



**HAL**  
open science

# Apprentissage profond auto-supervisé par simulation pour la saisie robotique adaptative

Amaury Depierre

► **To cite this version:**

Amaury Depierre. Apprentissage profond auto-supervisé par simulation pour la saisie robotique adaptative. Autre. Université de Lyon, 2021. Français. NNT : 2021LYSEC019 . tel-03871964

**HAL Id: tel-03871964**

**<https://theses.hal.science/tel-03871964v1>**

Submitted on 25 Nov 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NNT : 2021LYSEC19

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON  
RÉALISÉE AU SEIN DE L'ÉCOLE CENTRALE DE LYON

délivrée par l'École Doctorale N°512 InfoMaths  
Spécialité : Informatique

---

**Simulation Empowered Self-Supervised Deep Learning  
for Adaptive Robotic Grasping**

---

présentée et soutenue à huis clos par  
**Amaury DEPIERRE**  
le 21 Mai 2021

Directeur de thèse : **Pr. Liming CHEN**  
Co-directeur de thèse : **Dr. Emmanuel DELLANDRÉA**

**Jury**

---

<b>Mme. Véronique Perdereau</b>	Professeur, ISIR	Présidente
<b>M. Sylvain Calinon</b>	Senior Researcher, IDIAP	Rapporteur
<b>M. Jan Peters</b>	Professor, TU Darmstadt	Rapporteur
<b>Mme. Alessandra Sciutti</b>	Researcher, IIT	Examinatrice
<b>M. Matthieu Grard</b>	Ingénieur, Siléane	Examineur
<b>M. Liming Chen</b>	Professeur, ECL	Directeur de thèse
<b>M. Emmanuel Dellandréa</b>	Maître de conférence, ECL	Co-directeur de thèse





---

## ABSTRACT

---

Mastering robotic grasping is a necessary skill for a robot to perform tasks involving the manipulation of one or more objects. As automation increases, these tasks are nowadays found in more and more fields of industry such as car manufacturing, waste sorting, or food processing. In such cases, the environment can not be fully controlled, and it is, therefore, necessary to use systems capable of analyzing their environment and interacting with them, instead of just doing a set of preprogrammed tasks.

To do this, detecting the configuration of instances of objects in a scene using their 3D models is not always possible. In some cases, the models are not available. It is then possible to use a geometric approach to detect opportunities for grasp positions in images. However, these approaches require many parameters to be set manually so that the geometric criteria are adapted to the scene. To overcome this problem, methods based on deep learning can be used: from a large number of annotated images and good grasping positions, a deep neural network can learn how to predict good positions on new images. The acquisition of such a large amount of annotated data, however, represents an obstacle for robotics.

In collaboration with Siléane, a French industrial robotics company, the aim of this work is therefore to develop solutions for learning robotic grasping through simulated data, which are available in large quantities. This work has two main contributions.

Firstly, we propose a new neural network architecture that predicts grasp positions for a parallel-jaw gripper on images of isolated objects. Previous state-of-the-art work used small amounts of manually annotated data. In this work, we build a new large-scale dataset of synthetic images with automatically generated annotations through physics simulation and train our network using it. The use of a large amount of diverse data, rather than just a few images, allows the network to be trained on a wider range of situations, and thus be able to handle more different unknown cases.

Secondly, the work presented here deals with the detection of grasp locations in bin-picking context, *i.e.* in scenes with many objects occluding each other. While traditional approaches for this problem use local information, classifying the potential quality of a grasp according to the surrounding data in the image, our proposed network completes this information by adding the notion of object instances. Trained on self-supervised simulated images, it can thus estimate the quality of a grasp position based not only on local information but also on the global context of the object present at the considered posi-

tion in the image. Based on extensive experiments, we show that this double approach allows improving the quality of predictions, both in a simulated environment and in real robotic tasks.

**Keywords:** deep learning, simulation, robotic grasping, bin-picking, neural networks, synthetic data, self-supervision

---

## RÉSUMÉ

---

La maîtrise de la préhension robotique par un robot est nécessaire pour l'accomplissement de toutes les tâches nécessitant la manipulation d'un ou plusieurs objets. Avec une automatisation croissante de l'industrie, ces tâches se retrouvent aujourd'hui dans de nombreux domaines de l'industrie tels que l'automobile, le tri des déchets ou encore l'agro-alimentaire. Dans de tels cas, l'environnement ne peut pas être totalement contrôlé, et il est donc nécessaire de faire appel à des systèmes capables d'analyser leur environnement pour interagir avec eux.

Pour ce faire, on ne peut pas toujours utiliser les modèles 3D des objets pour détecter la configuration des instances dans une scène. Dans certains cas, les modèles ne sont pas disponibles. Il est alors possible d'utiliser une approche géométrique pour détecter des opportunités de prises robotiques dans des images. Cependant, ces approches nécessitent de régler de nombreux paramètres manuellement pour que les critères géométriques soient adaptés à la scène. Pour pallier à ce problème, il est possible d'utiliser des méthodes à base d'apprentissage automatique : à partir d'un grand nombre d'exemples d'images et de bonnes positions de prises, un réseau de neurones profond est capable d'apprendre à prédire des bonnes positions sur de nouvelles images. L'acquisition d'une telle quantité de données annotées représente cependant un obstacle pour la robotique.

En collaboration avec Siléane, une entreprise de robotique industrielle française, l'objectif de ce travail est donc de développer des solutions pour l'apprentissage de la préhension robotique à travers les données simulées, disponibles en grandes quantités. Dans ce domaine, ce travail apporte deux contributions.

Premièrement, nous proposons une nouvelle architecture de réseau de neurones permettant de prédire des positions de prises pour une pince à mors parallèles sur des images d'objets isolés. Les précédents travaux de l'état de l'art utilisaient de faibles quantités de données annotées manuellement. Dans ce travail, nous construisons une très grande base de données d'images synthétiques annotées automatiquement par simulation physique, que nous utilisons pour entraîner notre réseau. L'utilisation d'une grande quantité de données diversifiées, plutôt que de quelques images seulement, permet au réseau d'être entraîné sur des situations plus variées, et ainsi de pouvoir gérer de plus nombreux cas différents.

Dans un deuxième temps, les travaux présentés ici s'intéressent à la détection de prises au sein d'un vrac, *i.e.* d'un enchevêtrement de nombreux objets avec de forts recouvrements entre eux. Alors que les

approches traditionnelles dans ce domaine utilisent une vision locale, classifiant la qualité potentielle d'une prise en fonction des données alentours dans l'image, le réseau proposé complète cette information en ajoutant la notion d'instances d'objets. Entraîné sur des images simulées de manière auto-supervisée, il peut ainsi estimer la qualité d'une position de prise en se basant non seulement sur une information locale, mais également sur le contexte global de l'objet présent à la position considérée dans l'image. A partir de plusieurs expériences, nous montrons que cette double approche permet d'améliorer la qualité des prédictions, aussi bien dans un environnement simulé que dans de vrais contextes robotiques.

**Mots-clés :** apprentissage profond, simulation, préhension robotique, dévracage, réseau de neurones, données synthétiques, auto-supervision

I see a beautiful city and a brilliant people rising from this abyss. [...] I see the lives for which I lay down my life, peaceful, useful, prosperous and happy. [...] I see that I hold a sanctuary in their hearts, and in the hearts of their descendants, generations hence. [...] It is a far, far better thing that I do, than I have ever done; it is a far, far better rest that I go to than I have ever known.

— Charles Dickens, *A Tale of Two Cities*



---

## REMERCIEMENTS

---

Je tiens à remercier Liming Chen et Emmanuel Dellandréa qui ont accepté d'encadrer mes travaux de thèse, pour m'avoir consacré de leur temps et pour m'avoir prodigué leurs conseils.

Je remercie la société Siléane, et son fondateur Hervé Henry, pour m'avoir fait confiance pour les travaux de recherche menés dans le cadre de cette thèse CIFRE.

Merci également à Romain et Matthieu pour leurs conseils, à la fois scientifiques et professionnels.

Merci à ma famille qui m'a soutenu tout au long de ce travail, et qui m'a accompagné dans mes choix aussi bien personnels que professionnels.

Merci à tous mes collègues à Siléane et au LIRIS pour leur soutien et les moments enrichissants passés ensemble. Merci plus particulièrement à Jean-Louis, Florian, Julien, Maxime et Thomas.

Merci à Thomas, Benjamin, Maël, Camilo et Baptiste pour les nombreuses discussions intéressantes que nous avons pu partager.

Merci à Ghislain et Benoît pour m'avoir accompagné dans mes projets personnels menés en parallèle de la thèse, et grâce auxquels j'ai beaucoup appris.

Merci aux membres du jury d'avoir accepté la tâche d'évaluer les travaux réalisés dans le cadre de cette thèse.

Et enfin, merci à toutes les personnes qui ont contribué de près ou de loin à l'accomplissement de ce travail, mais que je ne peux toutes nommer.





---

## CONTENTS

---

Abstract	iii
Résumé	v
Remerciements	ix
List of Figures	xv
List of Tables	xvii
Acronyms and notations	xix
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Problems and Objectives . . . . .	2
1.2 Contributions . . . . .	3
1.2.1 Grasping isolated objects . . . . .	4
1.2.2 Grasping in piles . . . . .	5
1.3 Publications . . . . .	6
1.4 Contents . . . . .	7
<b>2 STATE-OF-THE-ART</b>	<b>9</b>
2.1 Robotic Grasping Overview . . . . .	9
2.2 Model-based grasping . . . . .	12
2.2.1 Object pose detection . . . . .	12
2.2.2 Grasp generation from object pose . . . . .	14
2.3 Model-free grasping . . . . .	14
2.3.1 Discriminative methods . . . . .	15
2.3.2 Generative methods . . . . .	15
2.3.3 Reinforcement-Learning methods . . . . .	17
2.3.4 Model-free grasping in piles . . . . .	17
2.3.4.1 Gripper-oriented methods . . . . .	18
2.3.4.2 Object-oriented methods . . . . .	18
2.3.4.3 Discussion . . . . .	19
2.4 Datasets for robotic grasping . . . . .	19
2.5 Conclusion . . . . .	22
<b>3 ISOLATED OBJECT GRASPING</b>	<b>23</b>
3.1 Generating a large-scale dataset . . . . .	23
3.1.1 Dataset generation . . . . .	24
3.1.1.1 Scene creation . . . . .	24
3.1.1.2 Image rendering . . . . .	25
3.1.1.3 Annotation generation . . . . .	26
3.1.1.4 Simulated grasp trial-based criterion . . . . .	29

3.1.2	Dataset validation . . . . .	31
3.1.2.1	Training setup . . . . .	31
3.1.2.2	Cross-dataset evaluation . . . . .	32
3.1.2.3	Real-robot testing . . . . .	33
3.2	Learning to grasp objects . . . . .	34
3.2.1	Grasp prediction using a scorer . . . . .	35
3.2.1.1	Anchor boxes . . . . .	35
3.2.1.2	Network architecture . . . . .	36
3.2.1.3	Loss functions . . . . .	38
3.2.1.4	Training procedure . . . . .	39
3.2.2	Experimental evaluation . . . . .	40
3.2.2.1	Experimental setup . . . . .	40
3.2.2.2	Results . . . . .	41
3.2.2.3	Real robot testing . . . . .	43
3.3	Conclusion . . . . .	44
3.3.1	Summary . . . . .	44
3.3.2	Contributions . . . . .	44
3.3.3	Extension to bin-picking . . . . .	45
4	SELF-SUPERVISED OBJECT AWARE BIN-PICKING . . . . .	47
4.1	Self-supervised training . . . . .	47
4.1.1	Motivations . . . . .	47
4.1.2	Self-supervised learning . . . . .	48
4.1.3	Proposed self-supervised approach . . . . .	48
4.2	Simulation environment . . . . .	50
4.2.1	Physics simulation . . . . .	51
4.2.1.1	Training objects selection . . . . .	52
4.2.1.2	Bulk generation . . . . .	53
4.2.2	Image generation . . . . .	54
4.2.3	Grasp simulation . . . . .	55
4.2.3.1	Parallel-jaw gripper . . . . .	56
4.2.3.2	Vacuum gripper . . . . .	57
4.2.4	Interactive environment . . . . .	61
4.3	Object-aware grasp detection network . . . . .	62
4.3.1	Network architecture . . . . .	62
4.3.1.1	U <sup>2</sup> -Net backbone . . . . .	62
4.3.1.2	Cascaded decoders . . . . .	64
4.3.2	Self-supervised grasp selection . . . . .	67
4.3.2.1	Network training . . . . .	68
4.4	Experiments . . . . .	69
4.4.1	Training details . . . . .	70
4.4.1.1	Images used . . . . .	70
4.4.1.2	Optimizer details . . . . .	71
4.4.1.3	Grippers . . . . .	71
4.4.1.4	Evaluation metrics . . . . .	72
4.4.2	Quantitative analysis . . . . .	72
4.4.2.1	Testing grasp selection . . . . .	73

4.4.2.2	Compared methods . . . . .	73
4.4.2.3	Performance comparison . . . . .	75
4.4.2.4	Segmentation performances . . . . .	77
4.4.3	Qualitative analysis . . . . .	79
4.4.4	Parameters influence . . . . .	81
4.4.4.1	Ablation studies . . . . .	81
4.4.4.2	Inference threshold influence . . . . .	83
4.5	Adaptation to new situations . . . . .	84
4.5.1	Modified simulation . . . . .	84
4.5.1.1	Simulation modifications . . . . .	84
4.5.1.2	Fine-tuning procedure . . . . .	87
4.5.1.3	Adaptation results . . . . .	88
4.5.2	Transfer to real-life applications . . . . .	90
4.5.2.1	Testing details . . . . .	90
4.5.2.2	Real applications results . . . . .	92
4.6	Conclusion . . . . .	94
4.6.1	Summary . . . . .	94
4.6.2	Contributions . . . . .	95
5	CONCLUSION . . . . .	97
5.1	Summary . . . . .	97
5.2	Contributions . . . . .	98
5.3	Perspectives . . . . .	98
	BIBLIOGRAPHY . . . . .	101
A	ISOLATED OBJECT GRASPING . . . . .	115
A.1	Background augmentation . . . . .	115
B	END-TO-END BIN-PICKING . . . . .	117
B.1	Rigid collision simulation . . . . .	117
B.2	Graspable Instance Segmentation . . . . .	118



---

## LIST OF FIGURES

---

Figure 1.1	Various examples of scenes encountered in grasp detection scenarios . . . . .	3
Figure 1.2	Examples of annotated data from our Jacquard Dataset . . . . .	5
Figure 2.1	Typical pipelines for robotic grasping applications	10
Figure 2.2	Examples of visual sensors . . . . .	10
Figure 2.3	Different kinds of robots used in the industry	11
Figure 2.4	Examples of robotic grippers . . . . .	11
Figure 2.5	Object pose estimation in a pile of objects . . .	13
Figure 2.6	Grasp locations on 3D models . . . . .	14
Figure 2.7	Rectangle grasp representation . . . . .	16
Figure 2.8	Process for a segmentation-inspired gripper-oriented method . . . . .	18
Figure 2.9	Ambiguous box detection in object piles . . . .	19
Figure 2.10	Examples of datasets used in robotic grasping	20
Figure 3.1	Our dataset generation pipeline . . . . .	24
Figure 3.2	Example of a scene generated in the simulation environment . . . . .	25
Figure 3.3	Comparison of two renderings . . . . .	25
Figure 3.4	All rendered images . . . . .	26
Figure 3.5	VHACD decomposition . . . . .	27
Figure 3.6	Rectangle grasp representation . . . . .	28
Figure 3.7	Grasp simulation . . . . .	28
Figure 3.8	Examples of annotated data from our Jacquard Dataset . . . . .	29
Figure 3.9	Examples of misclassification with the rectangle metrics . . . . .	30
Figure 3.10	Real robot setup . . . . .	33
Figure 3.11	Network predictions on real objects . . . . .	34
Figure 3.12	Error comparison between axis-aligned and oriented boxes . . . . .	35
Figure 3.13	Examples of oriented anchor boxes . . . . .	36
Figure 3.14	Architecture of our network . . . . .	37
Figure 3.15	Grasp prediction made by our network . . . .	42
Figure 3.16	Real robot setup . . . . .	44
Figure 3.17	Two steps grasp detection with object segmentation . . . . .	45
Figure 3.18	Examples of failed grasp detections on bulk . .	46
Figure 4.1	Comparison between supervised and self-supervised approaches . . . . .	49
Figure 4.2	Self-supervised training loop . . . . .	50

