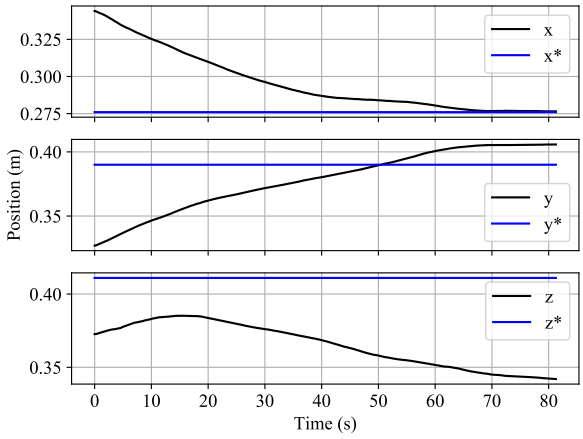
(e) Angular velocity in time

Source: Author

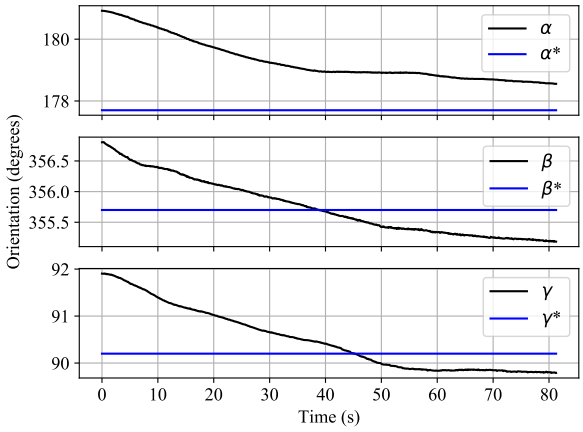
Figure 43: Online results when the robot is controlled by the Model 5 CNN



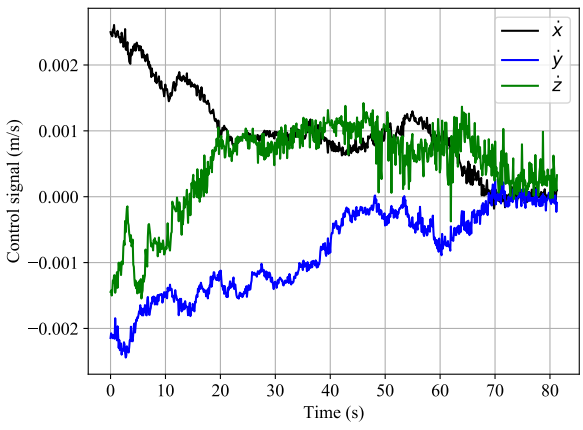
(a) Trajectory performed by the robot during visual servoing



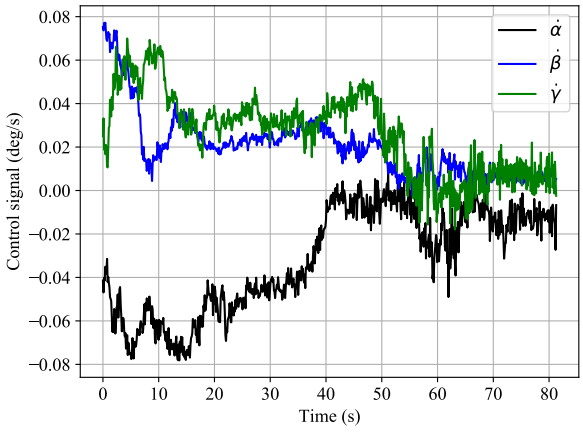
(b) Position in time (setpoint in blue)



(c) Orientation in time (setpoint in blue)



(d) Linear velocity in time



(e) Angular velocity in time

Source: Author

However, designed networks are still able to obtain final millimeter positioning errors and less than one degree orientation errors. Especially Model 1, which is simpler and with higher prediction speed, surprisingly manages to obtain a final positioning error of less than 5mm. All other networks can achieve this same performance, but only in one direction, while others reach the order of centimeters.

Regarding the trajectory made by the robot during the control, again, Model 1 is the one that has the most desired behavior. This is the closest to a pure straight line in 3D space, something obtained in classic PBVS, using $\mathbf{s} = ({}^{c^*}\mathbf{t}_c, \theta\mathbf{u})$, only when the parameters involved in the control law are perfectly estimated.

Model 2, which obtained the lowest MSE in the offline phase, ends the control with a relatively high error in positioning, however, it is the one that converges better in z. Regarding the orientation, it can be noted that this is the model that leads to the faster stabilization of the desired angles, something that can be attributed to the dissociation of the network outputs between linear and angular control signals.

Model 3 converges very well for x and y, but has a large final error in z, whereas Model 4, considered a better version of Model 3, converges well only in y. However, the behavior of the control signal generated by Model 4 is the closest to a default controller, in which speeds tend to zero quickly and no longer oscillate after that. Model 3, on the other hand, generates an extremely noisy control signal, which penalizes the trajectory made by the robot.

Finally, Model 5 achieves good positioning and good orientation, except for the z axis. The control signal $\dot{z}$ followed a good behavior for up to 20 seconds, at which point it started to diverge and generated a final positioning error of almost 70mm. Thus, considering the results obtained offline and in closed-loop, it can be considered that this is the model less suited to the task of visual servoing. This can be justified by the atrous convolutions layer, which proved to be inefficient for the task since the features that must be obtained for the control must be denser, not more sparse.

5.2.2.1 Stability considerations and comparison of results

As found in Chaumette and Hutchinson (2006), it is possible to use Lyapunov analysis to assess the stability of closed-loop visual servo systems. Considering the following candidate Lyapunov function,

$$\mathcal{L} = \frac{1}{2}\|\mathbf{e}(t)\|^2, \tag{5.1}$$

whose derivative is given by

$$\dot{\mathcal{L}} = \mathbf{e}^T\dot{\mathbf{e}} = -\lambda\mathbf{e}^T\mathbf{L}_e\hat{\mathbf{L}_e^+}\mathbf{e}, \tag{5.2}$$

the global asymptotic stability is achieved if the following sufficient condition is met:

$$\mathbf{L}_e\hat{\mathbf{L}_e^+} > 0. \tag{5.3}$$

Since $\mathbf{L}_{\theta u}$, given by Eq. 2.34, is nonsingular when $\theta \neq 2k\pi$ this condition is met, since $\mathbf{L}_e\hat{\mathbf{L}_e^{-1}} = \mathbf{I}_6$, under the strong hypothesis that all pose parameters are perfectly estimated (CHAUMETTE; HUTCHINSON, 2006). This is true for the interaction matrix in Eq. 2.33 of the considered PBVS, as it is full rank when $\mathbf{L}_{\theta u}$ is nonsingular.

However, in a classic VS system, the pose is estimated as a function of measurements in the image and camera calibration, so the interaction matrix can be biased due to errors, or inaccurate due to noise. So, as stated by Chaumette and Hutchinson (2006), even small errors in computing pose can strongly impact the efficiency and stability of the system.

This restriction motivated the network design in an end-to-end way, in which the network prediction is the control signal itself, eliminating the pose estimation stage. The impact of this consideration may be minimal, but considering that the problem of pose estimation from CNNs still needs major improvements (SHAVIT; FERENS, 2019; SÜNDERHAUF et al., 2018), it was decided to use the network directly as a controller, differently from the approach of Bateux et al. (2018) and Saxena et al. (2017), that used it as a pose estimator.

Regarding the work of Bateux et al. (2018), some considerations and comparisons can be made. The authors obtained good positioning results, in the order of sub-millimeters, in two specific scenes. For each scene, thousands of training instances were generated (using smaller standard deviations than those considered here) so that the control could be done in these two specific scenes, that is, the training and application scenarios are the same. Even so, during the test, the robot does not make a satisfactory trajectory and must make large displacements to reach the desired image. It can be said that this behavior comes from the propagation of the error in estimating the pose, during the control signal calculation.

To perform visual servoing on objects never seen before, the authors propose a "scene-agnostic CNN" based on the fine-tuning of a VGG from 100k couple of viewpoints from a publicly available dataset. During the test, the network can position the robot a few centimeters from the desired position, and then the control is switched to a classic DVS to obtain sub-millimetric accuracy.

The results obtained by Bateux et al. (2018) endorse the results achieved by the proposed networks. In particular, Model 1, which is approximately 90 times smaller than VGG, achieves remarkable positioning accuracy on a first-seen object without having to switch to a classic controller. Besides, the trajectory made by the robot approaches a straight line, something desirable in VS. So, the efficiency of CNNs, since trained with a

representative dataset, is proved for the task of visual servoing in unknown environments.

5.3 Dynamic Grasping

The last experiment carried out on the robot is the test of the entire system, as illustrated in Fig. 17. To do this, the robot takes a reference image of an anti stress ball and then performs a displacement $[\Delta x, \Delta y, \Delta z, \Delta\alpha, \Delta\beta, \Delta\gamma] = [0.15m, -0.15m, 0.05m, 5°, -5°, 5°]$, from where control starts. To mimic the behavior of an object that changes its desired position after the grasping attempt is started, it was preferred to change the pose of the robot, which has the same effect and allows for better monitoring and reproducibility.

The specific ball was not used in the grasping dataset, either in the control dataset. Therefore, the experiment evaluates the adaptation of the robot to a dynamic first-seen object using a method based entirely on learning. Only Model 1 is considered since this is the one that achieves the best results in the robot.

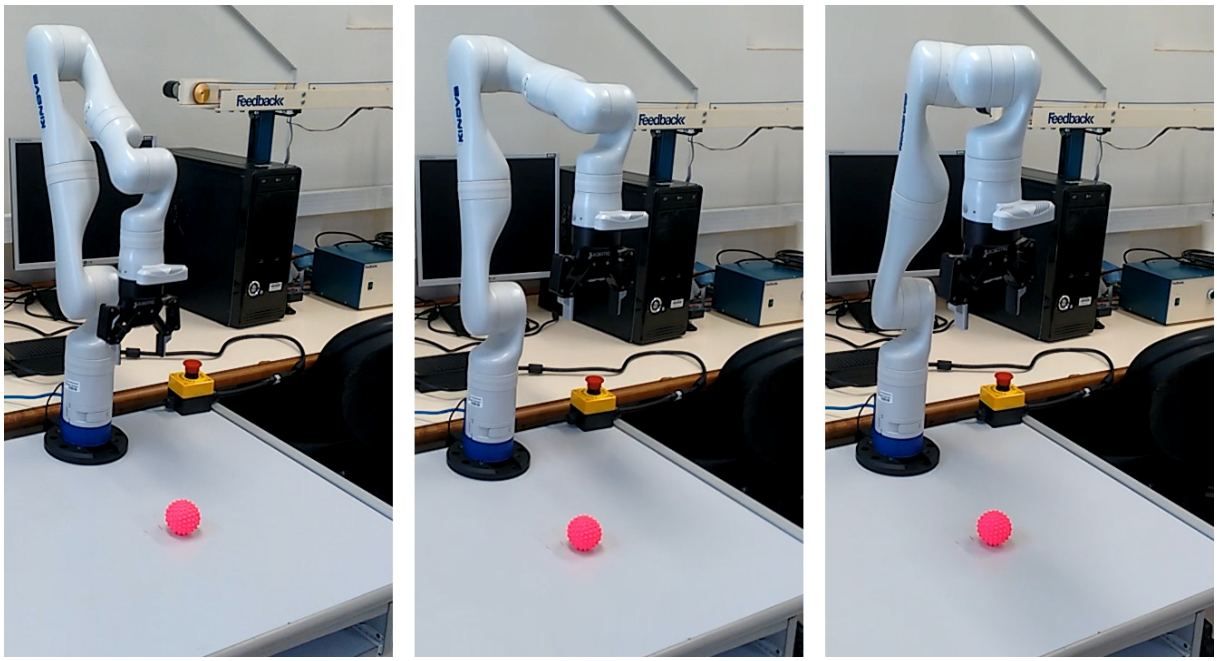
Fig. 44 illustrates the environment in which the robot is assembled and shows each step of the process. In Fig. 44a the robot setup is shown, highlighting the initial pose, followed by the pose in which the robot captures the reference image and the final pose, from where the dynamic grasping starts.