Figure 4.3	Interactions between the learning agent and the simulation environment . . . . .	52
Figure 4.4	Examples of our scanned 3D objects . . . . .	53
Figure 4.5	Difference between visual and collision models	54
Figure 4.6	Examples of simulated stacks . . . . .	55
Figure 4.7	All rendered images . . . . .	56
Figure 4.8	Example of a parallel-jaw grasp . . . . .	57
Figure 4.9	Vacuum gripper examples . . . . .	58
Figure 4.10	Process of a suction cup grasp . . . . .	59
Figure 4.11	Examples of uniform disc sampling . . . . .	60
Figure 4.12	Global architecture of our network . . . . .	63
Figure 4.13	Architecture of a RSU block . . . . .	65
Figure 4.14	Comparison of two types of segmentation . . .	67
Figure 4.15	Multiplication of the two outputs of our network	68
Figure 4.16	Example of a pile of textureless objects . . . . .	70
Figure 4.17	Simulated suction cups in context . . . . .	72
Figure 4.18	Detection results of the industrial baseline . .	74
Figure 4.19	Human interface for our click and pick simulation	75
Figure 4.20	IoU and precision-recall curves . . . . .	78
Figure 4.21	Inputs and outputs generated by different architectures . . . . .	80
Figure 4.22	Plot of the success rate versus $T$ . . . . .	84
Figure 4.23	Bottle model used for network adaptation . . .	85
Figure 4.24	Examples of our simple noise post-process effect	86
Figure 4.25	Plot of the success rate during the fine-tuning	88
Figure 4.26	Plot of the success rate during the fine-tuning with a parallel-plate gripper . . . . .	90
Figure 4.27	Real robotic setup for bin-picking . . . . .	91
Figure 4.28	Results of our network on semi-ordered piles .	93
Figure 4.29	Results of our network on heterogeneous piles	94
Figure 4.30	Grasp predictions of our network on piles . . .	95
Figure A.1	All background images used for augmentation	115
Figure B.1	Collision between a grasped object and the container . . . . .	118
Figure B.2	Occlusion tolerant graspable segmentation . .	120

---

## LIST OF TABLES

---

Table 3.1	Summary of the properties of publicly available grasp datasets . . . . .	23
Table 3.2	Accuracy of AlexNet trained on Cornell and Jacquard datasets . . . . .	32
Table 3.3	Grasp detection accuracies of multiple models	41
Table 3.4	Accuracies with multiple anchor sizes . . . . .	43
Table 4.1	Characteristics of the network for different input resolutions . . . . .	71
Table 4.2	Performances of multiple grasp detection methods . . . . .	76
Table 4.3	Ablation studies of our architecture . . . . .	82
Table 4.4	Performance results on real problems . . . . .	92





---

## ACRONYMS AND NOTATIONS

---

- RGB Red Green Blue, an image with three color channels
- RGB-D Red Green Blue Depth, an image with four channels, three for the color components, and one for depth information
- DNN Deep Neural Network, a type of machine learning algorithm based on networks built with a succession of layers
- FCN Fully Convolutional Network, a deep neural network architecture built as a succession of convolution layers, without any fully-connected layer
- $H \times W \times C$  Typical notation for image dimensions: height, width and channels (that can hold various types of information, such as colors, depth or normals)



---

## INTRODUCTION

---

As early as 1804, the development of the Jacquard machine in Lyon made it possible to simplify the textile manufacturing process, increasing the production capacities of the factories. Then, automation developed strongly in all sectors of industry, reducing the drudgery of work, and increasing productivity. Robots were introduced in factories in the second half of the 20<sup>th</sup> century. Initially operated by humans to manipulate hazardous materials in the emerging nuclear industry, they were soon fully autonomous on simple tasks, with fixed paths. Today, they can be found in more and more diversified and complex applications.

This diversification also makes processes more complex: a robot must no longer simply act on its environment, but must also perceive and understand it correctly to interact with it. Most often, this perception is made in the form of visual information, obtained through cameras. The better the analysis and understanding of this visual information is, the more robots can perform increasingly complex tasks. Thus, in an industry where the desire for automation is ever increasing, the design of robots capable of working in random or uncontrolled environments is crucial.

In collaboration with Siléane<sup>1</sup>, a French company specialized in industrial robotics in unknown environments, this work aims at exploring solutions for a crucial part of automation: robotic grasping. Robot grasping is one of the key skills a robotic system has to master to be able to physically interact with the environment. Specifically, our goal is to generalize the training of neural networks able to predict successful grasping locations from visual information. Current approaches to perform this task are either using strong knowledge about the target objects, such as full 3D models, or manually annotated data. However, they are not always available, or practical to gather.

Therefore, we explore the use of simulation to easily collect a large amount of annotated data that can be used to train deep neural networks for robotic grasping. Specifically, we address three main questions:

---

<sup>1</sup> [www.sileane.com](http://www.sileane.com)

- **As collecting a large amount of annotated data in robotics is a long and tedious process, how can we use simulation to easily generate such data?**
- **How to best learn efficient models able to extract successful grasp locations for either isolated or piled objects from visual information, without human supervision?**

### 1.1 PROBLEMS AND OBJECTIVES

For many tasks a robot might have to handle in a factory, object manipulation is required. Now, although it is natural for human beings, accustomed to grasping objects since the very first day of their life, grasping is a skill that robots do not have naturally: they have to be taught, or programmed, how to analyze what they perceive to find how to grasp objects.

Given the importance of this subject for the industry, it has received increasing interest from research communities, particularly in computer vision. Due to the many different situations that can be encountered when working on grasp detection, as illustrated by [Figure 1.1](#), data-oriented approaches quickly became the preferred way. Deep learning [63] has shown in recent years its capacity to achieve human-level performances on many computer vision tasks and is, therefore, a very common approach for grasp detection. However, to successfully train neural networks for a task, a large amount of data is required.

Unfortunately, gathering such a quantity of annotated data in robotics is not as simple as for other domains. Traditional approaches typically use manually annotated datasets [19], or data gathered using a physical robot [92]. Both these kinds of approaches share the same issue: they are very time-consuming, and can't be reasonably used to generate the quantity of data required by modern deep learning techniques. To overcome this issue of data gathering, the first objective of this work is to develop simulation environments that can be used to quickly and reliably gather data ready to train neural networks.

With a large amount of data available, efficient grasp detection neural networks can be trained. Most of the work for grasp detection is done by adapting object detection methods. Features are extracted from the image, and the network predicts a quality estimation at different positions, as well as some parameters for the final gripper position [133], similarly to what is done to detect objects in images. Our second objective is to develop new deep learning approaches outperforming existing baselines by leveraging the specificity of the grasping problem.



Figure 1.1: Various examples of scenes encountered in grasp detection scenarios. Top images are isolated objects, bottom ones are piles of objects, homogeneous or heterogeneous, typically found in bin-picking applications.

## 1.2 CONTRIBUTIONS

Our work is dedicated to solve the two problems discussed above in two different situations: grasping an isolated object, and grasping in a pile of objects, *i.e.* in a highly cluttered environment. The two situations have their own specifics and have to be handled differently. In both cases, simulation is used for data generation, and specificities of grasp detection problem are leveraged to create more efficient neural networks.

### 1.2.1 Grasping isolated objects

The first part of our work focus on grasping isolated objects, *i.e.* predicting grasps from an image containing only one object.

- our first contribution for the isolated object problem lies in the public release<sup>2</sup> of the Jacquard dataset [23], similar in nature to the widely used Cornell dataset [19], but with orders of magnitude more data. To ensure a large diversity in shapes, our Jacquard dataset was built using the 3D models of the publicly available ShapeNet dataset [16]. Once the object is loaded in a simulation environment, multiple grasps with a parallel-jaw gripper are physically simulated, trying to lift the object and move it away. Using the outcome of this simulation, a list of all grasping possibilities on the object is created. In parallel, an image of the scene including the object is rendered using a modern ray-tracing engine, resulting in a high degree of realism. Combining the outcome of the gripper simulation and the rendered images results in annotated data that can be used to train neural networks to predict new grasp locations on unseen objects. Overall, the Jacquard dataset contains 54k images of more than 11k different objects. All these images are annotated with multiple successful parallel-plate gripper locations (1.1M unique locations in the whole dataset), as can be seen on Figure 1.2.

Moreover, in addition to the dataset, a web interface is available for researchers to submit their predictions on images of the dataset. These submissions are then tested using the same simulator used for training, resulting in a success or failure outcome. This simulation interface can be used as a common baseline to test the accuracy of different models on the dataset, without having the exact same robotic system and objects in each laboratory working on this problem.

- our second contribution is a new network architecture improving the grasp detection rate for isolated objects. Contrary to state-of-the-art approaches [133] [18] that, inspired by object detection methods, predict separately a score and a deformation for a reference grasp, our proposed method used the specificity of the grasping problem and explicitly uses the correlation between the predicted deformation and score. We also performed experiments to compare our architecture to state-of-the-art baselines. Our network, using the correlation between the score and the grasp prediction, performed better on the Jacquard dataset, while maintaining similar performance than state-of-the-art approaches on the Cornell dataset.

---

<sup>2</sup> The dataset is available for researchers at <https://jacquard.liris.cnrs.fr/>



Figure 1.2: Examples of annotated synthetic images from our Jacquard Dataset. Rectangles represent successful parallel-jaw gripper locations.

### 1.2.2 Grasping in piles

The second part of our work focuses on detecting grasp opportunities in piles of objects. Contrary to the isolated object case, this is more challenging, because of occlusions in the image and interactions between the objects during the grasping process. Our main contribution is a new object-aware grasp detection network trained in a self-supervised way using an interactive simulation environment.

The environment can handle the simulation of multiple piles of many objects in real-time, as well as multiple grasping devices (parallel-jaw gripper and suction cups), and their interactions with the objects. This environment is also designed to be interactive, with a two-way communication channel. In one way, a training module can connect and receive images from the simulation, and in the other way, the environment can receive grasp proposals and execute them.

Interacting with this simulation environment, we train our new object-aware network in a self-supervised way: the network predicts grasps that are then tested in simulation, and learns from the response



of the simulator. Using a self-supervised approach allows the network to learn from relevant examples, without having to pre-generate some annotated data using a heuristic or a third-party method.

The network of our architecture follows the U<sup>2</sup>-Net backbone [95], with two cascaded decoders. The first one predicting a newly introduced graspable segmentation, *i.e.* a segmentation of the image in which only the graspable instances of the objects are separated from the other and the background. The second decoder is responsible for the actual grasp prediction and takes as input the concatenated features of both the encoder and the segmentation decoder. The grasp prediction decoder can thus benefit from the information inferred by its segmentation counterpart.

We extensively tested our object-aware architecture using both simulated and real-case images. Results showed that our network was able to learn a useful representation to detect successful gripper locations in highly cluttered scenes, found in typical applications like bin-picking.

### 1.3 PUBLICATIONS

Three papers have been published in international conferences based on the work done during this thesis. Due to direct industrial applications by Siléane for the bin-picking part, only work done on isolated object grasping has been published.

- [23] Amaury Depierre, Emmanuel Dellandréa, and Liming Chen. "Jacquard: A large scale dataset for robotic grasp detection." Published in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018)*.
- [90] Maxime Petit, Amaury Depierre, Xiaofang Wang, Emmanuel Dellandréa and Liming Chen. "Developmental Bayesian Optimization of Black-Box with Visual Similarity-Based Transfer Learning." Published in *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*.
- [24] Amaury Depierre, Emmanuel Dellandréa, and Liming Chen. "Optimizing Correlated Graspability Score and Grasp Regression for Better Grasp Prediction." Accepted at *2021 IEEE/RSJ International Conference on Robotics and Automation (ICRA 2021)*.

[23] presents our synthetic Jacquard dataset, and shows that it is suitable to train deep learning models for grasp detection tasks.

[90] uses the simulation environment developed during this thesis to automatically optimize parameters from a black-box algorithm through developmental bayesian optimization.

[24] presents our proposed architecture explicitly correlating grasp parameters generation and grasp quality estimation for isolated object grasp detection.

#### 1.4 CONTENTS

This work is organized as follows:

In [Chapter 2](#), we review the state-of-the-art on the robotic grasping problem, with a focus on data-based approaches, *i.e.* methods using a machine learning algorithm.

In [Chapter 3](#), we present our work on grasp detection for isolated objects. Specifically, we describe how we built the synthetic Jacquard dataset. Then, we detail our proposed neural network architecture explicitly correlating grasp parameters generation and grasp quality estimation, and show its performances compared to other approaches, on both the Jacquard dataset and real data.

In [Chapter 4](#), we lay out our contributions on grasp detection in a bin-picking context, *i.e.* with many instances of objects and a high rate of occlusion between them. In detail, we describe our architecture combining global information, with graspable instance segmentation, and local information with grasp quality estimation. We also present the way we train this architecture in a self-supervised way. We conduct experiments and ablation studies to demonstrate the benefits of this approach compared to others and show that, despite being trained only on simulated data, such a model can be used in real-life applications.

In [Chapter 5](#), we finally summarize our work and contributions and give some directions for possible future improvements of this work.



# 2

---

## STATE-OF-THE-ART

---

In this chapter, we review the state-of-the-art of vision-based machine learning methods for robotic grasping. The rest of this chapter is organized as follows. [Section 2.1](#) describes the object grasping problem and categorizes the methods used to solve this problem. [Section 2.2](#) presents the model-based methods, while [Section 2.3](#) details the model-free approaches. [Section 2.4](#) then shows the datasets used in grasp detection and [Section 2.5](#) concludes the chapter.

### 2.1 ROBOTIC GRASPING OVERVIEW

In this section, we review the different approaches for object grasping and categorizes them more precisely in subcategories. Object grasping methods are generally divided into analytic [\[106\]](#) and data-driven methods [\[9\]](#). In analytic approaches, the object shape is processed to find an adequate grasp location for the gripper [\[77\]](#). Data-driven approaches, on the other hand, are based on machine learning algorithms. They have become more and more popular, mainly due to three factors: data availability, better computational power, and significant progress in machine and deep learning algorithms [\[63\]](#). The following of this chapter will be focused on data-driven approaches, as they are currently the most studied and efficient methods for object grasping. For extensive surveys on analytical methods, readers can refer to [\[112\]](#), [\[7\]](#), or [\[106\]](#).

Data-driven approaches can then be separated as model-based or model-free, depending on whether or not they are using knowledge of a 3D model of the target object. Model-based methods typically start with a search algorithm to find the location of an object, before trying to predict grasp positions for this specific object. This allows precise object handling and positioning. On the other hand, model-free methods directly predict grasp locations based on the gripper abilities and are generally better at generalizing to new unseen objects. As can be seen on [Figure 2.1](#), these two approaches follow the same global three steps process:

1. **Data acquisition:** one or multiple sensors create a visual representation of the scene. Sensors are typically cameras (RGB or

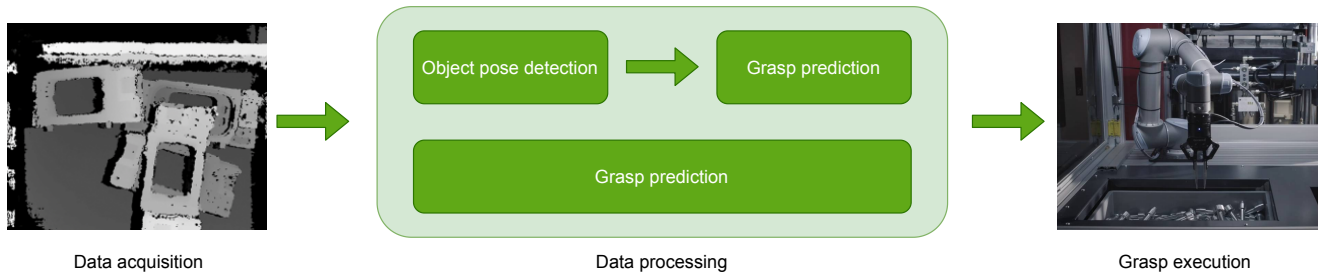


Figure 2.1: Typical pipeline for robotic grasping applications with data-driven methods. Model-based approaches (top row) typically first determine object poses in space before finding suitable grasp locations on them. Model-free approaches (bottom row) directly infers grasp location from visual information.

multispectral), 3D sensors, or a combination of them. [Figure 2.2](#) shows some examples of such sensors.



Figure 2.2: Example of visual sensors. (a) RGB camera, (b) Multispectral camera, (c) 3D sensor, (d) Stereo-vision camera, (e) Multi-camera setup, combining a RGB camera and a depth sensor

2. **Data processing:** one or more algorithms are used to process the scene representation from the data acquisition module. The result is a grasp candidate that can be forwarded to the third component. This step is the one this work studies.
3. **Grasp execution:** the grasp candidate extracted from the previous step is sent to a robotic system (examples can be seen in [Figure 2.3](#)) which is responsible for executing it. This involves path planning to the location and actual grasp execution, which can be performed with various kinds of grippers, as illustrated by [Figure 2.4](#). However, while some research is done on soft grippers [14] [113], or multi-fingers grippers to get closer to how human grasp objects [104] [30], most of the works in grasp detection are done for the two most used kinds: the parallel-plate gripper and the vacuum-based suction cups. This work focuses on these two types of grippers exclusively.



(a) Poly-articulated robotic arm, Fanuc



(b) Delta robot, Omron



(c) Scara robot, Epson

Figure 2.3: Different kinds of robots used in the industry. Each of them has specificities and has been developed to be more adapted to one task than the others. (a) Poly-articulated robotic arm, the most common industrial robot, (b) Delta robot, composed of parallel links connected to a common base, they can perform tasks at very high speeds (c) Scara robot, useful for tasks requiring precise lateral movements in the X-Y plane



(a) Parallel-jaw gripper, Robotiq



(b) Vacuum suction cup, Robotiq



(c) Gecko gripper, OnRobot



(d) mGrip, Soft Robotics



(e) Universal gripper, University of Chicago

Figure 2.4: Examples of robotic grippers of various kind. (a) Parallel-jaw gripper, (b) Vacuum-based suction cup gripper, (c) Adhesive Gecko gripper, (d) Soft multi-fingers gripper, (e) Universal coffee grains based gripper

This work mainly focuses on the second step, data processing. However, due to the large diversity in the first and third steps (respectively data acquisition and grasp execution), a grasp prediction algorithm can not be taken individually and always has to be seen as part of a specific pipeline, *i.e.* one has to consider the abilities of the robotic system and its end-effector when designing a grasp detection algorithm. Keeping that in mind, robotic grasping approaches can furthermore be categorized using many criteria depending on the type of data they process, how they process them, and how they interact with the robotic system that actually performs the grasp. Some of these criteria are:

- the type of objects the method can work on. Some algorithms can only be used on known rigid objects [123], while others can generalize to unknown shapes, or objects with flexible or deformable parts [98].

- the configuration of the scene. While some methods are trained to find the best suitable position to grasp an isolated object [64], some others are focused on extracting one object from a pile, with cluttered background [115] (this task is also called bin-picking).
- data used as inputs. Due to the diversity in sensors, different approaches have been developed using RGB images [55], depth images [73], RGB-D images [132] and point clouds [26] [96].
- the class of machine learning algorithm. Training can be performed with supervised learning [74] (with either manually defined ground truth or self-generated labels), reinforcement learning [49], or unsupervised learning [82].
- the way the predicted grasps are proposed. With discriminative [75], grasp candidates are sampled and then ranked using a classification algorithm (typically a neural network). With generative methods [98], suitable grasp locations are directly generated in an end-to-end trained way.
- gripper degrees of freedom. In a scenario where an object must be picked from a plane, a 2D planar grasp [98] detection system is sufficient. With more complicated scenes or grippers, 6 DoF [58] grasp coordinates might be necessary.
- the frequency of interactions between the robot and the grasp detection method. In a typical open-loop system [98], there is no feedback between the robotic system and the grasp detection algorithm. However, in a closed-loop system, the robot position is continuously updated depending on the visual information [66]. This kind of control is also called hand-eye coordination.

## 2.2 MODEL-BASED GRASPING

In a model-based robotic grasping method, grasp detection is generally a two-steps process. First, one or multiple poses of objects are estimated. Then, once the location and orientation of the object are known, a grasp location is generated and sent to the robotic system responsible for executing the grasp. [Section 2.2.1](#) and [Section 2.2.2](#) will describe these first and second steps respectively.

### 2.2.1 *Object pose detection*

Object pose detection methods aim at estimating the full pose (spatial location and orientation) of one or multiple rigid objects. This pose is generally expressed relative to a reference frame, typically either the camera or the robot. Such a task is not an easy one to master



for a learning algorithm due to many factors like noise in the input data, occlusions between objects, or variations in lighting conditions, as illustrated by Figure 2.5. Moreover, symmetries in object shape can lead to similar, if not identical, visual representation for different poses. Taking symmetries into account when estimating the pose is a whole research interest field [43] [45] [12] [13] and allow a more precise and robust grasp prediction.

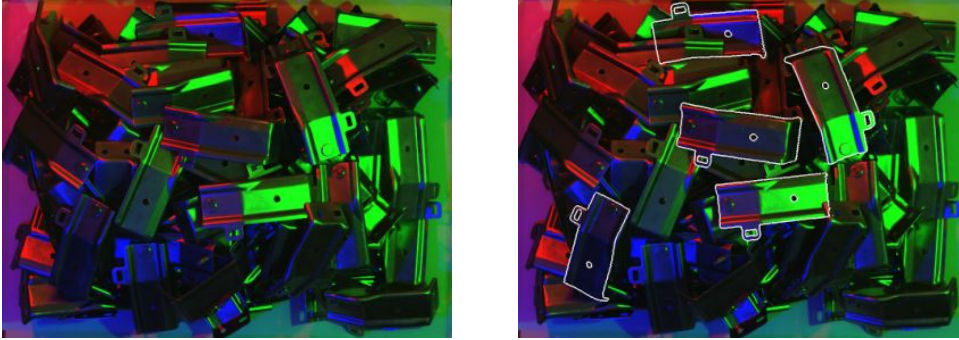


Figure 2.5: Object pose estimation in a pile of objects. This task is not easy due to occlusions and possible difficult lighting conditions. Left is the input image, right is the result of the detection with [102]

As shown in [27], methods using object knowledge for grasp prediction very often require a lot of manual tuning and configuration to achieve good performance, which is a limiting factor for scalability and generalization to new objects. For example, early template matching strategies [41] [42] require either a huge number of templates, or an approximation of the shape with geometric primitives [93], [86] to be able to locate an object in an arbitrary position. Therefore, most of the recent research on model-based object detection is focused on automatic systems with minimal to no manual input required and good generalization abilities.

The success of deep neural networks (DNN) for image processing problems inspired research in object pose estimation and most of the recent works used them. They can either be used as classification tools with a pose space discretization [57] [118] or as direct pose estimator with regression [120] [26] [58]. [123] is a good example of a neural network used for pose estimation: Tremblay et al. designed DOPE (Deep Object Pose Estimation) to predict the 2D coordinates of the corners of the object's bounding box in an image. The final object 6D pose is then obtained using EPnP [65].

Instead of working on images, some works directly use point clouds as inputs. In [26], the authors use PointNet++ [94] to estimate per point pose estimation. Points are then regrouped into clusters and the average pose is kept as the final prediction. With OP-Net [58], Kleeberger and Huber proposed a much faster and more precise way of predicting pose in highly cluttered scenes. Inspired by object detection methods in 2D [100], they discretize the 3D search volume in



small elements, and predict, for each cell, one pose and one confidence score.

### 2.2.2 Grasp generation from object pose

Once the object position is determined, one still has to predict a correct location on this object for the gripper to grasp it, as illustrated by [Figure 2.6](#). Even if this task is a less studied one than pose estimation, it is nonetheless a crucial part of the final grasping pipeline. As the 3D model of the object is known in this case, a traditional approach is to (1) predetermine many feasible grasps on the object using either a human expert or an analytic method, and (2) sort all these grasps in the context of the object pose using heuristics to find the best accessible gripper location [\[116\]](#). Further work proposed to replace the heuristics in the second step with deep learning, using a neural network to perform the search faster [\[117\]](#).

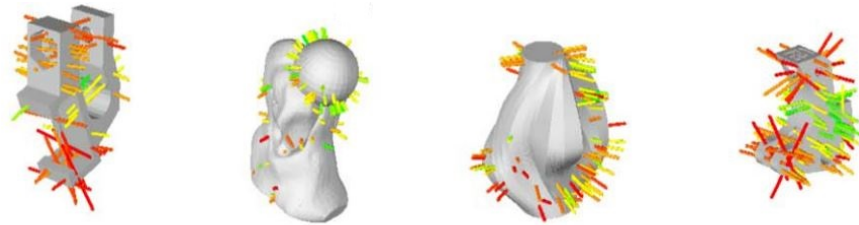


Figure 2.6: Grasp locations on 3D models for a parallel-jaw gripper. One of these grasps will be selected based on its accessibility given the detected pose of the object, and send to the robot for execution.

Regardless of the kind of approach, model-based methods always need a 3D model of the object and therefore require efforts to generalize to new shapes. As industry tends to automatize more and more tasks with robots, including tasks for which no 3D models are available (because each object has a unique shape like in waste sorting, or because objects are deformable like in traditional food industry applications). Model-free solutions have been developed to deal with this issue.

## 2.3 MODEL-FREE GRASPING

In a model-free approach, no prior knowledge about the objects is used. There is therefore no localization step to estimate the pose before a grasp is predicted. They are generally more able to generalize to new unseen objects and shapes [\[110\]](#), and trained end-to-end. For these reasons, they are a very active field of research in the robotic grasping community. [Section 2.3.1](#) and [Section 2.3.2](#) will deal with discriminative and generative methods respectively, while [Section 2.3.3](#) will re-

view the works using a reinforcement learning approach. [Section 2.3.4](#) will end this section by presenting works done about model-free grasp detection in piles of objects.

### 2.3.1 Discriminative methods

In discriminative methods, a set of grasp candidates is generated (for example with a cross-entropy method [105]) and the role of the learning algorithm is to sort them by grasp success likelihood. The robotic system can then choose one of the highest-ranked candidates and execute it. These methods are usually slower as they require many forward passes to evaluate each candidate. However, they have the advantage of being able to evaluate an arbitrary number of grasp candidates and are not limited by the necessity of a discretization of the grasping prediction space. An additional learning step can also be applied afterward to improve the grasp [82].

In [66], Levine et al. trained a CNN to predict the success of a proposed grasp from a RGB image. This CNN was then able to directly control the robot using hand-eye coordination to lead its gripper to a good location. While this proved the feasibility of this method, the 800,000 data samples they used were collected during 2 months on 14 robots, and thus the approach was not easily generalizable to other physical setups. Any significant modification in the camera placement or the robot model would require new data to be collected.

With Dex-Net [72] [73], a Grasp Quality CNN is also trained to predict the success of a proposed grasp. However, the grasp is not presented as a location in the image but is rather encoded directly in the image itself. The image is thus a local patch of the scene, centered and aligned around the proposed gripper location. The network was trained on synthetic depth images annotated using analytic physics metrics and showed good abilities to generalize to new unseen objects. Mahler et al. then extended their approach to other gripper types, like vacuum gripper [74], and even multi-gripper systems [75].

To avoid the expensive sampling and ranking of grasp proposals, fully convolutional networks can also be used [108]. Leveraging the parallelization abilities of GPUs, they can estimate the quality of thousands of grasps simultaneously.

### 2.3.2 Generative methods

Generative methods aim at directly predict a grasp configuration given an input image. They are usually associated with parallel-jaw grippers, as they require additional parameters prediction (like orientation or opening) compared to simpler gripper types like suction cups. Predictions, therefore, take the form of an oriented rectangle [52] in the image frame as can be seen on [Figure 2.7](#). Just like in traditional object

detection [100] [32], the problem can be summed up to a rectangle detection in the image. However, in the case of grasp detection, not only the position and size of the rectangle have to be predicted, but also its orientation.

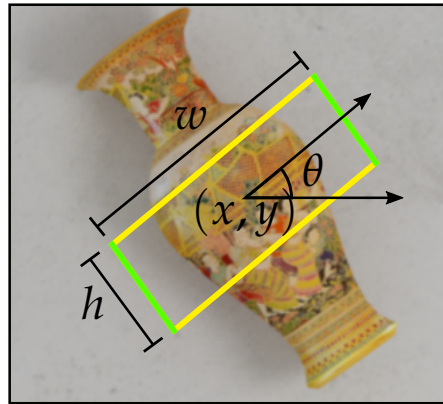


Figure 2.7: Example of a rectangle grasp representation for a parallel-jaw gripper. Green sides are the jaws while yellow sides represent the opening of the gripper.

For the problem of grasping an isolated object on a plane, Redmon and Angelova developed a neural network predicting an oriented rectangle given an image (RGB or RGD, where the blue channel of the image is replaced by depth information) [98]. Their network is also able to predict multiple grasps, one in each region of the image, with an associated score. This idea has then been adapted for object detection, leading to the YOLO (You Only Look Once) approaches [100] [99]. Object detection with two-stage systems like Faster-RCNN [101], also have counterparts for robotic grasp detection. In [64], the first stage is responsible for output candidates, while the second stage ranks them by grasp success likelihood. This illustrates how the two problems are closely linked, and ideas developed for one can often benefit the other.

Increasing depth and complexity of neural networks can increase performance [62] [3], but also computing time and required resources. Keeping the same network backbone based on a ResNet architecture [40], some works have been focused on using a better data representation to improve the accuracy in grasp prediction. In [38] and [18], the authors proposed to use reference grasps, also called anchors. The network is then trained to predict deformations with regards to these anchors, instead of directly predict the grasp parameters. Zhou et al. in [133] used only one size anchor box with multiple orientations instead of multiple axis-aligned grasps with different sizes. Using these anchors helps the network to learn, as it gives it prior knowledge of the expected size of the final grasp detections.

It is also possible to achieve good grasping performance using smaller fully convolutional neural networks. GG-CNN [80] [81] pre-

dicts in real-time both a quality estimation and grasp parameters for each pixel of the input image. As the network is small, predictions can be performed very quickly, allowing real-time grasp evaluation while the grasp is performed, and thus performing the prediction in a closed-loop way.

Jointly learning a secondary task in addition to grasping can also help the network to be more robust. TossingBot [132] is trained to both grasp and throw an arbitrary object at a given location. The algorithm is trained end-to-end using self-supervision and can both grasp an object and then throw it correctly by predicting its release velocity.

### 2.3.3 Reinforcement-Learning methods

Reinforcement learning [119] is not a new field of research in robotics. However, its combination with the progress in deep learning lead to new interesting results using deep reinforcement learning, allowing complex behaviours to be learned from trial and error in many domains, including robotics [59].

Contrary to traditional approaches, reinforcement learning considers a grasp as a sequence of actions. This allows the network to learn direct hand-eye coordination [66] [55] [50], or more high-level actions, like pushing [130] or shifting [5] objects before grasping them to optimize the grasp success likelihood.

In [97], Quillen et al. propose a simulated benchmark for reinforcement learning driven robotic grasping, and evaluate the performances of multiple deep reinforcement methods like DDPG [67] and double Q learning [126].

Using human demonstrations is also a possibility to train deep reinforcement learning algorithms. In [115], the network learns 6 DoF closed-loop grasping policies of novel objects by observing human grasping demonstrations.

When a system learns from trial and error, just like in reinforcement learning, but without using the temporal aspect of sequential actions, the more generic term of self-supervised learning is preferred. Self-supervised learning can be used to directly learn to grasp objects [6], or to collect additional data while performing grasping like depth [35] or pose estimation [114].

### 2.3.4 Model-free grasping in piles

Grasping an object out of a pile of them is a common issue, also referred to as bin-picking. This is a typical use case for which model-based approaches are the most suitable, but not always applicable, due to the lack of explicit models of the objects. Most of the time, algorithms developed for grasping isolated objects can generalize well on scenes with multiple instances [64], or even in small piles [81]

[71]. But new approaches have been developed to deal specifically with very dense piles, with highly cluttered backgrounds. They can be subdivided into two categories: gripper-oriented and object-oriented.

#### 2.3.4.1 *Gripper-oriented methods*

Gripper-oriented model-free methods aim at detecting grasp opportunities with respect to a considered end-effector. While early grasp detection systems relied on analytics heuristics [25], deep learning has also been quickly adopted.

Leveraging the power of convolutional neural networks (CNN), methods were developed for vacuum-based suction cups and parallel-plate grippers. Similarly to what is done in semantic segmentation [4] [17], pixel-wise affordance maps are inferred [2] [84] [131], as illustrated by Figure 2.8. These affordance maps can later be sampled to get a grasp location that is estimated to be successful by the network. Such methods, using CNNs, are very fast. They indeed use the full power of modern GPU architectures to predict many grasps simultaneously, which is much faster than performing multiple forward passes in one network. For these reasons, these methods are nowadays very common for grasp detection in model-free bin-picking contexts.

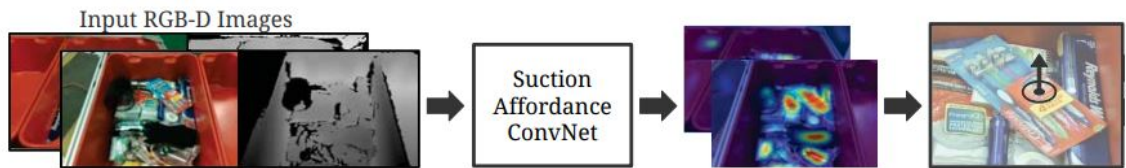


Figure 2.8: Typical process of a segmentation-inspired gripper-oriented method [131]. The input data, represented as images with multiple channels are sent through an encoder-decoder convolutional architecture, outputting an image with multiple probability channels. The value of each pixel of each output channel represents the likelihood of a given grasp to succeed at this location.