In Fig. 44b, three images of the robot are shown at different moments of the visual servoing, with the last image illustrating the robot's pose in the instant before the grasp execution when the stop condition is reached. Below each image of the robot, the differences between the current images of each instant and the desired image are also illustrated. The desired position of the ball is represented in blue and the current position in green. During the entire process, it is possible to see the prediction of the grasp network, so that the user can intervene any time he thinks that the grasping can be performed, according to the prediction seen. At the beginning of the movement, in which the robot has greater speed, prediction has an offset from the correct rectangle, but the two subsequent images show that the obtained rectangles allows for grasping.

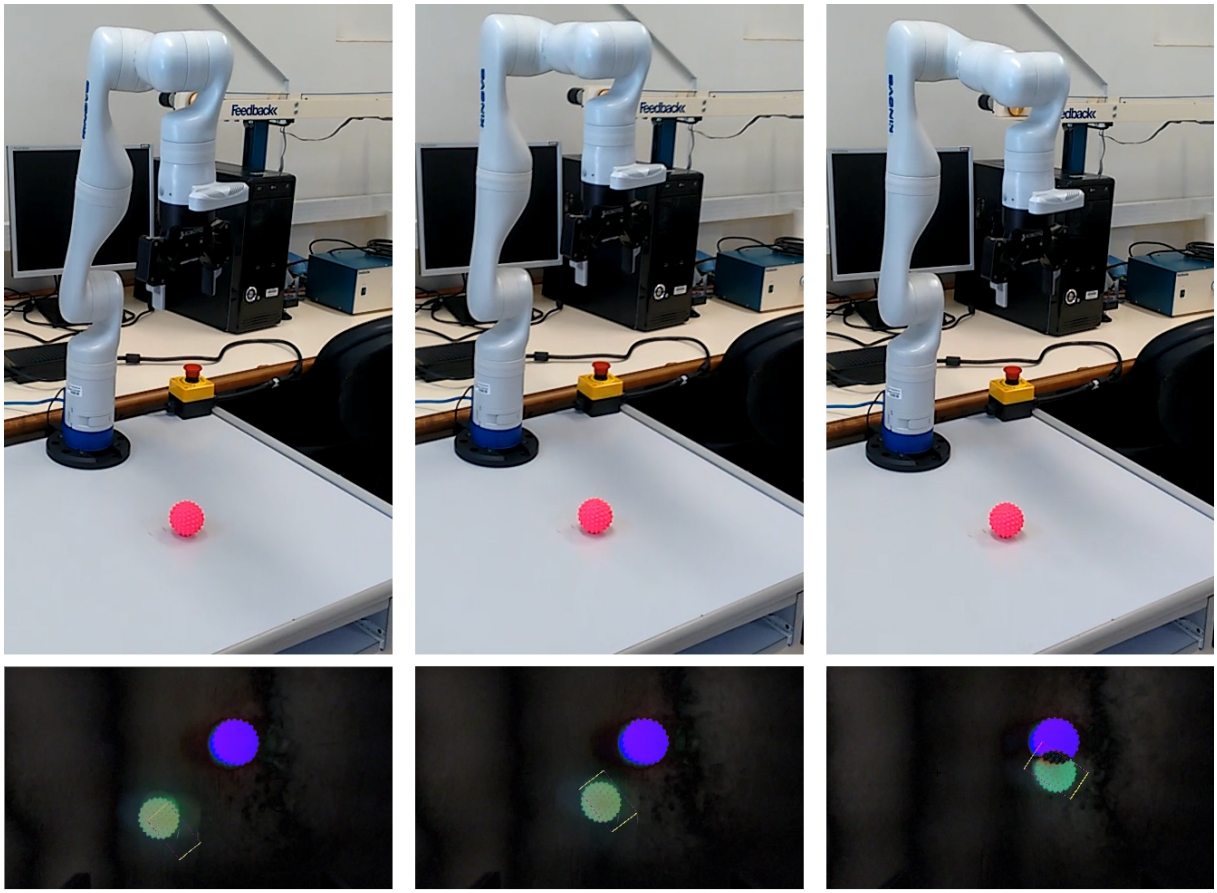
When the stop condition is reached, in which the current position of the ball is close enough to the desired position, the robot starts to approach the object for grasping. Fig. 45 shows some moments of the execution. In the first and second images, the robot adjusts the orientation and opening of the grippers and also moves towards the (x, y) position after mapping the network prediction. Depth information is not used, so the final z value is fixed. In the third image, the robot closes its grippers until the applied force reaches a threshold, and then takes the grasped ball to a pre-defined position.

The behavior of the robot and the applied control signal are illustrated graphically in Fig. 46. The robot comes close to convergence only in y, but as seen by the difference between the current and desired images in Fig. 44, it was enough to considerably reduce

Figure 44: Demonstration of the VS stage in the dynamic grasping



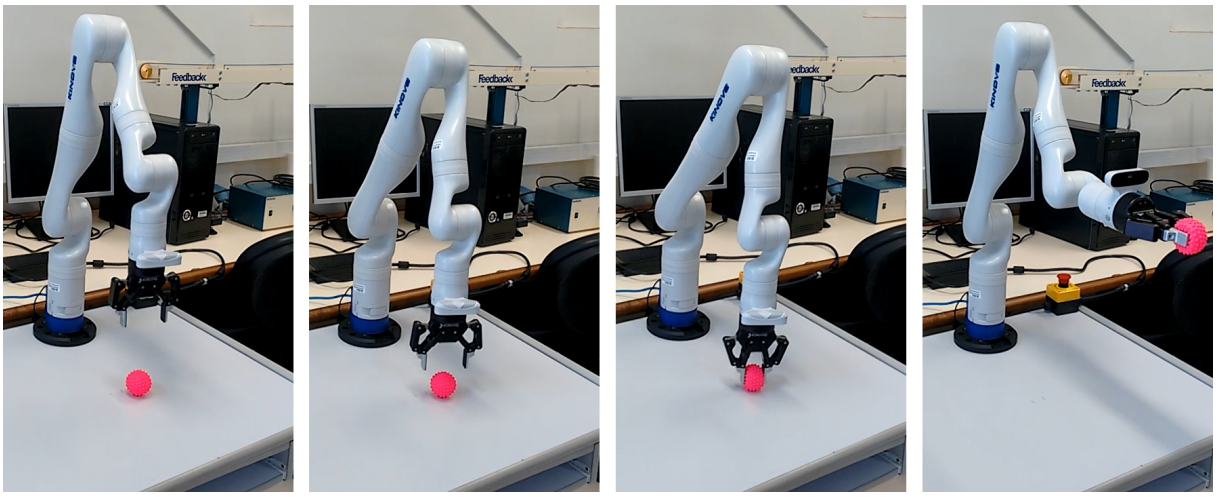
(a) Initial setup for VS



(b) VS execution both in Cartesian and image spaces

Source: Author

Figure 45: Demonstration of the grasp execution right after VS is done



Source: Author

the distance to the desired position. The behavior in z starts in the direction of divergence, but after iteration 60, it starts to converge. The angular velocity signal was still relatively large, in module, when the VS was interrupted.

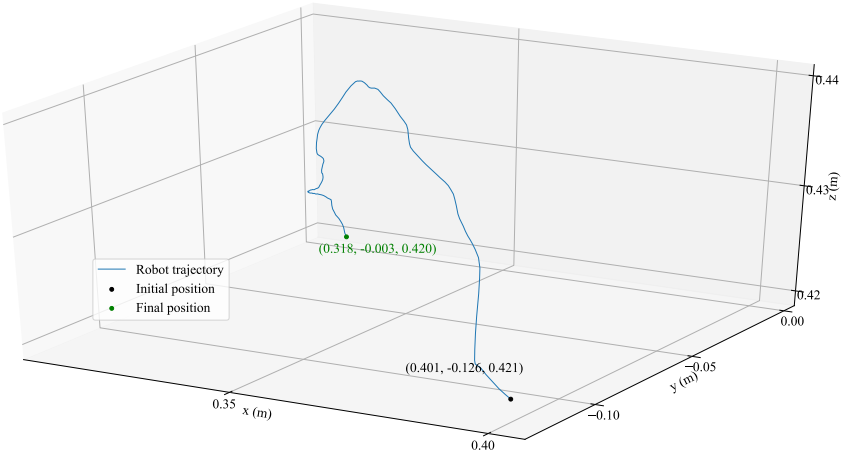
Therefore, the choice of the considered L1-norm threshold must be the trade-off between the positioning accuracy and the speed with which grasping is performed. In a dynamic grasping scenario, obviously the speed plays a more important role, so the system should perform the grasping as soon as possible, whereas the visual servoing step serves much more the task of keeping the object in the camera's field of view, than positioning accuracy.

For this reason, a relatively high threshold was chosen, so that the execution of the control does not take too long. It took 14s for the robot to stop visual servoing and perform grasping. This time can be reduced by considering higher gains and/or thresholds.

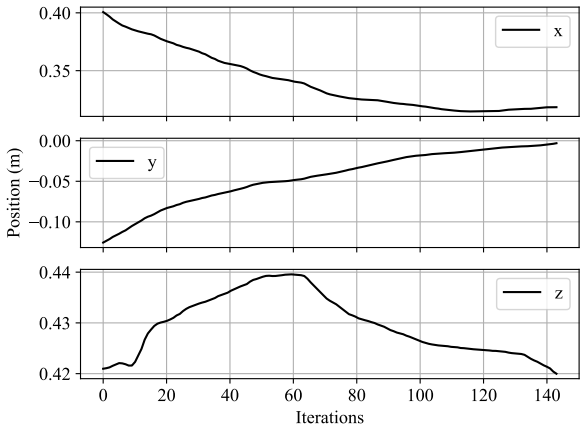
A second experiment to analyze the robot's behavior concerning a dynamic object is performed, in which a quick and unexpected change in the object's position is considered, as well as the case where the object moves with irregular speed. During the experiment, the position of the object is not annotated, so the description of the motion performed by the object is purely approximate and visual, illustrated by Fig. 47.