#### 2.3.4.2 *Object-oriented methods*

Object-oriented methods aim at mimicking model-based ones without the need for an actual object model. The generic idea is the same as with a model: estimating instance positions in the image and predicting grasps for one or more of these instances.

Just like a network can find cars or planes in images without explicitly knowing what they look like, it is possible to use a generic object detection architecture like Faster-RCNN [101] or YOLO [100] to estimate the position of objects in cluttered scenes [127]. However, for densely populated scenes, which are very common when dealing with bin-picking issues, a box detection of objects is often not sufficient. Multiple instances can indeed be found inside one detection box,



and thus be ambiguous for the grasp detection step, as illustrated by Figure 2.9.



Figure 2.9: In object piles with many instances, a box detection algorithm is not sufficient to determine the location of objects. Multiple instances can lie inside each of the detected rectangles.

An alternative to rectangle box detection methods is instance segmentation, assigning each pixel to only one single instance. Early work are built on heuristics to progressively merge pixels into objects [1] [124]. Further works take advantage of the advancements in deep learning to propose end-to-end learning-based methods for instance detection at pixel level, either by detecting instance boundaries [128] [37] or by directly segmenting instances [17]. Once the instances are detected, one can select the easiest to grasp, *i.e.* the topmost one, with the less occlusion, and grasp can be inferred. This grasp inference is performed either with simple heuristics like the estimated instance centre of mass [36], or more complex methods using point clouds and gripper models [89].

#### 2.3.4.3 Discussion

Gripper-oriented methods focus more on local information, looking for areas suitable for a specific gripper. On the other hand, object-oriented methods analyze the information to find information about how instances are located in relation to each other. **In this work, we propose to combine the advantages of both gripper and object-oriented model-free methods, by using a deep neural network trained to simultaneously predict instance segmentation and grasp quality estimation.**

## 2.4 DATASETS FOR ROBOTIC GRASPING

Deep learning improved robotic grasping detection performances by a huge margin. However, its main drawback is that it requires a lot

of data at training time. Depending on the studied problem, data can take various shapes, as illustrated by Figure 2.10.

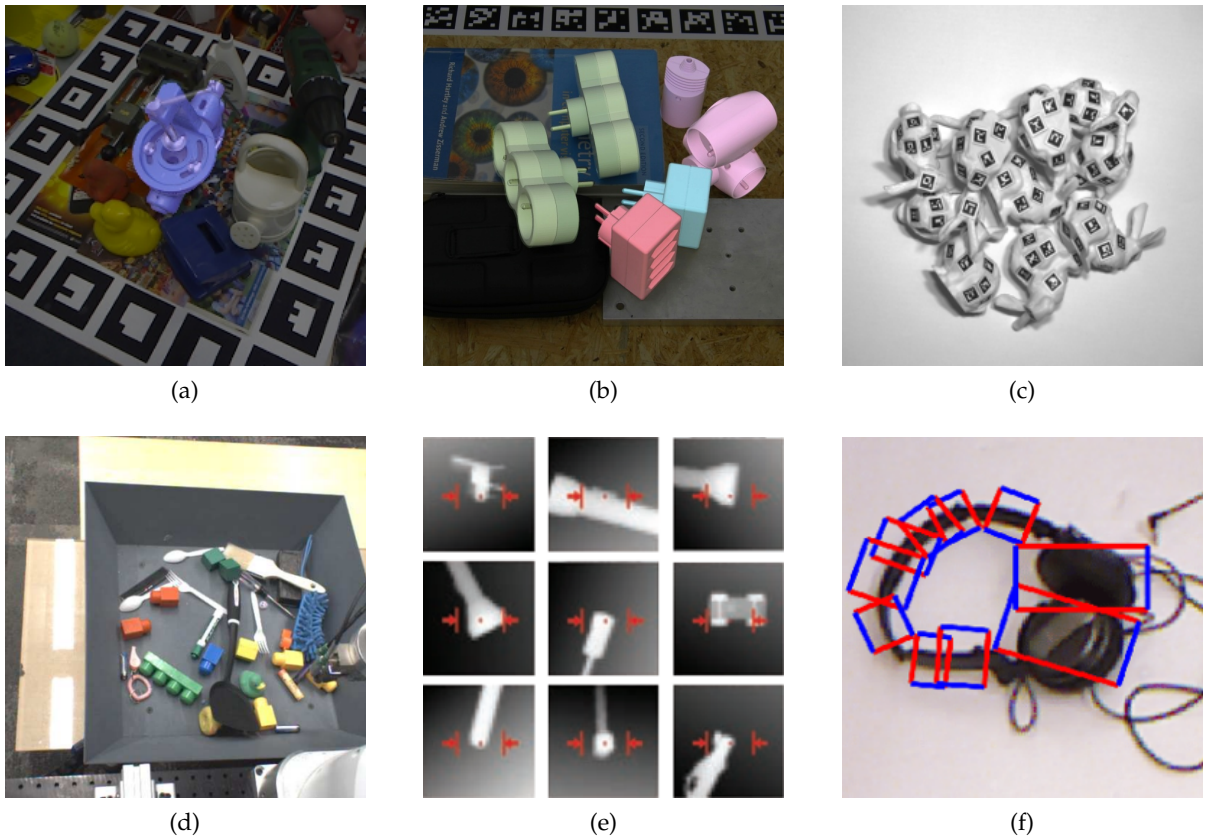


Figure 2.10: Examples of datasets used in robotic grasping for the two main tasks: object detection (top row) and grasp detection (bottom row). (a) manually annotated 6D pose of an object from the LINEMOD dataset [43] (b) multiple manually annotated object poses from the T-LESS dataset [46] (c) automatically generated poses of objects using fiducial markers from [12] (d) RGB image from collected using hand-eye coordination method [66] (e) depth images from Dex-Net [73], the image is centered on the grasp and is labeled as either positive or negative (f) RGB image from the Cornell dataset [19], each image is manually annotated with multiple successful parallel-jaw grasps

The Columbia Grasp Database [33] has been the first attempt to build a large-scale grasp database. Built using an analytic grasp planner, it was composed of more than 230k grasp positions on thousands of objects with multiple hands, and lead the path to a new algorithm for grasp planning from 3D models.

Contrary to other image processing tasks, where large, real-world, annotated datasets are available [22] [28] [68], gathering such a dataset is quite difficult for robotic applications. Generate data on a real-world system is indeed both expensive and time-consuming. Pinto and Gupta used a Baxter robot to collect more than 700h of grasp trials [92]. The

final result was only 50,000 data points, which is a lot for robotic systems, but quite small compared to traditional image datasets used in deep learning. Sharing experience between multiple robots allows to parallelize the data collection process and though speed it up [88]. Using 14 robots and two months, Levine et al. successfully trained a robust grasping algorithm from 800,000 grasp attempts [66], while it took several weeks and 7 robots to get 560,000 grasps in [55]. However, these datasets are bound to the setup they were generated on, and are hard to use for more generic purposes (different gripper, camera to object distance, robot position *etc...*).

It is also possible to generate data without using a robot. For 6D pose estimation, fiducial markers can be used to get the position of an object in the camera frame, like in the T-Less dataset [46]. For grasp annotations, asking humans to annotate real images is also a solution, like in the Cornell dataset [19], which is one of the most used grasping datasets. In both cases, the lack of diversity is a major issue for very deep networks. Cornell contains indeed only 8019 grasp annotations on 885 images of 240 objects. T-Less has almost 40,000 images, but with only 30 different objects. Moreover, the images are highly correlated as many are images from the same scene viewed from multiple viewpoints.

To avoid these issues, simulation is a common alternative to obtain data. Simulation environments like Gazebo [60], MuJoCo [122], Blender [8] or PyBullet [20] can be used to quickly simulate either physics, camera rendering or both. Simulation is not only fast and highly parallelizable but also easily customizable as all parameters of the environment are perfectly controlled. Examples of datasets generated using simulation are Siléane dataset [12] for object pose estimation or Dexnet 6.7 million synthetic grasp images and success annotations.

For parallel-jaw grasp prediction, the most used dataset is the Cornell dataset. With only 885 images of 240 objects, it lacks diversity and is thus not ideal to train very recent neural network architectures. **In this work, we introduce a new large-scale dataset, with more than 50k high-quality RGB-D images with multiple annotated grasps for a parallel-jaw gripper.**

The real world is never as perfect as a simulation is. No matter how carefully the simulation is programmed (gripper model, rendering parameters ...), there is always what is called a *reality gap* between the synthetic and the real data [10]. Reducing this gap is an ongoing research subject, and many techniques have been proposed besides improving the realism of the simulation. Examples of such techniques are domain randomization [121] [48] [49] [83], in which synthetic data distribution is massively expanded in order to include real data distribution, or domain adaptation [31] [11], in which synthetic and real



data distributions are brought together, for example using Generative Adversarial Networks (GANs) [34].

## 2.5 CONCLUSION

Robotic grasping can be learned in many ways, with or without object models, from real or synthetic images. As we want our work to be usable in context where object models are not available, we focus our attention on learning **model-free** grasp detection. We also advocate a massive **use of simulation** to collect larger quantity of data than with human annotation or robot automatic collection. And lastly, we use **self-supervision** to let the learning agent generate relevant data samples, without any human intervention needed.

# 3

---

## ISOLATED OBJECT GRASPING

---

In this chapter, we focus on the grasp detection problem on images of isolated objects. To do so, we present a new method to generate a large-scale dataset in [Section 3.1](#), and an extension of a state-of-the-art grasp detection network in [Section 3.2](#), trained using this previously generated dataset.

### 3.1 GENERATING A LARGE-SCALE DATASET

Data-driven methods for grasp detection are based on learning algorithms that need data to be trained on. Unlike other publicly available datasets built with manual annotations or by gathering robotic trials, we advocate the use of simulation to gather a large number of realistic images annotated with successful grasp locations in a fully-automated way. [Figure 3.1](#) illustrates our proposed generation pipeline. Applying this process to every object in a subset of ShapeNet [\[16\]](#), namely ShapeNetSem [\[109\]](#), we created a large-scale dataset for robotic grasping with a parallel-jaw gripper. This dataset, named the Jacquard dataset [\[23\]](#), has more than 50k images of 11k different objects, with 1 million unique successful grasp positions annotated, and is available for the research community [\[47\]](#). A summary of the properties of publicly available grasp datasets can be found in [Table 3.1](#) for comparison.

Dataset	Number of objects	Modality	Number of images	Multiple gripper sizes	Multiple grasps per image	Grasp location	Number of grasps	Automatized generation
Levine et al. <a href="#">[66]</a>	-	RGB-D	800k	No	No	Yes	800k	No
Mahler et al. <a href="#">[73]</a>	1500	Depth	6.7M	No	No	No	6.7M	Yes
Cornell <a href="#">[19]</a>	240	RGB-D	1035	Yes	Yes	Yes	8019	No
Jacquard (ours) <a href="#">[23]</a>	11k	RGB-D	54k	Yes	Yes	Yes	1.1M	Yes

Table 3.1: Summary of the properties of publicly available grasp datasets

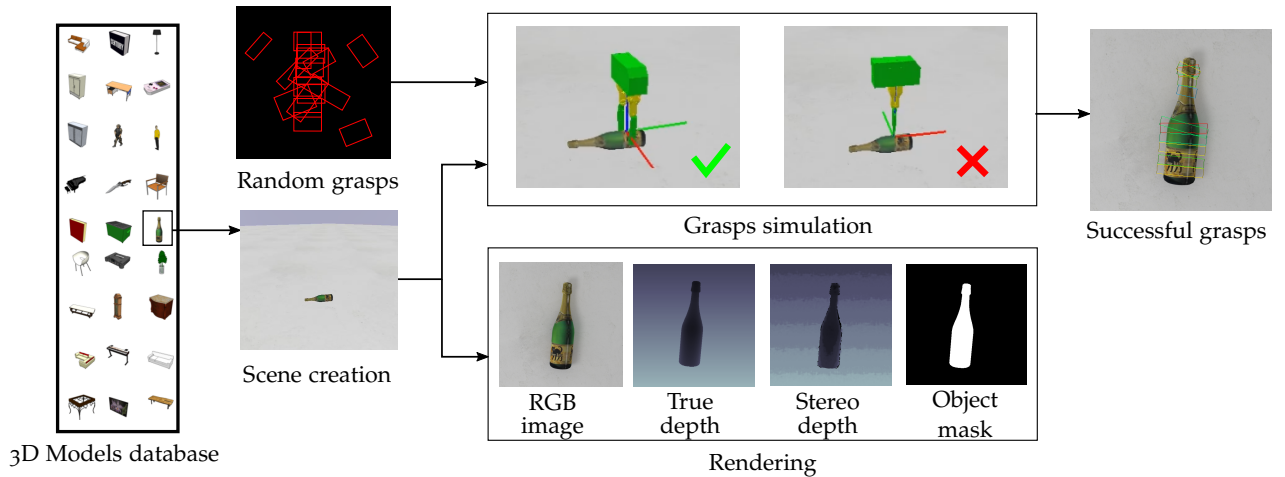


Figure 3.1: Our pipeline to generate annotated images from 3D models. A model from the database is selected and dropped in a stable position. Random grasps are generated from a probability map obtained with a simple heuristic algorithm before being tested in the simulation environment. In the rendering part, a synthetic camera renders the different images.

### 3.1.1 Dataset generation

#### 3.1.1.1 Scene creation

Scenes are all created in the same way. A plane with a white texture is created, the texture being randomly rotated and translated to avoid constant background. Then, an object from a pool of CAD models is selected and loaded into the simulation environment. As the objects might have a wide range of scales, we rescale the model so the longest side of its bounding box has a length between 8 and 90 cm. We also give the object a mass depending on its scale: 80 g for an 8 cm long object and 900 g for a 90 cm one. The mass distribution is independent of the shape of the object. For example, the mass of a hammer is evenly distributed instead of concentrated in the head. Upon loading, the object is dropped from a random position and orientation above the centre part of the plane. Once the object is in a stable position and is not moving or oscillating anymore, the object position and location are saved as a scene configuration. [Figure 3.2](#) represents a view of one scene with an object lying on the plane.

This scene configuration is then sent to two independent modules: one to render the images and one with a physics simulation engine to generate the grasp annotations. For our Jacquard dataset, we created up to five scenes for each object. This number was chosen as a balance between computation time for generating many scenes, and diversity in object views. This can however be changed to adapt to the type of object used. Objects having a lot of symmetries (typically a sphere)



Figure 3.2: Example of a scene generated in the simulation environment

would require fewer views as they only have one possible stable position.

#### 3.1.1.2 *Image rendering*

All the renderings of the scene are done using Blender [8] through its Cycles Renderer. Cycles is a ray-tracing physically-based rendering engine, allowing us to produce more realistic images than the renderer integrated inside the physics simulation environment (see Figure 3.3).



Figure 3.3: Comparison between the integrated Blender Renderer (left) and the ray-tracing Cycles Renderer (right).

A binary mask separating the object and the background, and two depth images are also rendered in addition to the RGB image. The first depth image is a perfect one, which is available as we are in a simulated environment. The second one is a more realistic, noisy one. Instead of adding some Gaussian noise to the depth as in [53], we follow the idea developed in [12]: two additional RGB renderings of the scene are made, and an off-the-shelf stereo-vision algorithm [44] is applied to them to get a depth image with realistic noise. To facilitate the matching between the two images during the stereo-vision process,

we simulate a projected pattern on the object. To do so, a grid mesh with randomly placed holes is generated and placed under a spotlight directed at the object above the scene. The result is a fine speckle all over the rendered scene. Visuals of all images created by the rendering module can be seen on [Figure 3.4](#).