The object considered for tracking is a screwdriver (not seen in training), whose desired position is represented in blue in the image. In the first sequence, the same conditions as in the previous experiment are maintained, that is, the robot takes an image in a certain pose, which serves as the desired image, and then moves to another pose to start the control. Sequence 2 occurs right after the object is abruptly moved to the right, causing the robot to track the object and then adjust the position.

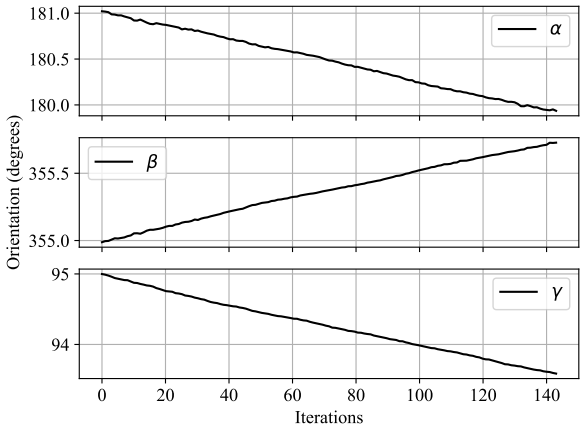
Figure 46: Results for VS in the dynamic grasping scenario: the robot is positioned $[\Delta x, \Delta y, \Delta z, \Delta\alpha, \Delta\beta, \Delta\gamma] = [0.15m, -0.15m, 0.05m, 5°, -5°, 5°]$ from the desired pose, but as soon as the control signal's norm is below 0.05, the system perceives that it no longer needs to track the object, so the grasp can be executed.



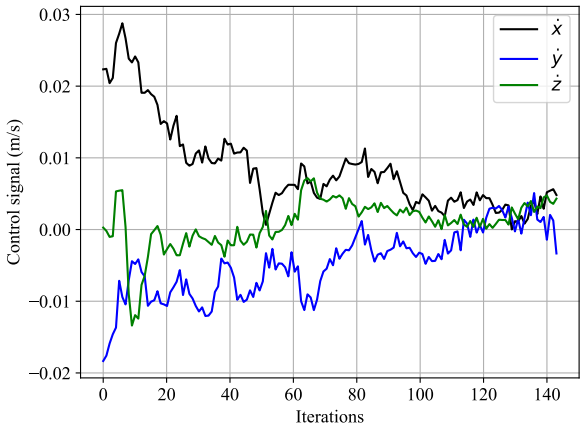
(a) Trajectory performed by the robot during visual servoing for grasping



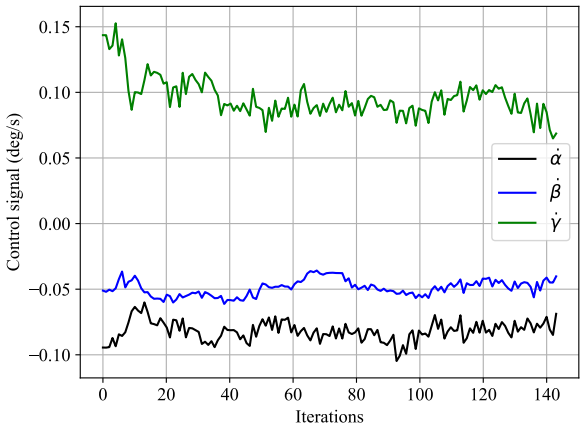
(b) Position per iteration



(c) Orientation per iteration



(d) Linear velocity per iteration

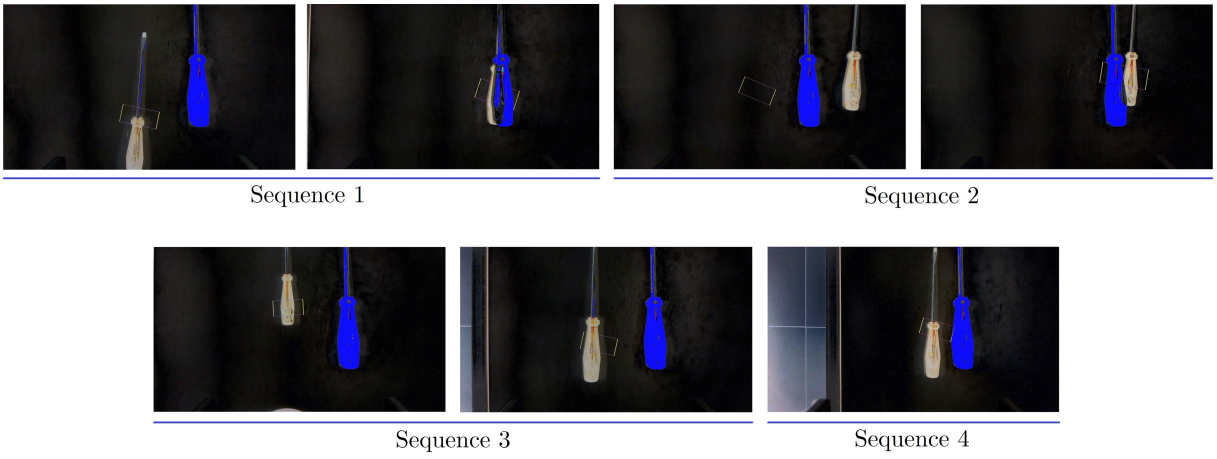


(e) Angular velocity per iteration

Source: Author

During Sequence 3, the screwdriver is moved with approximately constant speed in y (robot base frame), causing the robot to follow the movement with a certain offset between the current and desired images. Finally, in Sequence 4, the robot adjusts its position after the object stops moving, and, as soon as the stop condition is met, the control is ended and the grasping is performed.

Figure 47: Variation of difference between expected and current images during VS



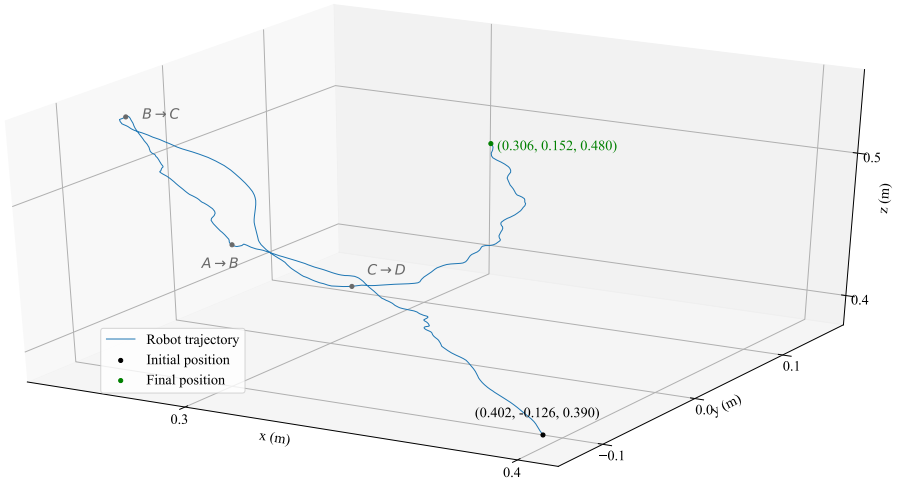
Source: Author

Fig. 48 graphically illustrates all stages of the experiment and shows the 3D trajectory made by the robot, highlighting the transition between sequences. From the linear speed signal it is possible to notice that the robot was about to stop the visual servoing and perform the grasping, but the sudden change in the position of the screwdriver, causes an overshoot in the signal. The position graph shows that the correct y position is quickly reached and only maintained until the beginning of Sequence 3. From this moment on, the y position of the robot progressively increases as the object moves in that same direction.

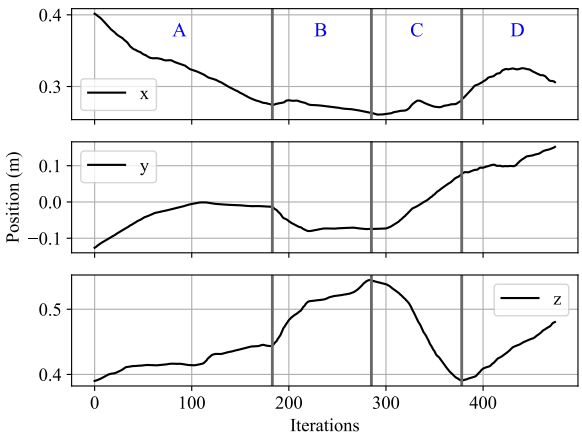
From the beginning of Sequence 4, the y position is only adjusted to match the desired image, until the control ends. It is interesting to note, however, that z varies greatly in response to these changes, even though it is not directly involved in them, while x remains almost constant, as expected. It may indicate that is necessary to adjust a particular gain for $\dot{z}$, or to retrain the network considering larger variations in z. At the end of the control, the speeds become very noisy because the floor appears in the image, confusing the network. However, even so, the stop condition is reached at a certain moment, and the grasping is successfully performed.

A final experiment to evaluate the dynamic grasping system was performed in order to give an idea about the efficiency of the network for different objects and different initial poses. Far from being a statistical survey, but capable of validating the network's applicability, 20 grasping attempts in a dynamic scenario were considered for four objects.

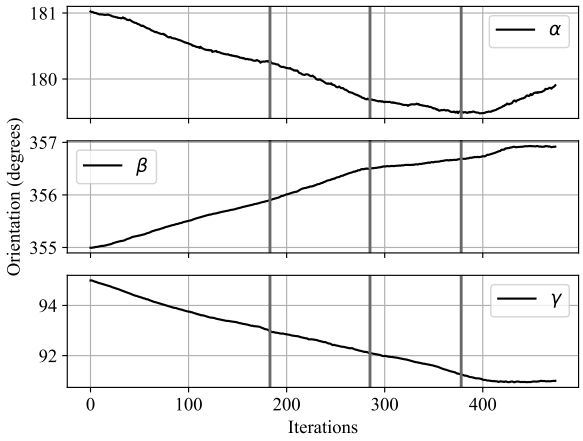
Figure 48: Robot's behaviour in the face of a quick and unexpected change in the object's position (A to B), followed by a displacement with lower, but irregular speed (C) and later adjustment as soon as the object is motionless (D)



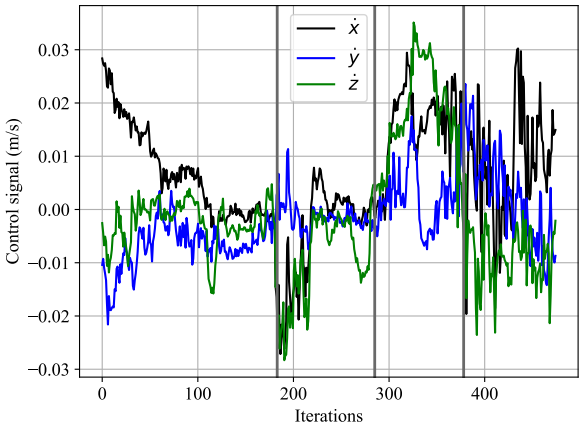
(a) Trajectory performed by the robot during visual servoing for grasping



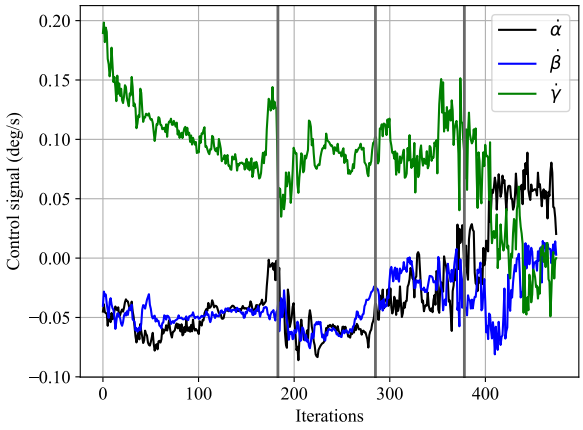
(b) Position per iteration



(c) Orientation per iteration



(d) Linear velocity per iteration



(e) Angular velocity per iteration

Source: Author

The robot take an image at a pre-defined position and then is repositioned considering a Gaussian distribution with standard deviations $[\sigma x, \sigma y, \sigma z, \sigma\alpha, \sigma\beta, \sigma\gamma] = [50\text{mm}, 50\text{mm}, 50\text{mm}, 2°, 2°, 2°]$. The results are shown in Table 14, indicating a success rate of 85%, in which, of the three failure cases, two occurred due to errors in the grasp network prediction, and one due to errors in the VS CNN.

Table 14: Result of the dynamic grasping system for twenty attempts involving four different objects. G and VS associated with unsuccessful attempts indicate whether the error occurred due to failure of the grasp or visual servoing network prediction.

Target object	Attempt 1	Attempt 2	Attempt 3	Attempt 4	Attempt 5
Screwdriver	✓	✓	✓	✓	✓
Scissors	✓	✓	✓	✗ (G)	✓
Pen	✓	✓	✓	✓	✓
Masking tape	✗ (VS)	✓	✓	✗ (G)	✓

Viereck et al. (2017) reported a success rate of 77% (58/75) when considering dynamic objects in a reinforcement learning scenario. Morrison, Corke and Leitner (2019), on the other hand, develop a reactive grasping system quite similar to the one proposed, except for the fact that the visual servoing is a classic PBVS and dependent on grasping prediction, that achieves 81% (76/94) accuracy. Both works considered cluttered scenes, which may difficult grasp detection, but not the dynamics of the set, which, in all attempts, is only moved 10cm in one direction and rotated 25 degrees in z, which does not reproduce a challenging dynamic scenario.

Considering single dynamic objects, only the work of Morrison, Corke and Leitner (2019) shows results. The authors reach 86% efficiency in 200 attempts, but, again, the control is done by a classic PBVS, depending on the pose predicted by the grasping network and the dynamics of the scene is the same as the presented for the cluttered scenario. Thus, although the comparison is not necessarily fair, since the methods consider different objects and a much larger number of attempts, it can be admitted that the developed system has greater applicability to deal with the task for which it was designed, especially considering the fact that none of the target objects were in the training phase.

6 CONCLUSION

This work addressed the problems of grasp detection and visual servoing using deep learning, and also applied them together as an approach to the problem of grasping dynamic objects.

A convolutional neural network was employed to obtain the points where a robot with parallel grippers can grasp an object using only RGB information. In order to use a simple network with a small number of parameters, the data augmentation technique was used in an innovative way, allowing to extract as much visual information as possible from the dataset images. As a result, the trained network is able to predict grasp rectangles at speeds that surpass the state-of-the-art in the Cornell Grasping dataset, without harshly penalizing accuracy. The trained network was also evaluated on a robot, with its own camera to test its generalization to different objects, not seen in the training. The visual results demonstrate that the network is robust to real-world noise and is capable of detecting grasps on objects of different shapes, colors, in different orientations and subjected to considerable lighting changes.

Five models of convolutional neural networks were designed as potential candidates for end-to-end visual servo controllers. The networks do not use any type of additional information other than a reference image and the current image, to regress the control signal. These networks were tested both offline and on a robot to assess applicability in a real-world and real-time scenario. The simplest model was able to achieve a millimeter accuracy in the final position considering a target object seen for the first time. To the best of knowledge, no works have been found in the literature that achieves such precision with a controller learned from scratch.

Finally, the trained networks that obtained the best results in each task, were embedded in a final system for grasping dynamic objects. In different scenarios, all *a priori* unknown to the network, the robot was able to keep the target object in the camera's field of view using the real-time visual servoing CNN. When the robot understands that it is close enough to the desired position, the prediction of the grasping network, which is also obtained in real-time, is mapped to world coordinates and the robot approaches the object to execute grasp.

The system at the current stage has the deficiency of not considering depth information during the execution of the grasping. In future work, the intention is to use information from an infrared sensor in the operational stage, maintaining training only with RGB, as this is seen as an advantage. Other considered adjustments relate to the determination of control gains, adaptation of the grasping network to predict multiple

rectangles, obtaining grasp dataset in a self-supervised manner with the robot, and making the visual servoing dataset publicly available.

The trained algorithms were not designed to surpass those modeled from prior knowledge, but so that they can be applied in situations where the knowledge of the system is limited or impossible to be obtained. To this end, it is concluded that the developed methodology has superior applicability due to its high capacity for generalization, simplicity, speed and, above all, accuracy.

6.1 Publication

The development of this work led to the publication of the following conference paper (RIBEIRO; GRASSI, 2019):

RIBEIRO, E. G.; GRASSI, V. Fast convolutional neural network for real-time robotic grasp detection. In: IEEE. **2019 19th International Conference on Advanced Robotics (ICAR)**. Belo Horizonte, Brazil, 2019. p. 49–54. DOI:10.1109/ICAR46387.2019.8981651

BIBLIOGRAPHY

ABADI, M.; BARHAM, P.; CHEN, J.; CHEN, Z.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; IRVING, G.; ISARD, M. et al. Tensorflow: a system for large-scale machine learning. In: USENIX. **12th USENIX Symposium on Operating Systems Design and Implementation**. Savannah, GA, USA, 2016. v. 16, p. 265–283.

AGUIRRE, L. A.; BRUCIAPAGLIA, A. H.; MIYAGI, P. E.; PIQUEIRA, J. R. C. (Ed.). **Encyclopedia of automatics: control and automation (In Portuguese)**. São Paulo: Blucher, 2007.

AMOR, H. B.; KROEMER, O.; HILLENBRAND, U.; NEUMANN, G.; PETERS, J. Generalization of human grasping for multi-fingered robot hands. In: IEEE. **2012 IEEE/RSJ International Conference on Intelligent Robots and Systems**. Algarve, Portugal, 2012. p. 2043–2050.

ASIF, U.; BENNAMOUN, M.; SOHEL, F. A. Rgb-d object recognition and grasp detection using hierarchical cascaded forests. **IEEE Transactions on Robotics**, v. 33, n. 3, p. 547–564, June 2017. ISSN 1552-3098.

ASIF, U.; TANG, J.; HARRER, S. Ensemblenet: Improving grasp detection using an ensemble of convolutional neural networks. In: BMVC. **29th British Machine Vision Conference**. Newcastle, England, 2018. p. 10.

ASIF, U.; TANG, J.; HARRER, S. Graspnet: An efficient convolutional neural network for real-time grasp detection for low-powered devices. In: IJCAI. **27th International Joint Conference on Artificial Intelligence**. Stockholm, Sweden, 2018. p. 4875–4882.

BACHILLER, M.; CERRADA, C.; CERRADA, J. Designing and building controllers for 3d visual servoing applications under a modular scheme. **Industrial Robotics**, I-Tech, p. 81, 2004.

BATEUX, Q.; MARCHAND, E.; LEITNER, J.; CHAUMETTE, F.; CORKE, P. Training deep neural networks for visual servoing. In: IEEE. **ICRA 2018-IEEE International Conference on Robotics and Automation**. Brisbane, Australia, 2018. p. 1–8.

BEN-DAVID, S.; BLITZER, J.; CRAMMER, K.; PEREIRA, F. Analysis of representations for domain adaptation. In: SCHöLKOPF, B.; PLATT, J. C.; HOFFMAN, T. (Ed.). **Advances in Neural Information Processing Systems 19**. Vancouver, Canada: MIT Press, 2007. p. 137–144.

BERGSTRA, J.; BREULEUX, O.; BASTIEN, F.; LAMBLIN, P.; PASCANU, R.; DESJARDINS, G.; TURIAN, J.; WARDE-FARLEY, D.; BENGIO, Y. Theano: A cpu and gpu math compiler in python. In: SCIPY. **Proc. 9th Python in Science Conf**. Austin, USA, 2010. v. 1.

BICCHI, A.; KUMAR, V. Robotic grasping and contact: A review. In: IEEE. **Intenational Conference on Robotics and Automation**. San Francisco, USA, 2000. v. 348, p. 353.

BIRGLEN, L.; SCHLICHT, T. A statistical review of industrial robotic grippers. **Robotics and Computer-Integrated Manufacturing**, v. 49, p. 88 – 97, 2018. ISSN 0736-5845.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. New York: Springer, 2006.

BOHG, J.; MORALES, A.; ASFOUR, T.; KRAGIC, D. Data-driven grasp synthesis—a survey. **IEEE Transactions on Robotics**, IEEE, v. 30, n. 2, p. 289–309, 2014.

BOTTOU, L. Online algorithms and stochastic approximations. In: SAAD, D. (Ed.). **Online Learning and Neural Networks**. Cambridge, UK: Cambridge University Press, 1998. Revised, oct 2012.

BOUSMALIS, K.; IRPAN, A.; WOHLHART, P.; BAI, Y.; KELCEY, M.; KALAKRISHNAN, M.; DOWNS, L.; IBARZ, J.; PASTOR, P.; KONOLIGE, K. et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In: IEEE. **2018 IEEE International Conference on Robotics and Automation (ICRA)**. Brisbane, Australia, 2018. p. 4243–4250.

BRACHMANN, E.; MICHEL, F.; KRULL, A.; YANG, M. Y.; GUMHOLD, S.; ROTHER, C. Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image. In: IEEE. **29th Conference on Computer Vision and Pattern Recognition (CVPR)**. Las Vegas, USA, 2016. p. 3364–3372.

CALDERA, S.; RASSAU, A.; CHAI, D. Review of deep learning methods in robotic grasp detection. **Multimodal Technologies and Interaction**, Multidisciplinary Digital Publishing Institute, v. 2, n. 3, p. 57, 2018.

CARON, G.; MARCHAND, E.; MOUADDIB, E. M. Photometric visual servoing for omnidirectional cameras. **Autonomous Robots**, Springer, v. 35, n. 2-3, p. 177–193, 2013.

CASTRO, D.; MARQUES, L.; NUNES, U.; ALMEIDA, A. T. de. Tactile force control feedback in a parallel jaw gripper. In: IEEE. **ISIE '97 Proceeding of the IEEE International Symposium on Industrial Electronics**. Guimaraes, Portugal, 1997. p. 884–888 vol.3.

CAVALLARI, T.; GOLODETZ, S.; LORD, N.; VALENTIN, J.; PRISACARIU, V.; STEFANO, L. D.; TORR, P. H. Real-time rgb-d camera pose estimation in novel scenes using a relocalisation cascade. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, 2019.