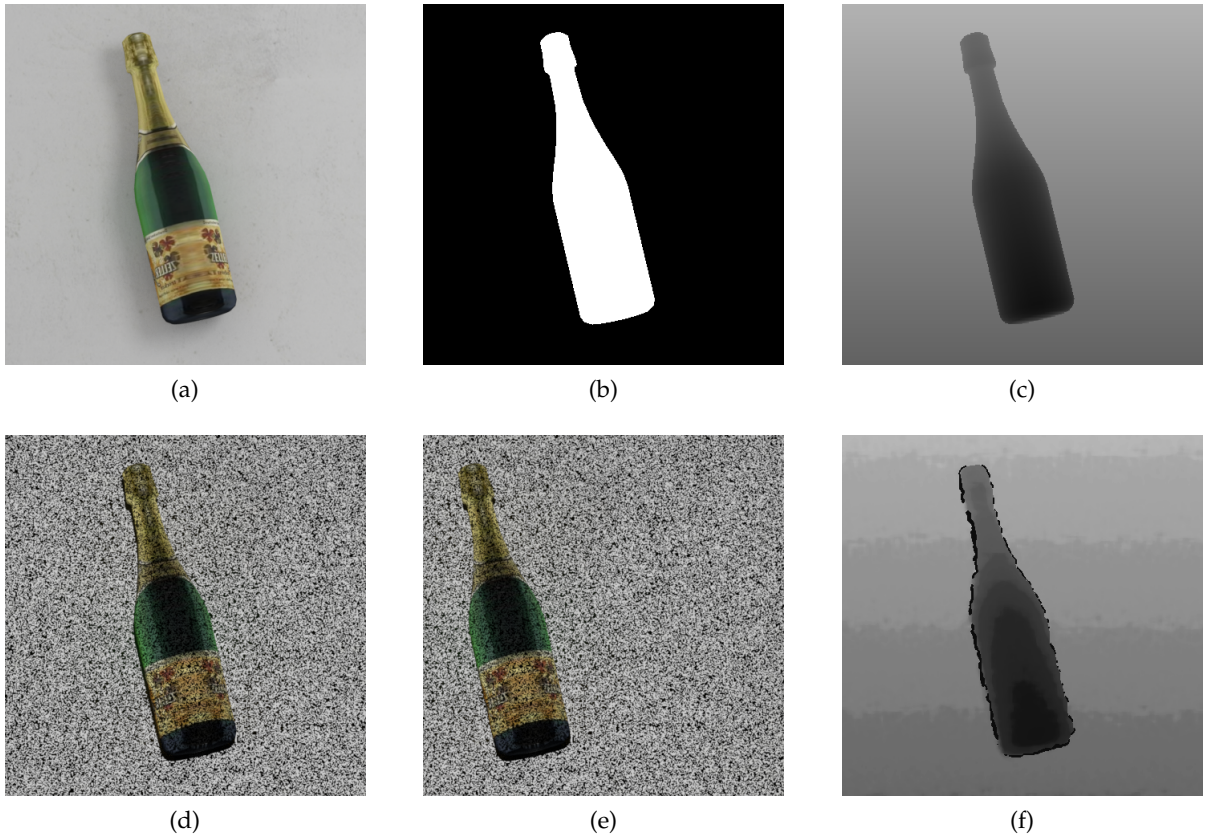


Figure 3.4: Visuals of all the output images of our rendering module. (a) RGB image (b) binary mask of the object (c) perfect depth image (d) left RGB image with the projected grid-pattern for the stereo-vision process (e) right RGB image with the projected grid-pattern for the stereo-vision process (f) noisy depth image obtained using (d) and (e)

### 3.1.1.3 Annotation generation

To generate grasp annotations, the real-time physics library PyBullet [20] is used. Contrary to the rendering module, the object model is not directly sent to the environment. Instead, we split the object into a hierarchical approximate convex decomposition [76] before loading it in the physics environment. This step reduces drastically the number of triangles in the model and therefore speed-up the calculations with a similar collision behaviour. A visual comparison of the model used for the rendering module and the one used for the physics environment can be found on [Figure 3.5](#). As the PyBullet environment is not used

for any image rendering, the texture information is not necessary and is therefore not kept during this simplification process.



Figure 3.5: Visual comparison of an original model (left) and its three parts decomposed VHACD collision model (right).

Different grippers with parallel-jaws are also simulated with two simple collision boxes sliding along an axis. For our Jacquard dataset, we set the max opening to 10 cm and a jaw size comprised in  $\{1, 2, 3, 4, 6\}$  cm. Having multiple jaw sizes for the gripper, combined with varied scales of objects, ensures that a wide range of grasp configurations can be performed during the data generation.

All our candidates and annotations are represented as planar grasps. This means that they are only composed of five values. A grasp example is shown by [Figure 3.6](#), and can be described as:

$$g = \{x, y, h, w, \theta\} \quad (3.1)$$

where  $(x, y)$  is the centre of a rectangle,  $(h, w)$  its size, and  $\theta$  its orientation relative to the horizontal axis of the image. This representation differs from the seven dimensions one described in [52] but *Lenz et al.* show in [64] that it works well in practice. The main advantage of this representation is that the grasp can be simply expressed in the image coordinates' system, without any information about the physical scene: both the  $z$  position of the parallel plates and the approach vector can be inferred from the depth image. Of course, unlike a simulation environment, when the grasp is performed by a real robot,  $h$  and  $w$  are respectively fixed and bounded by the shape of the physical gripper attached to it.

Planar grasp annotations are generated in a three steps process. First, thousands of random grasp candidates are generated, covering the whole area under the camera. All these potential grasp candidates are then tested through rigid body simulation using a gripper with a jaw size of 2 cm. Finally, all the successful positions of the previous step are once again tested with all the gripper sizes. The result is a set of

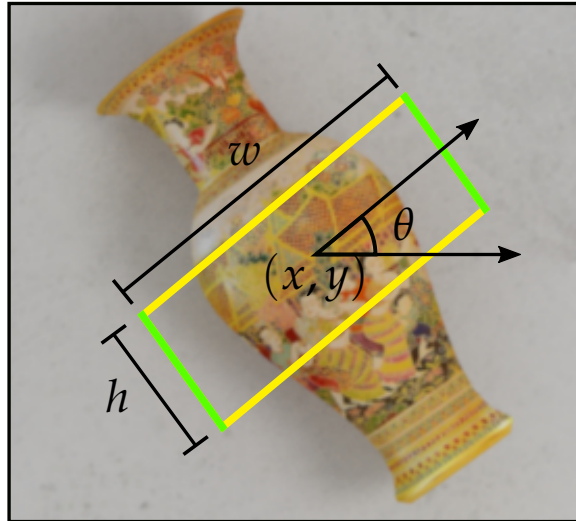


Figure 3.6: Example of a rectangle grasp representation for a parallel-plate gripper. Green sides are the jaws while yellow sides represent the opening of the gripper.

successful grasp locations, each having between 1 and 5 associated jaw sizes. Just like in a real robotic setup, a simulated grasp is considered successful if the object is correctly lifted, moved away, and dropped at a given location by the simulated robot (see Figure 3.7). Once all the random candidates have been tested, a final pass is performed on all the successful grasps to remove the ones which are too close to each other. This last step is necessary to ensure that all grasps are annotated only once as two random candidates could have been generated very close to each other.



Figure 3.7: Simulation of two grasps inside the PyBullet environment. Left is a successful grasp, as the object will be lifted and correctly dropped away, right is a failed one

As the number of possible grasps for one image is very large, a non-uniform probability distribution is used to generate the initial random candidates: more candidates are generated in the most promising parts of the image, and less in the empty areas of the image, where a grasp is less likely to succeed. Theoretically, candidates could be generated with a uniform distribution, but in this case, many grasps



would fall in an empty area, and it would be a waste of time and computation power to test them. For our Jacquard dataset, we used a simple heuristic looking for aligned edges in the image and generating the probability distribution from the density of such edges. However, our experiments showed us that any reasonable heuristic leads to a similar final grasps distribution in the image, sometimes at the cost of more random trials. With this method, the required number of grasp attempts to annotate a scene is reduced by orders of magnitude, while keeping diversity in grasp locations, as illustrated by Figure 3.8. Keeping diversity in the grasp distribution is very important for deep learning-oriented methods.

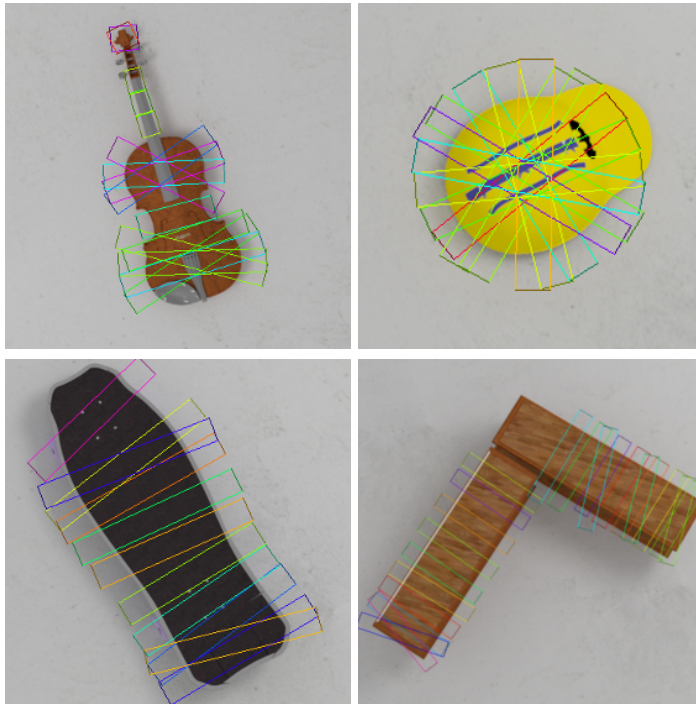


Figure 3.8: Examples of annotated synthetic images from our Jacquard Dataset. Rectangles represent successful parallel-jaw gripper locations. Darker sides indicate the position of the jaws, lighter ones the opening of the gripper.

#### 3.1.1.4 Simulated grasp trial-based criterion

With traditional grasping datasets (like for example the Cornell Dataset [19]), the criterion used to determine whether a prediction is correct or not is a rectangle-based metric. With this criterion, a grasp is considered to be correct if both:

- The angle between the prediction and the ground-truth grasp is smaller than a threshold (a typical value is  $30^\circ$ )
- The intersection over union ratio between the prediction and the ground-truth grasp is over a threshold (typically 25%)



While useful, as it can compare the performance of different methods on a common dataset, this criterion can however produce a lot of visually false-positives, *i.e.*, grasps that, from our human expertise, look bad, but that the rectangle metrics predict as good, as well as false-negatives, *i.e.*, grasps that, from our human expertise, look good, but that the rectangle metrics predict as bad. Such examples can be seen on [Figure 3.9](#).

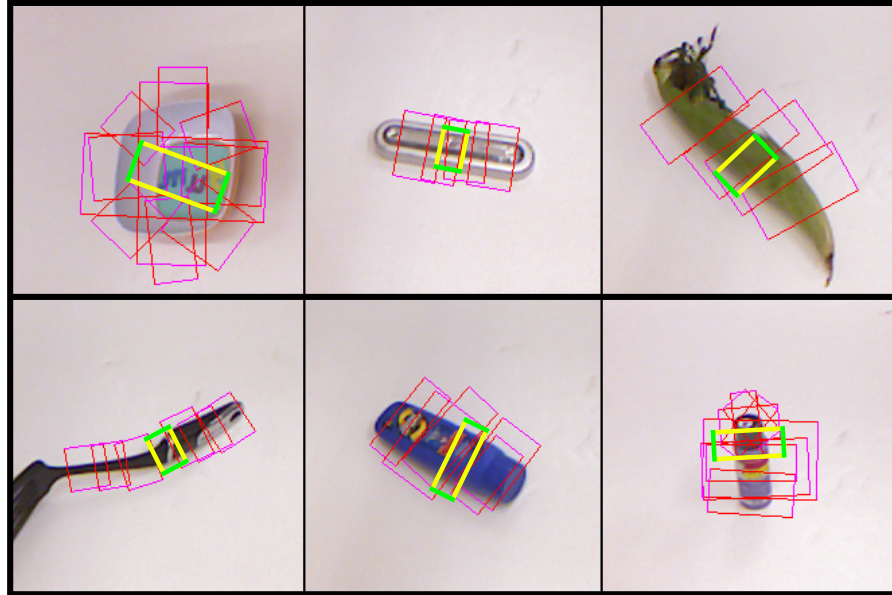


Figure 3.9: Examples of misclassification with the rectangle metrics. Prediction is in yellow and green, ground truth is in red and purple. On the top row, predictions are considered good by the geometric criterion, but will not succeed when performed with a real robot because of collisions. On the bottom row, predictions are considered good, but could reasonably be thought of as good locations for a real parallel-jaws gripper.

With the Jacquard dataset, we introduce a new criterion based on the simulation environment, subsequently called simulated grasp trial-based criterion (SGT). Specifically, when a new grasp should be evaluated as successful or not, the corresponding scene is rebuilt in the simulation environment, and the grasp is performed by the simulated gripper, in the exact same conditions as during the generation of the annotations. If the outcome of the simulated grasp is a success, *i.e.*, the object is successfully lifted and moved away by the simulated robot using the predicted grasp location, the prediction is then considered as a good grasp. This novel SGT criterion is much closer than the rectangle metrics to real-world situations, where a single object can have many successful grasp locations, including some slightly away from the annotated ones. Just like the rectangle metrics, the SGT criterion is independent of the method used to predict grasp locations.

For the purpose of reproducible research, we are releasing along with the Jacquard dataset a web interface [47] allowing researchers to send grasp requests to the simulation environment. Their requests are then processed, and the outcome of the SGT is sent back to them.

### 3.1.2 Dataset validation

In order to evaluate the relevance of our Jacquard dataset, we carried out two series of experiments:

- cross-dataset grasp prediction with the Cornell and Jacquard datasets in [Section 3.1.2.2](#)
- evaluation of grasp predictions using a real robot and gripper in [Section 3.1.2.3](#)

We start this section by describing the grasp prediction algorithm we used for both sets of experiments.

#### 3.1.2.1 Training setup

For all our experiments, we used an off-the-shelf CNN, *i.e.* AlexNet [61]. The network’s convolution weights have been pre-trained on ImageNet [22]. The top classification fully-connected layers are removed and replaced with a new fully-connected layer trained from scratch. This layer output is a vector of size 5, corresponding to one predicted grasp value for the image. To be able to use the AlexNet model with an RGB-D image, we simply normalize the depth image to get values close to color channels ones and duplicate the filters corresponding to the blue channel in the first pre-trained convolution layer. The network is trained through Stochastic Gradient Descent for 100k iterations, with a learning rate of 0.0005, a momentum of 0.9, and a weight decay of 0.001. The learning rate is divided by 10 after the first 75k iterations. To compute the error of the network, the Euclidean distance between the prediction and the closest annotation is used:

$$\mathcal{L} = \min_{g \in \mathcal{G}} \|g - \hat{g}\|^2 \quad (3.2)$$

where  $\mathcal{G}$  is the set of all the annotations for the image and  $\hat{g}$  is the network prediction.

Before training, we perform data augmentation by translating, rotating, and mirroring the images. For synthetic data, we also use the object’s segmentation mask to replace the default background with different textures (cardboard, paper, wood, grass ...) to generate more variabilities. Images used as background can be found in [Appendix A](#).

### 3.1.2.2 Cross-dataset evaluation

This series of experiments aims to show that: 1) our Jacquard grasp dataset, despite being totally synthetic, can be used to train DNNs to predict grasp locations on real images; 2) the diversity of objects and grasp locations is important for a trained CNN to generalize on unseen objects. To do so, two datasets are used to train the network on:

- the Cornell dataset, with its 885 RGB-D images of 240 objects, and 8019 hand-labeled grasp locations
- a subset of our Jacquard dataset, with 15k RGB-D images of 3k randomly selected objects, for a total of 316k different grasp annotations

To highlight 1), the neural network is trained on Jacquard and tested on Cornell; for 2), it is trained on Cornell and tested on Jacquard. For comparison, we also display a baseline performance trained and tested on the same dataset, *i.e.* Cornell or Jacquard. For this purpose, we performed training and testing of the network with 5-fold cross-validation, leading to 5 variants of the network with slightly different accuracies on each dataset. Each variant trained on Cornell (Jacquard, respectively) is then tested on the whole Jacquard (Cornell, respectively) dataset to evidence 1).

Training Dataset	Rectangle Metrics		SGT
	Cornell	Jacquard	Jacquard
Cornell	86.88% $\pm$ 2.57	54.28% $\pm$ 1.22	42.76% $\pm$ 0.91
Jacquard (ours)	81.92% $\pm$ 1.95	74.21% $\pm$ 0.71	72.42% $\pm$ 0.80

Table 3.2: Accuracy of AlexNet trained on Cornell and Jacquard datasets

Table 3.2 summarizes the experimental results evaluated by both rectangle metrics and SGT criterion. As can be seen, when the network is trained on the simulated Jacquard dataset and tested on real images from Cornell, it achieves a grasp prediction accuracy of 81.92%, which is quite close to the baseline performance of 86.88%. Furthermore, it can also be noticed that this network tends to predict grasps that are visually correct on real, hand-labeled images, despite being classified as wrong by the rectangle metrics. Examples of such predictions are shown on the bottom line of Figure 3.9.

In contrast, when AlexNet is trained on Cornell and tested on Jacquard, with a much wider diversity of objects and grasps, it depicts a grasp prediction accuracy of only 54.28%, which records a performance decrease of 20 points in comparison with its baseline performance. As for the other training, part of this gap could be explained

by the misclassification of the rectangle metrics. However, this performance decrease is confirmed by our SGT criterion: the network trained on Cornell only displays a grasp prediction accuracy of 42.76%, which is 30 points behind the 72.42% accuracy of the same CNN trained on Jacquard.