CHAUMETTE, F. Potential problems of stability and convergence in image-based and position-based visual servoing. In: KRIEGMAN, D.; HAGER, G.; MORSE, A. (Ed.). **The confluence of vision and control. Lecture Notes in Control and Information Sciences**. London: Springer, 1998. p. 66–78.

CHAUMETTE, F.; HUTCHINSON, S. Visual servo control. i. basic approaches. **IEEE Robotics & Automation Magazine**, IEEE, v. 13, n. 4, p. 82–90, 2006.

CHEN, L.; HUANG, P.; MENG, Z. Convolutional multi-grasp detection using grasp path for rgbd images. **Robotics and Autonomous Systems**, Elsevier, v. 113, p. 94–103, 2019.

CHEN, L.-C.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K.; YUILLE, A. L. Semantic image segmentation with deep convolutional nets and fully connected crfs. **arXiv preprint arXiv:1412.7062**, 2014.

CHEN, L.-C.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K.; YUILLE, A. L. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 40, n. 4, p. 834–848, 2017.

CHEN, S.; LI, Y.; KWOK, N. M. Active vision in robotic systems: A survey of recent developments. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 30, n. 11, p. 1343–1377, 2011.

CHOLLET, F. et al. **Keras**. 2015.

CHU, F.-J.; XU, R.; VELA, P. A. Real-world multiobject, multigrasp detection. **IEEE Robotics and Automation Letters**, IEEE, v. 3, n. 4, p. 3355–3362, 2018.

COLLEWET, C.; MARCHAND, E. Photometric visual servoing. **IEEE Transactions on Robotics**, IEEE, v. 27, n. 4, p. 828–834, 2011.

CORKE, P. **Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised**. Switzerland: Springer, 2017. v. 118.

CRAIG, J. J. **Introduction to robotics: mechanics and control, 3/E**. India: Pearson Education, 2009.

CUTKOSKY, M. R. **Robotic grasping and fine manipulation**. Germany: Springer Science & Business Media, 2012. v. 6.

CUTKOSKY, M. R. et al. On grasp choice, grasp models, and the design of hands for manufacturing tasks. **IEEE Transactions on robotics and automation**, v. 5, n. 3, p. 269–279, 1989.

DEGUCHI, K. A direct interpretation of dynamic images with camera and object motions for vision guided robot control. **International Journal of Computer Vision**, v. 37, n. 1, p. 7–20, Jun 2000. ISSN 1573-1405.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: IEEE. **2009 Conference on Computer Vision and Pattern Recognition (CVPR)**. Miami, USA, 2009. p. 248–255.

DIEBEL, J. Representing attitude: Euler angles, unit quaternions, and rotation vectors. **Matrix**, v. 58, n. 15-16, p. 1–35, 2006.

DOSOVITSKIY, A.; FISCHER, P.; ILG, E.; HAUSSER, P.; HAZIRBAS, C.; GOLKOV, V.; SMAGT, P. v. d.; CREMERS, D.; BROX, T. Flownet: Learning optical flow with convolutional networks. In: IEEE. **2015 International Conference on Computer Vision (ICCV)**. Santiago, Chile, 2015.

DOZAT, T. Incorporating nesterov momentum into adam. 2016.

DU, G.; WANG, K.; LIAN, S. Vision-based robotic grasping from object localization, pose estimation, grasp detection to motion planning: A review. **arXiv preprint arXiv:1905.06658**, 2019.

DU, G.; WANG, K.; LIAN, S.; ZHAO, K. Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review. **Artificial Intelligence Review**, Springer Science and Business Media LLC, Aug 2020. ISSN 1573-7462.

EPPNER, C.; BROCK, O. Planning grasp strategies that exploit environmental constraints. In: IEEE. **Robotics and Automation (ICRA), 2015 IEEE International Conference on**. Seattle, USA, 2015. p. 4947–4952.

ESPIAU, B.; CHAUMETTE, F.; RIVES, P. A new approach to visual servoing in robotics. **ieee Transactions on Robotics and Automation**, IEEE, v. 8, n. 3, p. 313–326, 1992.

FINN, C.; LEVINE, S. Deep visual foresight for planning robot motion. In: IEEE. **2017 International Conference on Robotics and Automation (ICRA)**. Marina Bay Sands, Singapore, 2017. p. 2786–2793.

FUKUSHIMA, K. Cognitron: A self-organizing multilayered neural network. **Biological cybernetics**, Springer, v. 20, n. 3-4, p. 121–136, 1975.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. **Biological cybernetics**, Springer, v. 36, n. 4, p. 193–202, 1980.

GARCIA, G. J.; CORRALES, J. A.; POMARES, J.; TORRES, F. Survey of visual and force/tactile control of robots for physical interaction in spain. **Sensors**, Molecular Diversity Preservation International, v. 9, n. 12, p. 9689–9733, 2009.

GHAZAEI, G.; LAINA, I.; RUPPRECHT, C.; TOMBARI, F.; NAVAB, N.; NAZARPOUR, K. Dealing with ambiguity in robotic grasping via multiple predictions. In: SPRINGER. **Asian Conference on Computer Vision**. Perth, Australia, 2018. p. 38–55.

GLOCKER, B.; IZADI, S.; SHOTTON, J.; CRIMINISI, A. Real-time rgb-d camera relocalization. In: IEEE. **2013 International Symposium on Mixed and Augmented Reality (ISMAR)**. Adelaide, Australia, 2013. p. 173–179.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: PASCAL2. **Proceedings of the thirteenth international conference on artificial intelligence and statistics**. Sardinia, Italy, 2010. p. 249–256.

GOETSCHALCKX, R.; POUPART, P.; HOEY, J. Continuous correlated beta processes. In: AAAI. **Twenty-Second International Joint Conference on Artificial Intelligence**. Barcelona, Spain, 2011.

GOH, G. B.; HODAS, N. O.; VISHNU, A. Deep learning for computational chemistry. **Journal of computational chemistry**, Wiley Online Library, v. 38, n. 16, p. 1291–1307, 2017.

GOODALE, M. A.; MILNER, A. D.; JAKOBSON, L.; CAREY, D. A neurological dissociation between perceiving objects and grasping them. **Nature**, Nature Publishing Group, v. 349, n. 6305, p. 154, 1991.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A.; BENGIO, Y. **Deep learning**. Cambridge, USA: MIT Press, 2016. v. 1.

GU, Q.; SU, J.; BI, X. Attention grasping network: A real-time approach to generating grasp synthesis. In: IEEE. **2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)**. Yunnan, China, 2019. p. 3036–3041.

GUALTIERI, M.; PAS, A. T.; SAENKO, K.; PLATT, R. High precision grasp pose detection in dense clutter. In: IEEE. **2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Daejeon, Korea, 2016. p. 598–605.

GUALTIERI, M.; PLATT, R. Viewpoint selection for grasp detection. In: IEEE. **2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Vancouver, Canada, 2017. p. 258–264.

GUO, D.; SUN, F.; FANG, B.; YANG, C.; XI, N. Robotic grasping using visual and tactile sensing. **Information Sciences**, Elsevier, v. 417, p. 274–286, 2017.

GUO, Y.; LIU, Y.; OERLEMANS, A.; LAO, S.; WU, S.; LEW, M. S. Deep learning for visual understanding: A review. **Neurocomputing**, v. 187, p. 27 – 48, 2016. ISSN 0925-2312. Recent Developments on Deep Big Vision.

HABERMAS, J. **Truth and justification**. New Jersey: John Wiley & Sons, 2014.

HASHIMOTO, H.; KUBOTA, T.; SATO, M.; HARASHIMA, F. Visual control of robotic manipulator based on neural networks. **IEEE Transactions on Industrial Electronics**, IEEE, v. 39, n. 6, p. 490–496, 1992.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: IEEE. **Proceedings of the IEEE conference on computer vision and pattern recognition**. Las Vegas, USA, 2016. p. 770–778.

HILL, J. Real time control of a robot with a mobile camera. In: RIA. **9th Int. Symp. on Industrial Robots, 1979**. Washington DC, USA, 1979. p. 233–246.

HOSSAIN, D.; CAPI, G.; JINDAI, M. Evolution of deep belief neural network parameters for robot object recognition and grasping. **Procedia Computer Science**, Elsevier, v. 105, p. 153–158, 2017.

HOUAISS, A. **Houaiss Dictionary (In Portuguese)**. Rio de Janeiro: Objetiva, 2001.

HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. **The Journal of physiology**, Wiley Online Library, v. 160, n. 1, p. 106–154, 1962.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO 15539:2000(E), **Manipulating industrial robots – Object handling with grasp-type grippers – Vocabulary and presentation of characteristics**. Geneva, Switzerland, 2000.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. **arXiv preprint arXiv:1502.03167**, 2015.

IQBAL, H. **HarisIqbal88/PlotNeuralNet v1.0.0**. Zenodo, 2018. Available at: <https://doi.org/10.5281/zenodo.2526396>.

Islam, S. M. S.; Rahman, S.; Rahman, M. M.; Dey, E. K.; Shoyaib, M. Application of deep learning to computer vision: A comprehensive study. In: CNSER. **2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)**. Fukuoma, Japan, 2016. p. 592–597.

JAMES, S.; WOHLHART, P.; KALAKRISHNAN, M.; KALASHNIKOV, D.; IRPAN, A.; IBARZ, J.; LEVINE, S.; HADSELL, R.; BOUSMALIS, K. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In: IEEE. **2019 Conference on Computer Vision and Pattern Recognition (CVPR)**. Long Beach, California, 2019. p. 12627–12637.

JANG, E.; VIJAYANARASIMHAN, S.; PASTOR, P.; IBARZ, J.; LEVINE, S. End-to-end learning of semantic grasping. **arXiv preprint arXiv:1707.01932**, 2017.

JARRETT, K.; KAVUKCUOGLU, K.; LECUN, Y. et al. What is the best multi-stage architecture for object recognition? In: IEEE. **Computer Vision, 2009 IEEE 12th International Conference on**. Kyoto, Japan, 2009. p. 2146–2153.

JIA, Y.; SHELHAMER, E.; DONAHUE, J.; KARAYEV, S.; LONG, J.; GIRSHICK, R.; GUADARRAMA, S.; DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. In: ACM. **Proceedings of the 22nd ACM international conference on Multimedia**. Orlando, USA, 2014. p. 675–678.

JIANG, Y.; MOSESON, S.; SAXENA, A. Efficient grasping from rgbd images: Learning using a new rectangle representation. In: IEEE. **Robotics and Automation (ICRA), 2011 IEEE International Conference on**. Shanghai, China, 2011. p. 3304–3311.

JOHNS, E.; LEUTENEGGER, S.; DAVISON, A. J. Deep learning a grasp function for grasping under gripper pose uncertainty. In: IEEE. **2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Daejeon, Korea, 2016. p. 4461–4468.

KAPLAN, A.; HAENLEIN, M. Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence. **Business Horizons**, v. 62, n. 1, p. 15 – 25, 2019. ISSN 0007-6813.

KAPPLER, D.; BOHG, J.; SCHAAL, S. Leveraging big data for grasp planning. In: IEEE. **2015 IEEE International Conference on Robotics and Automation (ICRA)**. Seattle, USA, 2015. p. 4304–4311.