All these figures suggest that Jacquard can indeed be used to train CNNs for effective grasp location prediction. Furthermore, thanks to the diversity of objects and grasp locations, Jacquard enables the trained CNN to be much better for generalization.

### 3.1.2.3 Real-robot testing

How good is a grasp predicted by a DNN trained with our synthetic data, in real? To answer this question of possible reality gap, we mounted a parallel plate gripper on a Fanuc M-20iA robotic arm (see [Figure 3.10](#)). To ensure a wide variability in shapes, material, and scales, we selected 15 everyday objects (both toys and furniture), and 13 industrial components. To determine whether or not a grasp is successful, we used the same criterion as in the simulation environment, but this time using the aforementioned real grasping robot instead of the simulated one: the grasp is considered successful only if the object is lifted, carried away and correctly dropped. For this test, we once again compared the two networks trained on the Cornell dataset and a subset of our Jacquard dataset.

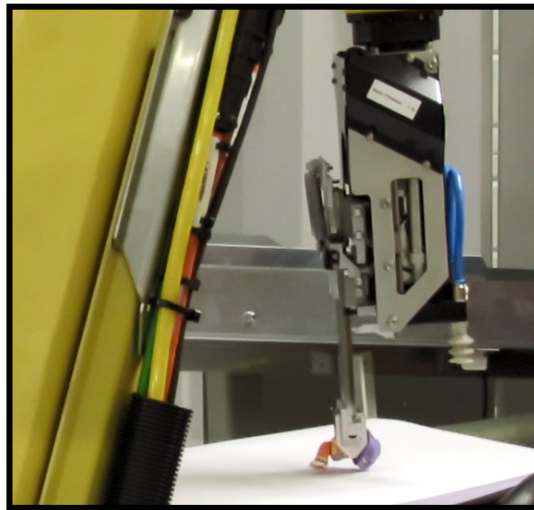


Figure 3.10: Picture of the robot performing a grasp predicted by a trained neural network on one of our real objects. The camera is located above the grasping area, looking downwards.

The experimental results show that the grasp predictor using AlexNet trained on the Jacquard dataset displays a grasp successful rate of 78.43%, which is even 6 points higher than its grasp accuracy inside the simulation environment using the SGT criterion (see [Table 3.2](#)). This generalization skill of the trained grasp predictor can be explained

by the large diversity of shapes and grasp locations in the Jacquard dataset. For most of the failed cases, the proposed grasp was not stable enough: the rectangle in the image was visually good, and the object was successfully lifted but dropped during the movement of the robot. Now with the same network trained on Cornell, the robot succeeded only 60.46% of the time, mostly due to bad rectangle localization in the image. [Figure 3.11](#) shows some examples of the objects for which the network trained on Cornell failed to predict a good grasp, but the one trained on Jacquard succeeded.

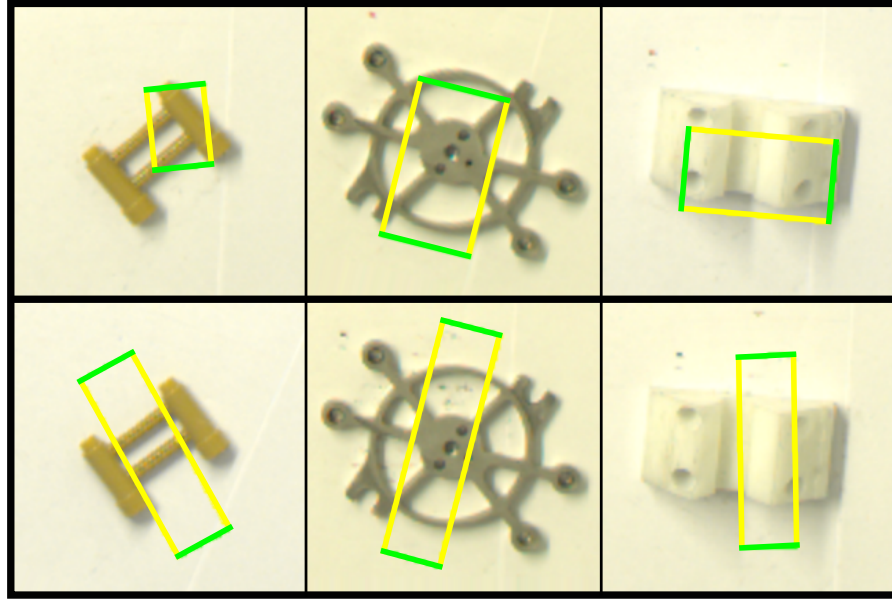


Figure 3.11: Samples of grasps predictions on real-life objects using the network trained on Cornell (top row), and Jacquard (bottom row)

### 3.2 LEARNING TO GRASP OBJECTS

In [Section 2.4](#), we presented a method to automatize the gathering of a large-scale grasping dataset using simulation. State-of-the-art methods for grasp detection in such annotated images of an isolated object rely on neural networks predicting multiple grasp locations, each with an associated score to order them [\[38\]](#) [\[133\]](#) [\[18\]](#). Inspired by object detection methods like [\[100\]](#), this kind of approach assumes that the predicted score is not highly correlated to the predicted location. As shown by [Figure 3.12](#), this is not the case when the prediction is an oriented grasp: a small error on the prediction can lead to a very different outcome when the grasp is performed. To overcome this issue, we present a novel DNN architecture, which directly correlates its grasp quality evaluation with its grasp prediction to improve grasp regression through a newly introduced loss.

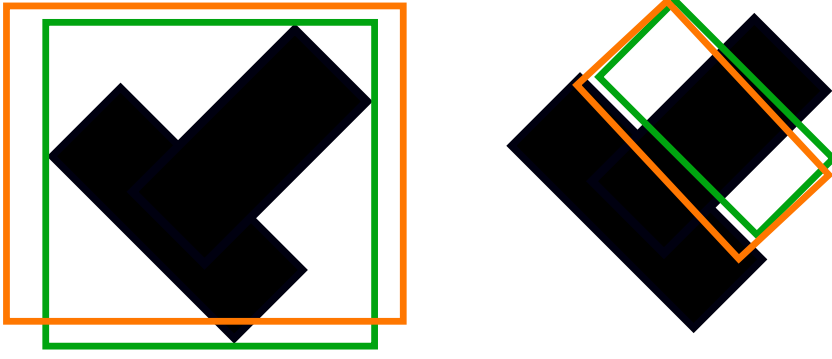


Figure 3.12: Error comparison between axis-aligned (left) and oriented boxes (right). The L2-error between the green and the orange box is the same in both cases. For axis-aligned boxes, the small error does not change the score that has to be predicted: the object is still contained in the box. For oriented boxes, the same small error changes the outcome of the grasp: one of the jaws will collide with another part of the object, preventing the grasp to be successful.

### 3.2.1 Grasp prediction using a scorer

In this section, we present our neural network architecture to predict grasps locations from images of isolated objects. For this purpose, we consider only planar grasps and use the 5D grasp representation defined in Equation 3.1.

#### 3.2.1.1 Anchor boxes

To simplify the regression problem, prior knowledge about the position, size and orientation of the grasp is introduced through oriented reference grasps as proposed in [133]. These reference grasps, also called anchor boxes, are defined as  $g_a = (g_{ax}, g_{ay}, g_{aw}, g_{ah}, g_{a\theta})$ . The grasp is then defined as a deformation  $\delta = (\delta_x, \delta_y, \delta_w, \delta_h, \delta_\theta)$  of a reference grasp according to the following equation:

$$\begin{aligned}
 x &= \delta_x * g_{aw} + g_{ax} \\
 y &= \delta_y * g_{ah} + g_{ay} \\
 w &= \exp(\delta_w) * g_{aw} \\
 h &= \exp(\delta_h) * g_{ah} \\
 \theta &= \delta_\theta * (180/k) + g_{a\theta}
 \end{aligned} \tag{3.3}$$

where  $k$  is the number of different anchor boxes. Figure 3.13 illustrates three different examples of oriented anchor boxes.



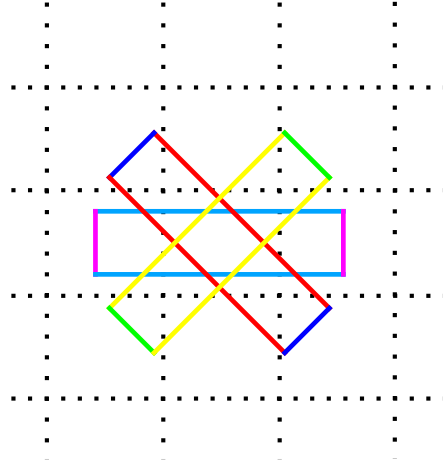


Figure 3.13: Three examples of oriented anchor boxes with the same dimensions and angles of  $-45^\circ$ ,  $0^\circ$  and  $45^\circ$  respectively. They are all centered on the same pixel of the feature map.

### 3.2.1.2 Network architecture

Figure 3.14 presents the global architecture of our network. It contains three main components: a feature extractor (FE), a primary grasp predictor (PGP), and a scorer, a new module compared to state-of-the-art models. Given a feature delivered by FE from an input image, PGP aims to predict grasp parameters along with a primary grasp score indicating the quality of the corresponding grasp. This primary grasp score is predicted independently from the grasp parameters, although the same feature map is used as input for both predictions. As a result, the scorer refines this primary grasp score and delivers a final one for each set of predicted grasp parameters and the visual neighbourhood of the corresponding grasp position.

**FEATURE EXTRACTOR (FE)** For our experiments, we have chosen to use the popular ResNet-50 network [40]. Of course, any other fully convolutional backbone can be used as a feature extractor. The fully convolutional constraint is important: it allows the backbone to be used with any input image size, instead of being restricted to one specific size. For example, instead of the traditional  $224 \times 224$  input size of ResNet, our network takes an image of size  $320 \times 320$  pixels. In our experiments, the output of the FE is the result of the fourth convolution block. Therefore, the output feature maps have a size of  $20 \times 20 \times 1024$ .

**PRIMARY GRASP PREDICTOR (PGP)** The second part of our network is an oriented anchor box-based grasp detector. We used the state-of-the-art architecture presented in [133], with two heads. Specifically, two separate convolutional layers are added to predict separately from an input feature map, for each pixel of this feature map, and for

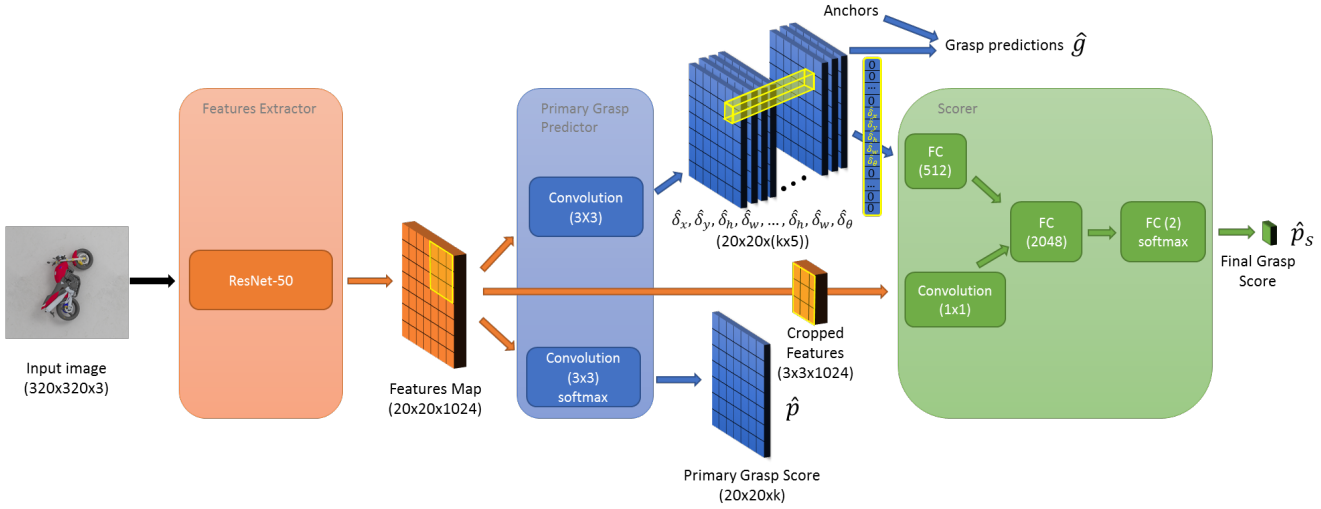


Figure 3.14: Global view of our architecture and its three components: the feature extractor, the intermediate grasp predictor, and the scorer network.

each reference anchor box, five regression values  $\hat{\delta} = (\hat{\delta}_x, \hat{\delta}_y, \hat{\delta}_w, \hat{\delta}_h, \hat{\delta}_\theta)$ , as well as a primary grasp quality score  $\hat{p}$ . This primary grasp quality score shows the quality of the corresponding oriented reference grasp: a score close to 1 means high confidence of a good location, while a score close to 0 indicates an inadequate position or orientation for a parallel plate gripper. It is important to note that this primary grasp prediction network does not have the anchors as input: it only predicts a deformation that is afterward applied to the corresponding anchor.

**SCORER** The primary grasp quality score, predicted by the previous PGP, only depends on the visual features and not on the final predicted grasp parameters  $\hat{g}$ . So a reference grasp on a good location could have a high score despite being a bad prediction, once the final grasp  $\hat{g}$  is computed through Equation 3.3. Moreover, the score estimation cannot be used to improve regression quality. To deal with these issues, we extend this state-of-the-art network by a third component: a scorer network. Inspired by two-stage object detection architectures [101] this scorer is implemented on the top of PGP. However, unlike in object detection, its role is not to refine the predicted grasp parameter by PGP but its grasp quality score. Similarly to the Grasp Quality CNN used in different versions of Dex-Net [73] [74] [108], the scorer network predicts the likelihood  $\hat{p}_s$  that a proposed grasp is a good one, using not only visual information but also the actual predicted grasp parameters.

In Figure 3.14, we can see the detailed implementation of this scorer network. We kept this network small to avoid adding too many computation costs and memory usages. It takes as input the visual neighbourhood of a grasp position along with a set of estimated grasp



parameters. Specifically, a  $3 \times 3$  area from the feature map around the grasp position is sent through a  $1 \times 1$  convolutional layer with 1024 filters. Its output is then flattened to a vector of size 9216 ( $3 \times 3 \times 1024$ ). As for each potential grasp position, there exist  $k$  anchors with the corresponding grasp parameters, we encode the input of grasp parameters for a given grasp as a vector of size  $5 \times k$ , where every value is set to 0 except the 5 values corresponding to the considered anchor box, which are set to the  $\hat{\delta}$  output from PGP. Keeping a  $5 \times k$  dimension vector allows the network to differentiate the  $k$  base anchors, while not having to transform the  $\hat{\delta}$  coordinates with Equation 3.3. This  $5 \times k$  vector is passed through a 512 neuron fully connected layer, and the result is concatenated with the image vector. The result is a 9728 dimensional vector processed by two last fully connected layers, resulting in a graspability score and a non-graspability score. These two scores are then processed by a softmax to get a grasp success likelihood. All the layers of the scorer network (except the last one) are followed by a leaky ReLU [70] with a negative slope set to 0.1.

In short, our whole model with its three components takes one image as input and predicts three outputs:

- a grasp parameter prediction from PGP
- a primary grasp quality score from PGP describing the likelihood that the corresponding reference anchor box is a good grasp
- a final score from Scorer delivering the likelihood that the actual predicted grasp is a good one

### 3.2.1.3 Loss functions

There are three different loss functions in our architecture. Loss functions for the prediction of the primary grasp quality score by PGP and the refined grasp quality score from the scorer are both softmax cross-entropy losses. They are both computed only on a subset of the predictions:  $P$  positive and  $3P$  negative using PGP, and  $T$  grasps for the scorer.

$$\mathcal{L}_{intermediate}(\hat{p}) = -\frac{1}{4P} \sum_{i=1}^{4P} p_i \log(\hat{p}_i) + (1 - p_i) \log(1 - \hat{p}_i) \quad (3.4)$$

$$\mathcal{L}_{scorer}(\hat{\delta}, \hat{p}_s) = -\frac{1}{T} \sum_{i=1}^T p_{si}(\hat{\delta}_i) \log(\hat{p}_{si}) + (1 - p_{si}(\hat{\delta}_i)) \log(1 - \hat{p}_{si}) \quad (3.5)$$

where  $p_i$  (resp.  $p_{si}(\hat{\delta}_i)$ ) is the ground truth associated with the  $i^{\text{th}}$  primary score prediction (resp. scorer score prediction) calculated as explained in [Section 3.2.1.4](#).