KENDALL, A.; GAL, Y.; CIPOLLA, R. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In: IEEE. **2018 Conference on Computer Vision and Pattern Recognition (CVPR)**. Salt Lake City, USA, 2018. p. 7482–7491.

KINGMA, D. P.; BA, J. **Adam: A Method for Stochastic Optimization**. 2014.

KINOVA ROBOTICS. **KINOVA Gen3 Ultra lightweight robot User Guide**. 2019.

KRAGIC, D.; CHRISTENSEN, H. I. et al. Survey on visual servoing for manipulation. **Computational Vision and Active Perception Laboratory, Fiskartorpsv**, Citeseer, v. 15, p. 2002, 2002.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: NIPSF. **Twenty-sixth Conference on Neural Information Processing Systems**. Lake Tahoe, USA, 2012. p. 1097–1105.

KROEMER, O.; DETRY, R.; PIATER, J.; PETERS, J. Combining active learning and reactive control for robot grasping. **Robotics and Autonomous systems**, Elsevier, v. 58, n. 9, p. 1105–1116, 2010.

KUMRA, S.; KANAN, C. Robotic grasp detection using deep convolutional neural networks. In: IEEE. **2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Vancouver, Canada, 2017. p. 769–776.

KWIATKOWSKI, R.; LIPSON, H. Task-agnostic self-modeling machines. **Science Robotics**, Science Robotics, v. 4, n. 26, p. 4, 2019.

LAMPE, T.; RIEDMILLER, M. Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In: IEEE. **The 2013 international joint conference on neural networks (IJCNN)**. Dallas, USA, 2013. p. 1–8.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Ieee, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Y. et al. Generalization and network design strategies. **Connectionism in perspective**, Citeseer, p. 143–155, 1989.

LEE, A. X.; LEVINE, S.; ABBEEL, P. Learning visual servoing with deep features and fitted q-iteration. **arXiv preprint arXiv:1703.11000**, 2017.

LENZ, I.; LEE, H.; SAXENA, A. Deep learning for detecting robotic grasps. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 34, n. 4-5, p. 705–724, 2015.

LEÓN, B.; MORALES, A.; SANCHO-BRU, J. **From robot to human grasping simulation**. Switzerland: Springer, 2014.

LEVINE, S.; FINN, C.; DARRELL, T.; ABBEEL, P. End-to-end training of deep visuomotor policies. **The Journal of Machine Learning Research**, JMLR. org, v. 17, n. 1, p. 1334–1373, 2016.

LEVINE, S.; PASTOR, P.; KRIZHEVSKY, A.; QUILLEN, D. Learning hand-eye coordination for robotic grasping with large-scale data collection. In: SPRINGER. **International symposium on experimental robotics**. Roppongi, Tokyo, 2016. p. 173–184.

LEVINE, S.; PASTOR, P.; KRIZHEVSKY, A.; IBARZ, J.; QUILLEN, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 37, n. 4-5, p. 421–436, 2018.

LIANG, H.; MA, X.; LI, S.; GÖRNER, M.; TANG, S.; FANG, B.; SUN, F.; ZHANG, J. Pointnetgpd: Detecting grasp configurations from point sets. In: IEEE. **2019 International Conference on Robotics and Automation (ICRA)**. Montreal, Canada, 2019. p. 3629–3635.

LITJENS, G.; KOOI, T.; BEJNORDI, B. E.; SETIO, A. A. A.; CIOMPI, F.; GHAFOORIAN, M.; LAAK, J. A. van der; GINNEKEN, B. van; SáNCHEZ, C. I. A survey on deep learning in medical image analysis. **Medical Image Analysis**, v. 42, p. 60 – 88, 2017. ISSN 1361-8415.

LIU, W.; WANG, Z.; LIU, X.; ZENG, N.; LIU, Y.; ALSAADI, F. E. A survey of deep neural network architectures and their applications. **Neurocomputing**, Elsevier, v. 234, p. 11–26, 2017.

MAHLER, J.; LIANG, J.; NIYAZ, S.; LASKEY, M.; DOAN, R.; LIU, X.; OJEA, J. A.; GOLDBERG, K. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. **arXiv preprint arXiv:1703.09312**, 2017.

MAHLER, J.; MATL, M.; LIU, X.; LI, A.; GEALY, D.; GOLDBERG, K. Dex-net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning. **2018 IEEE International Conference on Robotics and Automation (ICRA)**, IEEE, May 2018.

MAHLER, J.; POKORNY, F. T.; HOU, B.; RODERICK, M.; LASKEY, M.; AUBRY, M.; KOHLHOFF, K.; KRÖGER, T.; KUFFNER, J.; GOLDBERG, K. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In: IEEE. **2016 IEEE International Conference on Robotics and Automation (ICRA)**. Stockholm, Sweden, 2016. p. 1957–1964.

MALIS, E.; CHAUMETTE, F.; BOUDET, S. 2 1/2 d visual servoing. **IEEE Transactions on Robotics and Automation**, IEEE, v. 15, n. 2, p. 238–250, 1999.

MCCARTHY, J.; EARNEST, L.; REDDY, D. R.; VICENS, P. J. A computer with hands, eyes, and ears. In: ACM. **Proceedings of the December 9-11, 1968, fall joint computer conference, part I**. San Francisco, USA, 1968. p. 329–338.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MILLER, W. Sensor-based control of robotic manipulators using a general learning algorithm. **IEEE Journal on Robotics and Automation**, IEEE, v. 3, n. 2, p. 157–165, 1987.

MOHAMMED, H. B. R.; VINAYAKUMAR, R.; SOMAN, K. **A short review on Applications of Deep learning for Cyber security**. 2018.

MOHTA, K.; KUMAR, V.; DANIILIDIS, K. Vision-based control of a quadrotor for perching on lines. In: IEEE. **2014 IEEE International Conference on Robotics and Automation (ICRA)**. Hong Kong, China, 2014. p. 3130–3136.

MORRISON, D.; CORKE, P.; LEITNER, J. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. **arXiv preprint arXiv:1804.05172**, 2018.

MORRISON, D.; CORKE, P.; LEITNER, J. Learning robust, real-time, reactive robotic grasping. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, p. 183–201, 2019.

MUIS, A.; OHNISHI, K. Eye-to-hand approach on eye-in-hand configuration within real-time visual servoing. **IEEE/ASME transactions on Mechatronics**, IEEE, v. 10, n. 4, p. 404–410, 2005.

MUÑOZ, G. L. L. Análise comparativa das técnicas de controle servo-visual de manipuladores robóticos baseadas em posição e em imagem. 2011.

MURALI, A.; LI, Y.; GANDHI, D.; GUPTA, A. **Learning to Grasp Without Seeing**. 2018.

NESTEROV, Y. A method for unconstrained convex minimization problem with the rate of convergence o $(1/k^2)$. **Doklady AN USSR**, v. 269, p. 543–547, 1983.

NIKANDROVA, E.; KYRKI, V. Category-based task specific grasping. **Robotics and Autonomous Systems**, Elsevier, v. 70, p. 25–35, 2015.

NILSSON, N. J. **Principles of artificial intelligence**. Berlin: Springer, 2014.

OLIVEIRA, R.; ALVES, E.; MALQUI, C. Redes neurais convolucionais aplicadas à preensão robótica. In: **Anais do 13 Congresso Brasileiro de Inteligência Computacional**. Niterói, RJ: ABRICOM, 2017. p. 1–11.

PALMIERI, G.; PALPACELLI, M.; BATTISTELLI, M.; CALLEGARI, M. A comparison between position-based and image-based dynamic visual servoings in the control of a translating parallel manipulator. **Journal of Robotics**, Hindawi, v. 2012, 2012.

PANDYA, H.; KRISHNA, K. M.; JAWAHAR, C. Servoing across object instances: Visual servoing for object category. In: IEEE. **2015 IEEE International Conference on Robotics and Automation (ICRA)**. Seattle, USA, 2015. p. 6011–6018.

PARK, D.; CHUN, S. Y. Classification based grasp detection using spatial transformer network. **arXiv preprint arXiv:1803.01356**, 2018.

PAS, A. ten; PLATT, R. Using geometry to detect grasp poses in 3d point clouds. **Robotics Research**, Springer, v. 1, p. 307, 2017.

PIATER, J.; JODOGNE, S.; DETRY, R.; KRAFT, D.; KRÜGER, N.; KROEMER, O.; PETERS, J. Learning visual representations for perception-action systems. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 30, n. 3, p. 294–307, 2011.

PIEPMEIER, J. A.; MCMURRAY, G. V.; LIPKIN, H. Uncalibrated dynamic visual servoing. **IEEE Transactions on Robotics and Automation**, IEEE, v. 20, n. 1, p. 143–147, 2004.

PINTO, L.; GUPTA, A. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In: IEEE. **Robotics and Automation (ICRA), 2016 IEEE International Conference on**. Stockholm, Sweden, 2016. p. 3406–3413.

POLYAK, B. T. Some methods of speeding up the convergence of iteration methods. **USSR Computational Mathematics and Mathematical Physics**, No longer published by Elsevier, v. 4, n. 5, p. 1–17, 1964.

PONTI, M. A.; COSTA, G. B. P. da. Como funciona o deep learning. **arXiv preprint arXiv:1806.07908**, 2018.

PRATTICHIZZO, D.; TRINKLE, J. C. Grasping. In: **Springer handbook of robotics**. USA: Springer, 2016. p. 955–988.

QIAN, N. On the momentum term in gradient descent learning algorithms. **Neural networks**, Elsevier, v. 12, n. 1, p. 145–151, 1999.

QUILLEN, D.; JANG, E.; NACHUM, O.; FINN, C.; IBARZ, J.; LEVINE, S. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. **arXiv preprint arXiv:1802.10264**, 2018.

RAI, A.; SUTANTO, G.; SCHAAL, S.; MEIER, F. Learning feedback terms for reactive planning and control. In: IEEE. **2017 IEEE International Conference on Robotics and Automation (ICRA)**. Marina Bay Sands, Singapore, 2017. p. 2184–2191.

RAMACHANDRAM, D.; RAJESWARI, M. A short review of neural network techniques in visual servoing of robotic manipulators. In: AIAI. **Malaysia-Japan Seminar On Artificial Intelligence Applications In Industry**. Kuala Lumpur, Malaysia, 2003. p. 2425.

RATLIFF, N.; BAGNELL, J. A.; SRINIVASA, S. S. Imitation learning for locomotion and manipulation. In: IEEE. **2007 7th IEEE-RAS International Conference on Humanoid Robots**. Pittsburgh, USA, 2007. p. 392–397.

REDMON, J.; ANGELOVA, A. Real-time grasp detection using convolutional neural networks. In: IEEE. **Robotics and Automation (ICRA), 2015 IEEE International Conference on**. Seattle, 2015. p. 1316–1322.

REN, G.; SHAO, Z.; GUAN, Y.; QU, Y.; TAN, J.; WEI, H.; TONG, G. A fast search algorithm based on image pyramid for robotic grasping. In: IEEE. **2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Vancouver, Canada, 2017. p. 6520–6525.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In: NIPSF. **Advances in neural information processing systems**. Montréal, Canada, 2015. p. 91–99.