The regression loss for the prediction of grasp parameters is composed of two terms: a classic smooth L1 function to ensure the predicted grasps match the annotated ground truth, and a newly introduced second term using the scorer output to guide the gradients in a direction that improves the estimated quality of the predicted grasp.

$$\begin{aligned} \mathcal{L}_{reg}(\hat{\delta}, \hat{p}_s) = & \frac{\alpha}{P} \sum_{i=1}^P \sum_{m \in \{x, y, w, h, \theta\}} L1_{smooth}(\delta_{mi} - \hat{\delta}_{mi}) \\ & - \frac{1}{T} \sum_{i=1}^T \log(\hat{p}_{si}) \end{aligned} \quad (3.6)$$

where  $\delta_i$  are the ground truth values obtained from the annotated grasp inverting [Equation 3.3](#). For all our experiments,  $\alpha$  is set to 2.

As we do not want to update PGP’s weights to predict grasps that are easier to classify for the scorer (*i.e.* reducing  $\mathcal{L}_{scorer}$ ), gradients from  $\mathcal{L}_{scorer}$  are only used to update the scorer and the FE networks. Similarly, gradients from  $\mathcal{L}_{reg}$  are not used to update the scorer network but only PGP and the FE networks.

#### 3.2.1.4 Training procedure

**ANCHOR SELECTION** As all the grasp parameter predictions are generated with respect to reference anchors, choosing them correctly is crucial for the performance of the whole network. In [\[18\]](#), Chu, Xu, and Vela used 3 different scales and aspect ratios for a total of 9 axis-aligned anchors. However, [\[133\]](#) showed that orientation is more important for accuracy than the dimensions of the box. Therefore, we also used for our experiments only one anchor with  $k = 6$  different orientations.

Instead of using a ratio of 1:1 with an arbitrary, manually set size, we compute, before the training process, a mean box of all the ground-truth grasps from the training dataset. This approach gives us values for  $h$  and  $w$  of the anchor boxes and has been proven to yield good results in object detection [\[99\]](#). As grasps usually have a larger  $w$  than  $h$ , using a mean grasp helps as it provides network reference grasps closer to the real ones than when using a 1:1 ratio, while also reducing the number of hyperparameters needing manual optimization.

**GRASP SELECTION** The primary score of PGP is trained using the fast but less accurate Angle Matching strategy presented in [\[133\]](#): the ground truth score  $p$  is set to 1 if the distance between the angles of the corresponding anchor and an annotated ground truth grasp is under a threshold and set to 0 otherwise. The scorer network uses

Jaccard Matching instead: the ground truth score  $p_s$  of a predicted deformation  $\hat{\delta}$  is set to 1 if the corresponding grasp  $\hat{g}$  calculated through Equation 3.3 is both close in orientation and has an intersection over union with a ground-truth grasp over a threshold. As this is more time-consuming than Angle Matching, not all the  $20 \times 20 \times k$  predictions are evaluated during training. Only a subset of all the predictions is used to compute the gradients involving our scorer network. To determine which of the grasps are selected, we use the primary score delivered by PGP only based on the visual input. The  $T$  anchor boxes with the highest scores are selected and only the corresponding  $T$  grasps are evaluated by the scorer network and used for training. At test time, all the grasps are evaluated by the scorer.

### 3.2.2 Experimental evaluation

In this section, we present the experimentations we made using our architecture. Specifically, we define the experimental setup in Section 3.2.2.1, then analyse the results (Section 3.2.2.2), and finally present the performance on a real robotic setup in Section 3.2.2.3.

#### 3.2.2.1 Experimental setup

**DATASETS** To be able to compare our architecture with other models, we used the widely adopted Cornell Grasping Dataset [19]. State-of-the-art models achieve very high accuracy rates on this dataset, and the few unsuccessful detection results are in fact visually consistent with good grasp locations, even if they do not match any ground truth grasp with traditionally used Jaccard Matching. For this reason, we also used the larger Jacquard Dataset presented in Section 3.1.1. Just as in the previous work, we divided the dataset into 5 parts and performed cross-validation. This separation was carried out object-wise, which means that images containing one object are either all in the training dataset or all in the testing one. Presented performances are the averaged accuracy over the five tests.

**TRAINING AND EVALUATION DETAILS** As overfitting is a common issue with neural networks, especially when fully connected layers are deployed, we used online data augmentation to make sure the network does not see twice the exact same image during the whole training process. In detail, the RGB image is randomly rotated around its center, mirrored, shifted up to 50 pixels in both axes, and finally rescaled to  $320 \times 320$  pixels before feeding to the network. This augmentation is not performed at testing time.

To help the training, all the weights of the feature extractor are initialized from a pretraining phase on the large RGB dataset ImageNet [22]. Weights for the other layers are randomly initialized. The network is trained through Stochastic Gradient Descent with a momentum of

Algorithm	Backbone	Dataset	
		Cornell	Jacquard
Depierre et al. [23]	AlexNet	86.88%	74.21%
Chu et al. [18]	ResNet-50	95.5%	-
Zhou et al. [133]	ResNet-50	94.91%	82.13%*
	ResNet-101	96.61%	-
Morrisson et al. [81]	GG-CNN	88%	78%
	GG-CNN2	-	84%
Ours (using primary score)	ResNet-50	95.02%	83.61%
Ours (scorer)	ResNet-50	95.2%	85.74%

\*our implementation

Table 3.3: Grasp detection accuracies of multiple models on Cornell and Jacquard dataset

0.9 for 100k iterations. The batch size is set to 10 for all the models. The learning rate is set to 0.001 and the weight decay to 0.0001. To train the scorer network,  $T$  is set to 64 as we found this was a good trade-off to balance positive and negative examples, as well as being a value small enough to avoid overly increasing training time. To train the scorer, we used Jaccard Matching with thresholds of  $15^\circ$  and 25% for the intersection over union. For evaluation, we used the more common  $30^\circ$  criterion (and 25% for the intersection over union) to be consistent with previous work.

### 3.2.2.2 Results

Once trained with the whole architecture (FE, PGP, and Scorer), our proposed model delivers from an input RGB image 3 outputs: regressed grasp parameters for each potential grasp position by PGP, a primary grasp quality score from PGP, and a refined final one from Scorer for each anchor on that position. To evaluate the influence of our scorer network on performances, we compared the success rate of multiple architectures on both the Cornell and Jacquard datasets. For our model, we evaluated accuracy using both the primary grasp probability from PGP and the final refined one from the Scorer, although they were trained together during the training procedure. The results are presented in Table 3.3. To obtain a baseline of comparison with a similar architecture to ours without the scorer network, we also trained the algorithm proposed in [133] on the Jacquard Dataset.

As the results show, our proposed architecture performs similarly to the state-of-the-art on the Cornell Dataset with 95.2% compared to 95.5% for the network from [18] with the same ResNet-50 backbone, and is only 1.4 points behind the performance of [133] with a much

larger ResNet-101 backbone. On the Jacquard dataset, our architecture achieves state-of-the-art performances with an accuracy of 85.74%, outperforming the previous state-of-the-art performance by 1.74 points. Compared to a baseline at the third row with similar architecture (FE, PGP) but without the scorer part, our model performs 3.61 points better, showing that the scorer and its associated loss are useful for the grasp prediction problem. By comparing the last two lines of Table 3.3, we also observe that the grasp quality score predicted by the scorer, based on the visual input and the grasp parameter prediction, is more accurate than the primary one, based only on the visual information.

Figure 3.15 shows some predictions made by our architecture on both the real-images of the Cornell dataset and synthetic images from our Jacquard dataset. As can be seen, our network is capable of predicting correct grasping positions in a wide range of configurations. Some proposed grasps are wide, with a large opening to wrap around the object, while some others are quite small, to fit a thinner part.

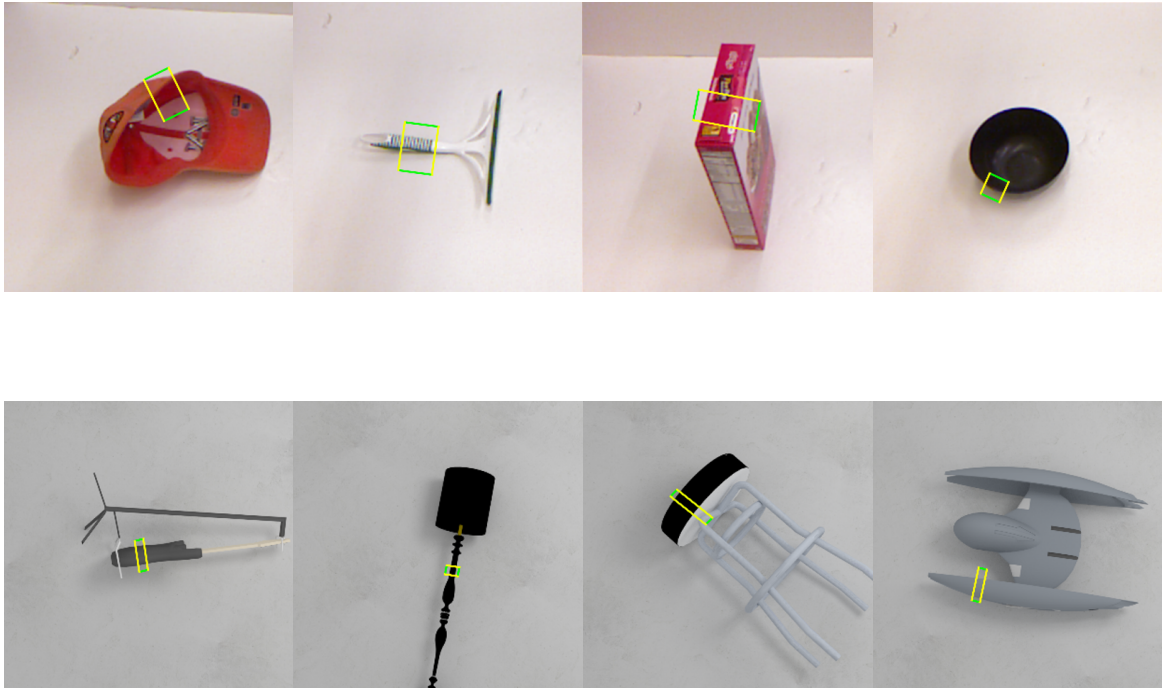


Figure 3.15: Grasp predictions for a parallel-plate gripper made by our network. Green lines represent the jaws, and yellow lines the opening of the gripper. Images on the top row are real ones from the Cornell dataset and images on the bottom row are synthetic ones from our Jacquard dataset.

**ANCHOR SIZE INFLUENCE** Contrary to [133] in which only one anchor of size  $54 \times 54$  with multiple orientations is used, in our architecture, the size of the anchor is computed by averaging all the ground-truth grasps in the training dataset. To evaluate the influence of this parameter on the final accuracy, we trained our model on the

Anchor size	54x54	54x27	108x54	108x27	Mean (91x26)
Accuracy	84.62%	85.39%	84.06%	84.53%	85.74%

Table 3.4: Accuracy of our model with different anchor sizes on the Jacquard Dataset

Jacquard dataset with different anchor sizes and ratios. The results are presented in Table 3.4. We can deduce from this table that anchor size is not the most crucial hyperparameter for this kind of approach, with an accuracy of 84.06% for the worst network and 85.74% for the best one with the mean anchors. However, using the mean box does not only provide the best performance but also simplifies the training process, as it removes the need for an intensive grid-search to optimize this hyperparameter.

**GRASP SELECTION INFLUENCE** To evaluate the effect of the selection of the  $T$  grasp parameter predictions using the primary score, we compared our full model to the same architecture but trained without this selection process: at training time, all the  $20 \times 20 \times k$  grasp proposals were processed by the scorer during the forward pass, and the  $T$  with the highest final score were used for back-propagation (instead of the  $T$  with the highest primary score). This model, in addition to being slower and consuming more memory during training, only has an accuracy of 82.36%, 3.38 points less than the model using the primary score predicted by PGP. This shows that using the primary quality score to train the scorer on hard examples (which are potentially good based on the visual input but not necessarily good once the real criterion is evaluated) helps the network to converge towards a better solution.

### 3.2.2.3 Real robot testing

One important question when training models on datasets is how they can generalize on real applications. Therefore, to evaluate this reality gap, we tested our model on a real robotic arm. We used 15 common household objects and 10 industrial parts to ensure a wide diversity in shapes, colors, and materials. Figure 3.16 shows our physical setup performing a grasp on one of the household objects. Similarly to what we did in Section 3.1.2.3, we considered a grasp was successful when the robot could lift the object, move it away, and put it back on the gripping area without dropping it in the middle. With this setup, we achieved an accuracy of 88.1% for our baseline without the scorer and 92.4% for our full model. Both were trained on the full Jacquard Dataset. This 4.3 points difference shows that not only our architecture performs better on the dataset it was trained on but also that it has a better internal grasp representation, allowing it to be more accurate in real case applications.



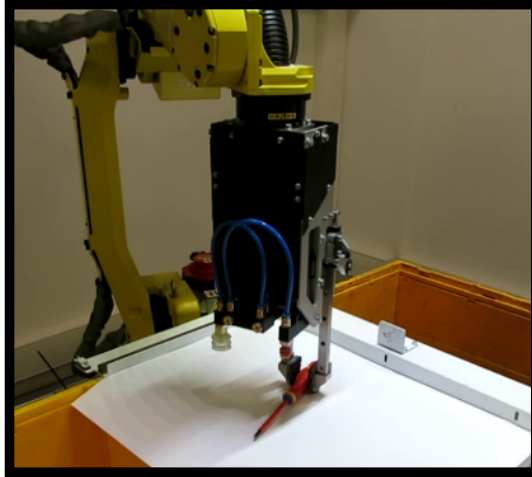


Figure 3.16: Picture of the robot performing a grasp predicted by our trained neural network with the scorer module on one of the household objects. The camera is located above the grasping area, looking downwards.

### 3.3 CONCLUSION

#### 3.3.1 *Summary*

In this chapter, we proposed a new method to generate synthetic images with grasp annotations, and we applied this method to a database of 3D models to create a large-scale dataset for robotic grasping, referred to as the Jacquard dataset. We also presented a new neural network architecture to learn grasp prediction from such images.

Specifically, we first detailed the proposed process to be able to generate realistic images with grasp annotations for a parallel-plate gripper using simulation. We evaluated the dataset generated with this method against the commonly used Cornell dataset [19] on a simple neural network architecture, and showed our larger Jacquard dataset leads to a better generalization ability of the network. We then presented in detail our new neural network architecture, using a scorer module to extend state-of-the-art approach [133]. We finally showed that this extension leads to both better grasp predictions on unseen images from the training dataset and better grasp performances on a real robotic system and real images of isolated objects.

#### 3.3.2 *Contributions*

State-of-the-art approaches for grasp detection in images predict at the same time grasp propositions and quality scores and are trained on hand-labeled real images. Unlike them, **our proposed architecture scores graspability based on grasp prediction and achieves state-of-the-art performances on our large Jacquard dataset composed of**

**automatically annotated synthetic images**, while performing similar to other approaches on the Cornell dataset. Our Jacquard dataset is available online<sup>1</sup> for the research community and has been used by over 130 research teams since its release in 2018.

### 3.3.3 Extension to bin-picking

The presented method only applies to images of isolated objects. As shown by Figure 3.17, it requires an additional segmentation step to be applied to bin-picking, with many instances in the image. However, this two steps process for bin-picking has some limitations and is not always able to generate an actual good grasp in the bulk context. Segmentation can indeed be incomplete, and thus the image of the object on which grasps are detected does not represent the real shape of the object. And even with a perfect segmentation, the grasp predicted on the isolated object can sometimes be unfeasible by the robot when transposed back in the bulk context. Figure 3.18 shows some of these cases where the approach can fail to grasp an object.

Moreover, this approach is based on supervised learning. While useful when large quantities of data are available for the studied problem, it can only lean on existing data. A new dataset has to be generated in case we want to change the considered gripper for example.

In Chapter 4, we will further investigate end-to-end self-supervised learning from the base image to overcome these limitations.

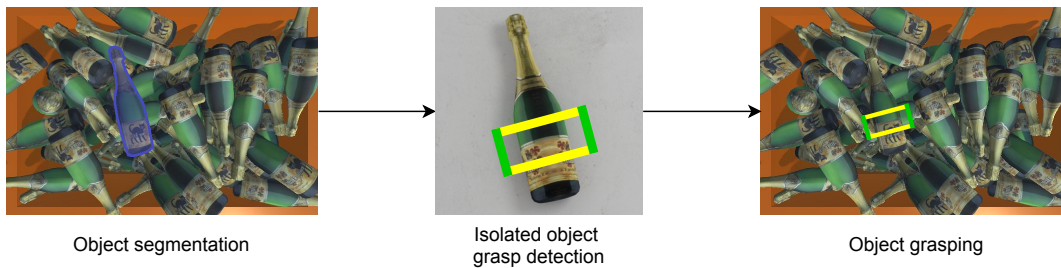


Figure 3.17: A two steps grasp detection process. First, the object is isolated from the background through segmentation. Then a grasp is predicted on the image of the isolated object. Last, this grasp is performed on the original scene.