RIBEIRO, E. G.; GRASSI, V. Fast convolutional neural network for real-time robotic grasp detection. In: IEEE. **2019 19th International Conference on Advanced Robotics (ICAR)**. Belo Horizonte, Brazil, 2019. p. 49–54.

ROBERT, C. Machine learning, a probabilistic perspective. **CHANCE**, Taylor & Francis, v. 27, n. 2, p. 62–63, 2014.

ROBOTIQ INC. **2F-85 & 2F-140 - Instruction Manual**. 2019.

ROCCO, I.; ARANDJELOVIC, R.; SIVIC, J. Convolutional neural network architecture for geometric matching. **IEEE transactions on pattern analysis and machine intelligence**, v. 41, n. 11, p. 2553, 2019.

RODRIGUEZ, A.; MASON, M. T.; FERRY, S. From caging to grasping. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 31, n. 7, p. 886–900, 2012.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

ROSS, S.; GORDON, G.; BAGNELL, D. A reduction of imitation learning and structured prediction to no-regret online learning. In: AISTATS. **Proceedings of the fourteenth international conference on artificial intelligence and statistics**. Ft. Lauderdale, USA, 2011. p. 627–635.

RUBERT, C.; KAPPLER, D.; MORALES, A.; SCHAAL, S.; BOHG, J. On the relevance of grasp metrics for predicting grasp success. In: IEEE. **Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on**. Vancouver, Canada, 2017. p. 265–272.

RUDER, S. An overview of gradient descent optimization algorithms. **arXiv preprint arXiv:1609.04747**, 2016.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. Malaysia: Pearson Education Limited, 2016.

SADEGHI, F. Divis: Domain invariant visual servoing for collision-free goal reaching. **arXiv preprint arXiv:1902.05947**, 2019.

SADEGHI, F.; TOSHEV, A.; JANG, E.; LEVINE, S. Sim2real viewpoint invariant visual servoing by recurrent control. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 4691–4699.

SALISBURY, J. K.; CRAIG, J. J. Articulated hands: Force control and kinematic issues. **The International journal of Robotics research**, SAGE Publications Sage UK: London, England, v. 1, n. 1, p. 4–17, 1982.

SAMPEDRO, C.; RODRIGUEZ-RAMOS, A.; GIL, I.; MEJIAS, L.; CAMPOY, P. Image-based visual servoing controller for multirotor aerial robots using deep reinforcement learning. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Madrid, Spain, 2018. p. 979–986.

SANDERSON, A. Image based visual servo control using relational graph error signal. In: IEEE. **Proc. Int. Conf. Cybernetics and Society**. Cambridge, USA, 1980.

SAXENA, A.; DRIEMEYER, J.; NG, A. Y. Robotic grasping of novel objects using vision. **The International Journal of Robotics Research**, Sage Publications Sage UK: London, England, v. 27, n. 2, p. 157–173, 2008.

SAXENA, A.; PANDYA, H.; KUMAR, G.; GAUD, A.; KRISHNA, K. M. Exploring convolutional networks for end-to-end visual servoing. In: IEEE. **2017 IEEE International Conference on Robotics and Automation (ICRA)**. Marina Bay Sands, Singapore, 2017. p. 3817–3823.

SCHALKOFF, R. J. **Artificial Intelligence Engine**. New York, USA: McGraw-Hill, Inc., 1990.

SCHMIDT, P.; VAHRENKAMP, N.; WÄCHTER, M.; ASFOUR, T. Grasping of unknown objects using deep convolutional neural networks based on depth images. In: IEEE. **2018 IEEE International Conference on Robotics and Automation (ICRA)**. Brisbane, Australia, 2018. p. 6831–6838.

SHADEMAN, A.; FARAHMAND, A.-M.; JÄGERSAND, M. Robust jacobian estimation for uncalibrated visual servoing. In: IEEE. **2010 IEEE International Conference on Robotics and Automation**. Anchorage, USA, 2010. p. 5564–5569.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. **Understanding machine learning: From theory to algorithms**. Cambridge, UK: Cambridge university press, 2014.

SHAVIT, Y.; FERENS, R. Introduction to camera pose estimation with deep learning. **arXiv preprint arXiv:1907.05272**, 2019.

SHI, H.; LI, X.; HWANG, K.-S.; PAN, W.; XU, G. Decoupled visual servoing with fuzzyq-learning. **IEEE Transactions on Industrial Informatics**, IEEE, v. 14, n. 1, p. 241–252, 2016.

SHIMOGA, K. B. Robot grasp synthesis algorithms: A survey. **The International Journal of Robotics Research**, Sage Publications Sage CA: Thousand Oaks, CA, v. 15, n. 3, p. 230–266, 1996.

SHIRAI, Y.; INOUE, H. Guiding a robot by visual feedback in assembling tasks. **Pattern recognition**, Citeseer, v. 5, n. 2, p. 99–106, 1973.

SILVA, I. N. D.; SPATTI, D. H.; FLAUZINO, R. A.; LIBONI, L. H. B.; ALVES, S. F. dos R. Artificial neural networks. **Cham: Springer International Publishing**, Springer, 2017.

SILVEIRA, G. On intensity-based nonmetric visual servoing. **IEEE Transactions on Robotics**, IEEE, v. 30, n. 4, p. 1019–1026, 2014.

SILVEIRA, G.; MALIS, E. Direct visual servoing: Vision-based estimation and control using only nonmetric information. **IEEE Transactions on Robotics**, IEEE, v. 28, n. 4, p. 974–980, 2012.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. et al. **Robot modeling and control**. New York, USA: Wiley, 2006. v. 3.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

STULP, F.; THEODOROU, E.; BUCHLI, J.; SCHAAL, S. Learning to grasp under uncertainty. In: IEEE. **2011 IEEE International Conference on Robotics and Automation**. Shanghai, China, 2011. p. 5703–5708.

SUN, X.; ZHU, X.; WANG, P.; CHEN, H. A review of robot control with visual servoing. In: IEEE. **2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)**. Tianjin, China, 2018. p. 116–121.

SÜNDERHAUF, N.; BROCK, O.; SCHEIRER, W.; HADSELL, R.; FOX, D.; LEITNER, J.; UPCROFT, B.; ABBEEL, P.; BURGARD, W.; MILFORD, M. et al. The limits and potentials of deep learning for robotics. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 37, n. 4-5, p. 405–420, 2018.

SUTSKEVER, I.; MARTENS, J.; DAHL, G.; HINTON, G. On the importance of initialization and momentum in deep learning. In: NSF. **International conference on machine learning**. Atlanta, USA, 2013. p. 1139–1147.

TANNER, M. A.; WONG, W. H. The calculation of posterior distributions by data augmentation. **Journal of the American statistical Association**, Taylor & Francis, v. 82, n. 398, p. 528–540, 1987.

TOBIN, J.; BIEWALD, L.; DUAN, R.; ANDRYCHOWICZ, M.; HANDA, A.; KUMAR, V.; MCGREW, B.; RAY, A.; SCHNEIDER, J.; WELINDER, P. et al. Domain randomization and generative models for robotic grasping. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Madrid, Spain, 2018. p. 3482–3489.

TOBIN, J.; FONG, R.; RAY, A.; SCHNEIDER, J.; ZAREMBA, W.; ABBEEL, P. Domain randomization for transferring deep neural networks from simulation to the real world. In: IEEE. **2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)**. Vancouver, Canada, 2017. p. 23–30.

TROTTIER, L.; GIGUERE, P.; CHAIB-DRAA, B. Sparse dictionary learning for identifying grasp locations. In: IEEE. **2017 IEEE Winter Conference on Applications of Computer Vision (WACV)**. Santa Rosa, USA, 2017. p. 871–879.

TRUCCO, E.; VERRI, A. **Introductory techniques for 3-D computer vision**. USA: Prentice Hall Englewood Cliffs, 1998. v. 201.

UNGERLEIDER, L. G.; HAXBY, J. V. 'what'and 'where'in the human brain. **Current opinion in neurobiology**, Elsevier, v. 4, n. 2, p. 157–165, 1994.

VERES, M.; MOUSSA, M.; TAYLOR, G. W. Modeling grasp motor imagery through deep conditional generative models. **IEEE Robotics and Automation Letters**, IEEE, v. 2, n. 2, p. 757–764, 2017.

VIERECK, U.; PAS, A.; SAENKO, K.; PLATT, R. Learning a visuomotor controller for real world robotic grasping using simulated depth images. In: CORL. **Conference on Robot Learning**. Mountain View, USA, 2017. p. 291–300.

WANG, A. S.; ZHANG, W.; TRONIAK, D.; LIANG, J.; KROEMER, O. Homography-based deep visual servoing methods for planar grasps. In: IEEE. **2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Macau, China, 2019. p. 6570–6577.

WANG, Z.; LI, Z.; WANG, B.; LIU, H. Robot grasp detection using multimodal deep convolutional neural networks. **Advances in Mechanical Engineering**, v. 8, n. 9, p. 1–12, 2016.

WEI, G.-Q.; HIRZINGER, G. Multisensory visual servoing by a neural network. **IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)**, IEEE, v. 29, n. 2, p. 276–280, 1999.

WEI, J.; LIU, H.; YAN, G.; SUN, F. Robotic grasping recognition using multi-modal deep extreme learning machine. **Multidimensional Systems and Signal Processing**, Springer, v. 28, n. 3, p. 817–833, 2017.

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

YANG, M.; YU, K.; ZHANG, C.; LI, Z.; YANG, K. Denseaspp for semantic segmentation in street scenes. In: IEEE. **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. Salt Lake City, USA, 2018. p. 3684–3692.

ZEILER, M. D. **ADADELTA: An Adaptive Learning Rate Method**. 2012.

ZHANG, F.; LEITNER, J.; MILFORD, M.; UPCROFT, B.; CORKE, P. Towards vision-based deep reinforcement learning for robotic motion control. **arXiv preprint arXiv:1511.03791**, 2015.

ZHANG, H.; LAN, X.; BAI, S.; ZHOU, X.; TIAN, Z.; ZHENG, N. Roi-based robotic grasp detection for object overlapping scenes. **arXiv preprint arXiv:1808.10313**, 2018.

ZHOU, X.; LAN, X.; ZHANG, H.; TIAN, Z.; ZHANG, Y.; ZHENG, N. Fully convolutional grasp detection network with oriented anchor box. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. Madrid, Spain, 2018. p. 7223–7230.

ZHOU, Y.-T.; CHELLAPPA, R. Computation of optical flow using a neural network. In: IEEE. **IEEE International Conference on Neural Networks**. Anchorage, USA, 1988. v. 1998, p. 71–78.

APPENDIX A – BACKPROPAGATION ALGORITHM

Once the backpropagation algorithm is used in an optimized way for training convolutional neural networks, its basic functioning is developed below. The procedure consists of two stages. The first is known as the forward phase, in which an input k is simply propagated through the layers of the network, without any change in their weights, to obtain the output $y_j(k)$. The second is the backward phase, in which $y_j(k)$ is compared with the desired output value $d_j(k)$, generating an error value, which will be propagated by the network, updating its weights. The value of this error is obtained from a function, whose gradient will guide the backpropagation process.

Considering a training set with p samples, $d_j(k)$ the desired output value for the k^{th} sample and $y_j(k)$ the value obtained by the network for that same sample, the prediction error is given by Eq. A.1. In the equation, j is a neuron in the output layer and m is the total number of neurons in this layer.

$$E(k) = \frac{1}{2}\sum_{j=1}^{m}(d_j(k) - y_j(k))^2 \tag{A.1}$$

The Mean Squared Error (MSE) is given by the average of $E(k)$ for the p samples, according to Eq. A.2.

$$E_{mse} = \frac{1}{p}\sum_{k=1}^{p}E_k \tag{A.2}$$

The choice between Eq. A.1 and Eq. A.2 as the function to be minimized is the main difference between the Gradient Descent (GD) and Stochastic Gradient Descent (SGD) algorithms (PONTI; COSTA, 2018). For cases in which the training set is small, Eq. A.2 is used because, once there is memory space to allocate all training samples, the post forward phase error is summed iteratively until all training samples have been propagated over the network. Only then, the MSE is calculated and backpropagated (GOODFELLOW et al., 2016). This is the GD procedure.

For cases in which the available memory is limited given the large training set, as in deep learning problems, it is necessary to use random samples of data instead of analyzing all existing instances. Thus, for the SGD case, a sample is propagated through the network and its error is retropropagated right afterward by applying Eq. A.1.

In practice, although, the most common is to use mini-batches with fixed size containing random instances of the data. If, for example, the size of the mini-batch is m and the size of the data is M, an approximation of the GD method is obtained after m/M

iterations. In each iteration, the GD is applied considering only instances of the current mini-batch (PONTI; COSTA, 2018). This case, although formally described as Mini-batch Gradient Descent, is commonly understood as an extension of the SGD.

The backpropagation algorithm for a mini-batch with p instances tries to minimize the Eq. A.3, which is written from Eqs. A.1 and A.2.

$$E = E_{mse} = \frac{1}{2p}\sum_{k=1}^{p}\sum_{j=1}^{m}(d_j(k) - y_j(k))^2 \tag{A.3}$$

The gradient of this function with respect to the network weights is given by the chain rule, as shown in Eq. A.4. Considering a PMC with input layer, two hidden layers and output layer, y_j is the output of the j^{th} output neuron, u_j is the input of this neuron and W_{ji} is the vector of weights that relates the output of the 2^{nd} hidden layer neurons $\bar{y}$, with the neuron j.

$$\nabla E = \frac{\partial E}{\partial W_{ji}} = \frac{\partial E}{\partial y_j}.\frac{\partial y_j}{\partial u_j}.\frac{\partial u_j}{\partial W_{ji}} \tag{A.4}$$

The partial derivation of each term results in Eqs. A.5.

$$\frac{\partial u_j}{\partial W_{ji}} = \bar{y}_i \qquad \frac{\partial y_j}{\partial u_j} = g'(u_j) \qquad \frac{\partial E}{\partial y_j} = -\frac{1}{p}\sum_{k=1}^{p}(d_j(k) - y_j(k)) \tag{A.5}$$

Replacing these equations in Eq. A.4, the gradient is rewritten as the Eq. A.6.

$$\nabla E = -\frac{1}{p}\sum_{k=1}^{p}(d_j(k) - y_j(k)).g'(u_j).\bar{y}_i \tag{A.6}$$

Thus, the variation of the weights of this layer must happen in the opposite direction to the gradient and must be weighted by a learning rate η, to avoid instability in training, as indicated by the Eq. A.7.

$$\Delta W_{ji} = -\eta\nabla E = \eta.\frac{1}{p}\sum_{k=1}^{p}(d_j(k) - y_j(k)).g'(u_j).\bar{y}_i \tag{A.7}$$

Or, iteratively,

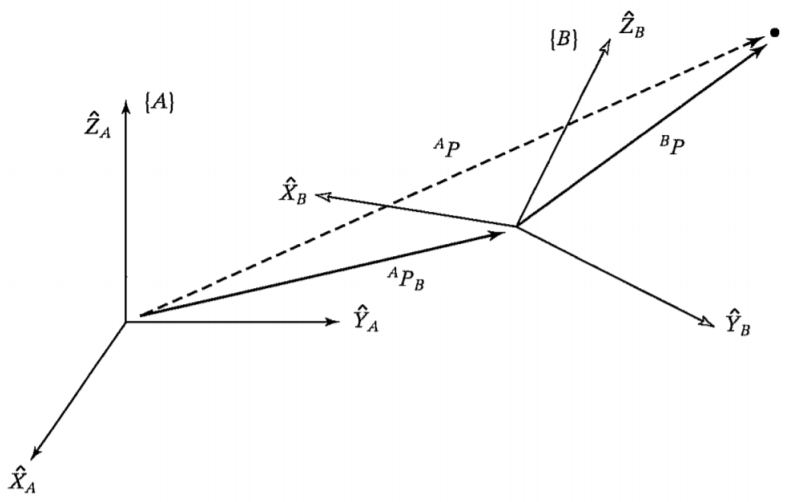
$$W_{ji}(t+1) = W_{ji}(t) + \eta.\frac{1}{p}\sum_{k=1}^{p}(d_j(k) - y_j(k)).g'(u_j).\bar{y}_i. \tag{A.8}$$

The same idea is used to update the weights of the intermediate layers. For a deeper understanding, it is worth reading Silva et al. (2017), Goodfellow et al. (2016), Bishop (2006) and Bottou (1998).

APPENDIX B – SPATIAL TRANSFORMATIONS

A point in the Euclidean space $\mathbb{R}^n$ can be described in homogeneous coordinates, using a $(n+1)$-dimensional vector. Thus, a point $\mathbf{P} = (p_1, p_2, ..., p_n)$ in Cartesian coordinates, becomes $(\mathbf{P}, 1) = (kp_1, kp_2, ..., kp_n, k)$ in homogeneous coordinates, where k is a non-zero scale factor (MUÑOZ, 2011). Through homogeneous coordinates it is possible to represent transformations between Coordinate Systems (CS) in a more compact way. Fig. 49 exemplify one of those transformations.

Figure 49: Representation of a point in two coordinate systems



Source: Adapted from Craig (2009)

The figure illustrates the coordinate systems $\{A\}$ and $\{B\}$ and a point $\mathbf{P}$ in space represented in terms of the frame $\{A\}$, ${}^A\mathbf{P}$, and $\{B\}$, ${}^B\mathbf{P}$. The two coordinates are related by a ${}^A\mathbf{P}_B$ translation and by a ${}^AR_B$ rotation. That is, $\{B\}$ can be obtained from $\{A\}$ using these operations. If the objective is to represent the point P related to the frame $\{B\}$, one must first invert the vector that indicates the translation between the frames so that it is described in terms of $\{B\}$ and the same must be done with the rotation matrix ${}^AR_B$, obtaining ${}^BR_A$. Thus. ${}^B\mathbf{P}$ can be obtained from:

$$ {}^B\mathbf{P} = {}^BR_A{}^A\mathbf{P} + {}^B\mathbf{P}_A. \tag{B.1} $$

Or, in homogeneous coordinates:

$$ {}^b\tilde{\mathbf{P}} = \mathbf{T}^a\tilde{\mathbf{P}}, \tag{B.2} $$

where $\tilde{\mathbf{P}} = [p_x, p_y, p_z, 1]$ and $\mathbf{T}$ is:

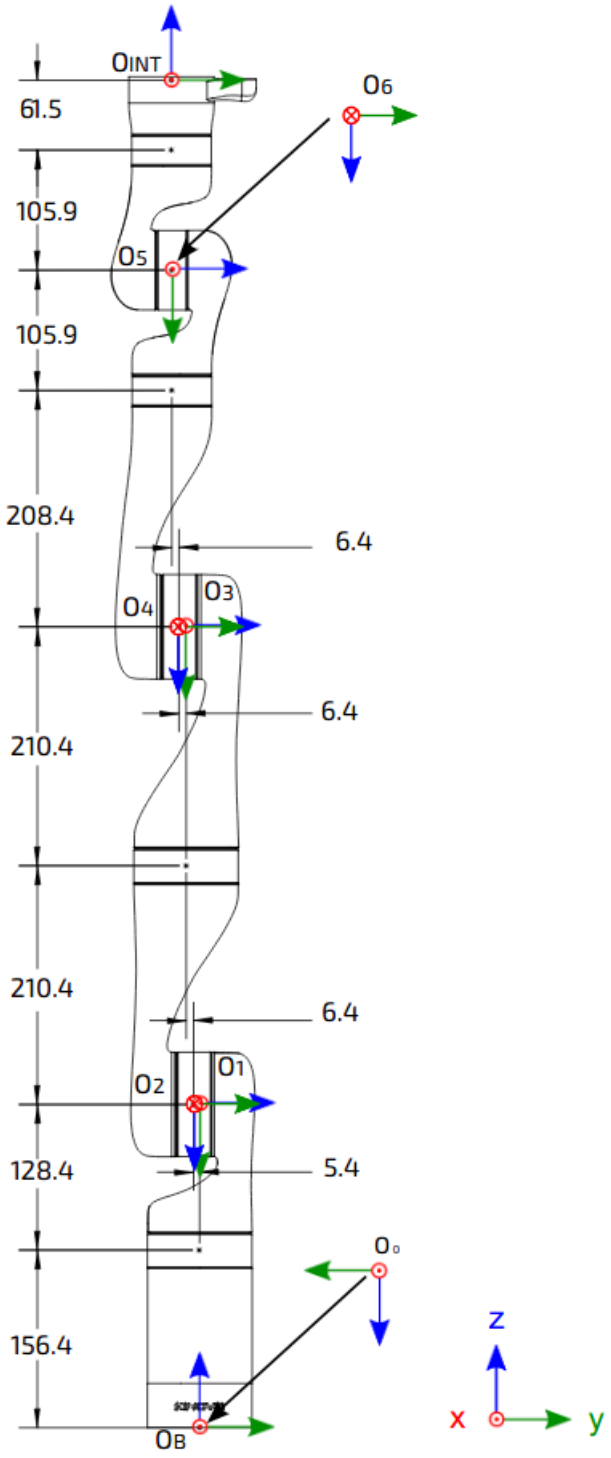
$$\begin{bmatrix} {}^{B}R_{A} & {}^{B}\mathbf{P}_{A} \\ 0 & 1 \end{bmatrix}. \tag{B.3}$$

Note that the notation used here is the standard form, in which a leading superscript denotes the reference frame, in which a set of coordinates are defined (CHAUMETTE; HUTCHINSON, 2006). So, for example, ${}^{A}\mathbf{T}_{B}$ represents the homogeneous transformation that expresses the origin of the frame $\{B\}$ relative to frame $\{A\}$. As demonstrated above, this transformation can also be used to represent a point, defined in terms of the frame $\{B\}$, in the frame $\{A\}$.

For further details about spatial transformations, it is recommended the reading of Craig (2009) and Spong et al. (2006).

ANNEX A – KINOVA GEN3'S DENAVIT-HARTENBERG PARAMETERS

Figure 50: Denavit-Hartenberg parameters of the Kinova Gen3



Source: Kinova Robotics (2019)