<sup>1</sup> <https://jacquard.liris.cnrs.fr/>



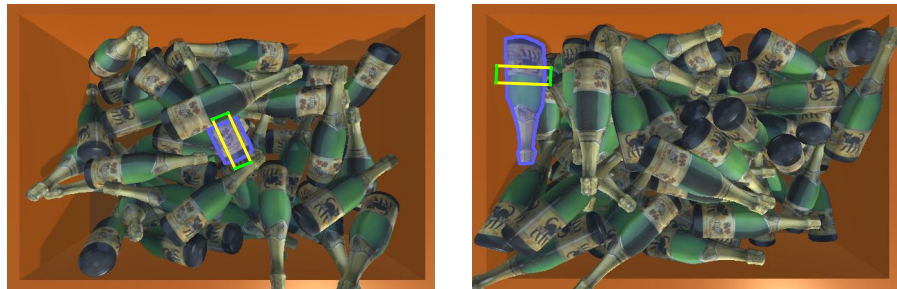


Figure 3.18: Examples of failed grasp detections on bulk images. Left: failure is due to an incomplete segmentation. Right: segmentation is correct, but the grasp is not valid in the context due to a collision with the containing box.

# 4

---

## SELF-SUPERVISED OBJECT AWARE BIN-PICKING

---

Segmenting objects out of bulk context to predict grasps on them is useful for model-free approaches. However, it causes issues when the context in the cluttered scene is crucial to predict the grasp outcome. To overcome this issue, we propose in this chapter a new model-free data-driven method trained in a self-supervised way using simulation. This chapter is organized as follows: [Section 4.1](#) explains why and how we use self-supervision to train our network instead of traditional supervision, then [Section 4.2](#) presents the simulation environment we built for interactive bin-picking, and [Section 4.3](#) describes the architecture of our object-aware grasp detection network. The experiments we conducted to test our network against other approaches are detailed in [Section 4.4](#), while [Section 4.5](#) shows how our architecture can be quickly adapted to new situations. Finally, we conclude this chapter in [Section 4.6](#).

### 4.1 SELF-SUPERVISED TRAINING

#### 4.1.1 *Motivations*

In [Chapter 3](#), the network is trained with images of many different isolated objects, annotated with all the possible good grasp locations. This diversity of shapes seen during its training allows the network to be very generic, and therefore generalize well to almost any object. However, generating such a dataset for piles of objects would be much more difficult for two reasons:

- simulating a pile of objects is slower than an isolated object, due to the many collisions occurring between them
- there are a lot more possible grasp locations to test in a pile than for an isolated object

Thus, obtaining a dataset with as many objects as the one presented in [Chapter 3](#) would not be feasible in practice. Obviously, this requires a paradigm change: the network can no longer be trained on thousands of different objects and shapes. Instead, we use a self-supervised

approach to train our network, progressively generating data when they are needed.

#### 4.1.2 *Self-supervised learning*

Self-supervised learning is a generic term defining approaches where a system can generate the data it needs to learn. In reinforcement learning, for example, an agent collects data samples interacting with its environment and learns from a reward signal [119]. Unlike reinforcement learning, the time aspect is not considered in our case. A grasp is not defined as an episode, with multiple steps sequentially executed: each successive grasp is considered independently from the other. However, the network still interacts with the environment and learns from a trial and error process. Our self-supervised method thus lies between traditional supervised learning [21], because it learns in a supervised way, and reinforcement learning, because of the way it interacts with its environment to get data samples.

The network generates the data it needs during the training by interacting with its environment. As the data are generated on-the-fly using the network itself, and not before the training, no heuristics or third-party detection methods have to be used. A base version of the network is thus trained on a few objects, learning core concepts of robotic grasping in this particular situation. Then, when a case with a different configuration is encountered, the network can be used to generate more data using the same self-supervised approach and be fine-tuned with them if necessary. This approach is ideal for real-life applications, as the whole adaptation process can be automatized, without any human intervention (either image annotations, or coding to adapt the simulation environment to a new situation). [Figure 4.1](#) summarizes the two approaches: a generic network trained in a supervised way, and an adaptable network trained with self-supervision.

#### 4.1.3 *Proposed self-supervised approach*

To perform the self-supervised training, two processes are running simultaneously: one to train the neural network using the generated data, and one to generate data using the partially trained neural network. [Figure 4.2](#) illustrates this training loop. This loop is independent of the nature (simulated or real) of the environment used to gather annotated data.

The data generation module runs in parallel to the training module: while the network is learning from previously generated data, new ones are generated using a copy of the same network. It means that except for the initial data generated to initialize the network, using our self-supervised approach does not add any additional time to the overall duration of the training procedure.

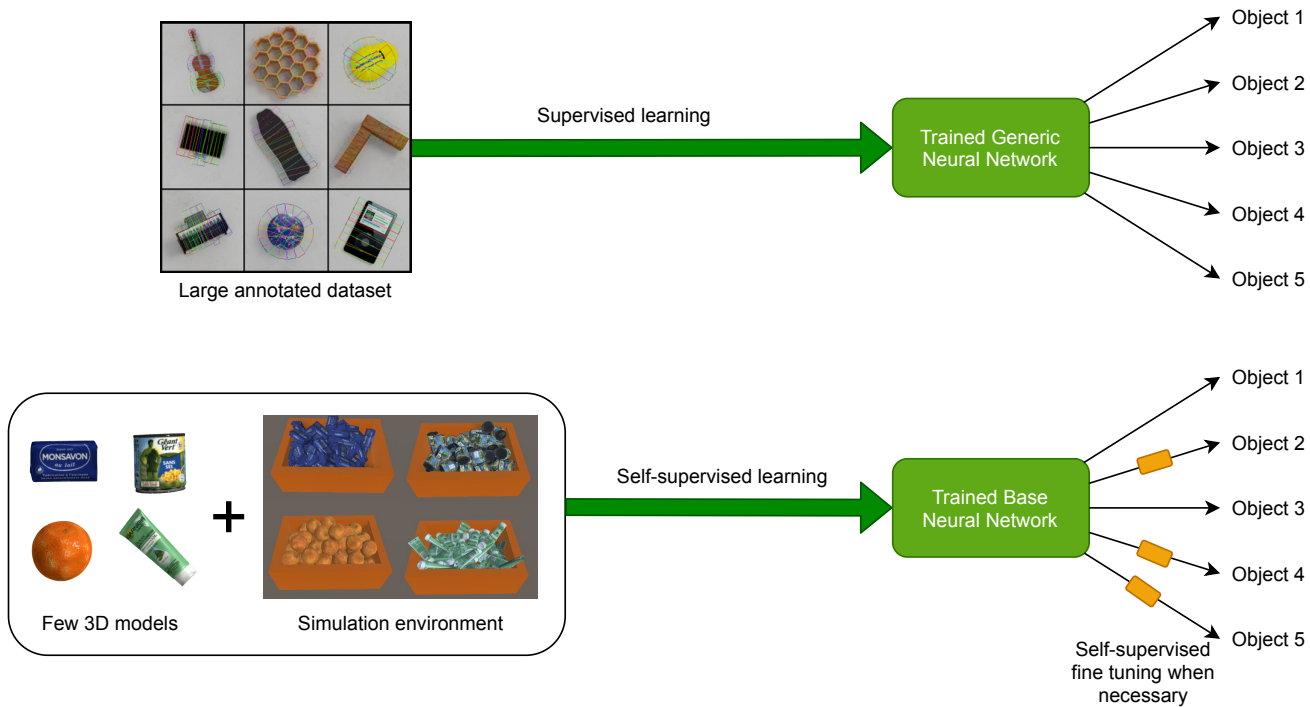


Figure 4.1: Comparison between supervised and self-supervised approaches. In the self-supervised approach, the network is less generic but can be more easily fine-tuned to be adapted to new situations, like new objects.

The data module takes trained weights as an input, uses them to generate new data by interacting with the environment, and outputs some new data to be used to train the network.

Both network weights and generated data are not transferred from and to the other module in real-time. Instead, the synchronization is performed at each training epoch, similarly to what is done in reinforcement learning [111]. When the network has seen each data sample once, the weights of the data generation network are updated with the new ones, and the generated data are sent to the learning module to be stored in a replay buffer. To avoid infinite memory usage due to the many samples that are generated, the oldest samples are discarded and replaced by the newest ones. Algorithm 1 summarizes the process used for data generation.

As the simulation environment does not regenerate a new pile of objects after each grasp attempt, but rather only when the bin is almost empty, successive data samples created on the same simulation setup are highly correlated, sometimes with part of the image being strictly identical. Initially developed for reinforcement learning [78], this approach of storing the data in a replay buffer instead of using them in real-time helps to stabilize the training by breaking this correlation, distributing successive data samples into different batches for the training.

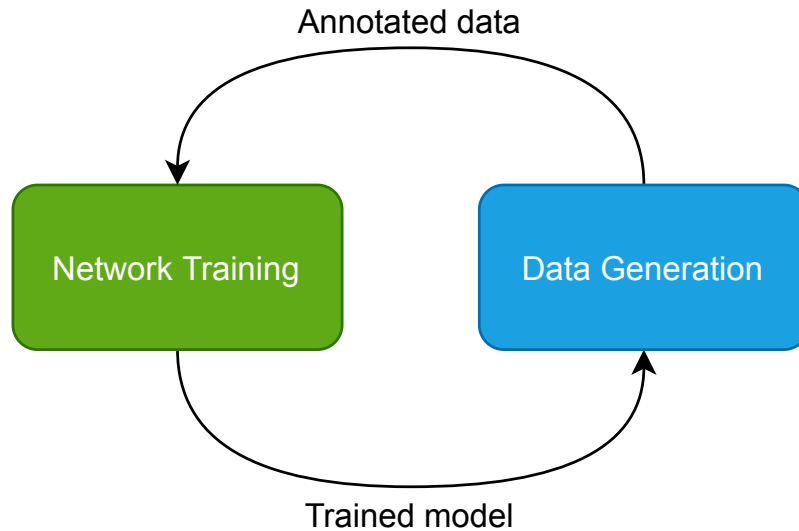


Figure 4.2: Illustration of our self-supervised training loop. The network training module updates the weights of the neural network, while the data generation module uses these weights to propose grasp parameters to the robotic environment (which can be either simulated or real).

#### 4.2 SIMULATION ENVIRONMENT

Data-driven methods for grasp prediction require data to be trained on. This is even more true for self-supervised training, where data have to be generated on-the-fly during the training procedure. Using humans to annotate images is not feasible for training that can last for many days. As robots do not need to sleep or eat, it is possible to use them to gather data automatically [66] [55], but it can take several months in these conditions for one training to be performed. Waiting such a long time is not acceptable in many -if not all- real-life applications in the industry.

To overcome this issue, we advocate once more the use of simulation to quickly gather a large amount of data. However, as data have to be generated during the training procedure, it is not possible to use an offline data generation similar to the one used in [Chapter 3](#) to create a large-scale database before starting the training. Instead, we need an interactive environment that is capable of quickly generating data as they are needed by the training. More specifically, the environment needs to meet multiple constraints:

- physics: simulating bulks of objects require a physics engine capable of handling many collisions and propagating them in a realistic way in real-time
- rendering: as we want to train on images, the environment has to be able to simulate camera output in real-time

---

**Algorithm 1:** Detailed data generation process for one training epoch

---

**Input:**  $W_{train}$ , network trained weights  
**Output:**  $data$ , set of annotated data samples

```

 $data \leftarrow []$ 
 $I \leftarrow \emptyset$ 
 $g \leftarrow \emptyset$ 
 $r \leftarrow 0$ 
update the network copy with  $W_{train}$ 
while training epoch is not finished do
     $I \leftarrow$  image from an idling agent in the simulation
    environment
    feed the network with  $I$  and get a grasp proposal  $g$ 
    send  $g$  to the agent
     $r \leftarrow$  outcome of  $g$ 
     $data \leftarrow [data, \{I, g, r\}]$  // append new sample to  $data$ 
end while
send  $data$  to the training module

```

---

- communication: the environment must have a two-way communication channel with the neural network to allow self-supervised training as described in [Section 4.1](#)

To implement such an environment, we used the Unity software [125]. Unity’s main purpose is developing video games. As such, it is developed to emulate a real-time simulation of virtual worlds, with physics interactions, and rendering through synthetic cameras. It also has an ML-Agents toolkit [54] that can handle two-way communication to train machine learning algorithms and has been proved to be useful to generate synthetic data to train object detection algorithms [51]. [Figure 4.3](#) illustrates how a learning agent and the environment interact with each other during our self-supervised training loop.

To speed up the generation process and generate data for multiple objects simultaneously, the individual robotic simulated setup is cloned multiple times. This results in parallel agents collecting data samples independently from each other, which has been proven to be more efficient for reinforcement learning tasks [79]. In our experiments, we used a simulated environment with 12 agents. They all have the same characteristics and behave exactly the same.

#### 4.2.1 Physics simulation

Simulating bulks of objects colliding with each other is much more difficult than simulating the collisions of one isolated object as presented in [Chapter 3](#). Instead of just computing the reaction of the floor

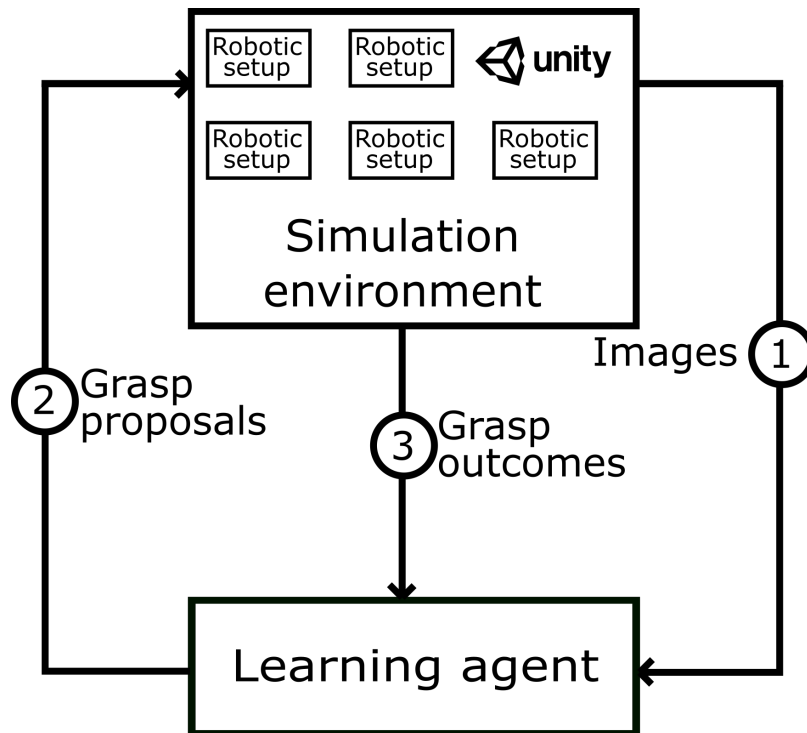


Figure 4.3: Interactions between the learning agent and the simulation environment

on the object, the forces applied by one object on all of its neighbours have to be calculated, and propagated to their own neighbours, until an unmovable object (like the floor, or one side of the containing box) is met. In our simulation environment, all these calculations are made by Nvidia’s PhysX [87] physics engine.

#### 4.2.1.1 Training objects selection

Even if PhysX has been improved to handle cases like that with its version 4, accurately simulating large piles of objects is still difficult, and can often lead to instabilities, with large and unpredictable forces created inside the stack. This phenomenon occurs mainly when the collision models of the objects are complex, with many faces, or with long and pointy parts.

Because of this reason, it is not possible to use any arbitrary 3D model as we did in Chapter 3. Instead, we used a set of objects scanned with a 3D scanner. Some examples of these objects can be seen on Figure 4.4. To reduce even more the physics simulation costs, the collisions are not computed directly on the models, but on simplified versions using V-HACD decomposition [76]. The difference between visual 3D models and the decompositions can be seen on Figure 4.5. All the models have been manually scaled and tested, to ensure the resulting simulated pile was stable, even with stacks of many objects.



Figure 4.4: Examples of our scanned 3D objects used to generate piles of objects. Objects have various shapes and textures.

#### 4.2.1.2 Bulk generation

To generate stacks of objects that are as close as possible to those encountered in real-cases applications, all the objects are placed inside a rectangular box. This box is stationary and measures  $600 \times 400 \times 300$  mm. The models are scaled so that the box can contain roughly 50 independent objects. The mass of the object is arbitrarily assigned to 1 kg. Having a different mass would not change the behaviour of the simulation, but having a constant mass of 1 kg makes it easier to set the parameters of the grippers. To ensure diversity in the piles of objects, each object is instantiated with a random position and orientation above the box and then falls to form a stack with high levels of occlusions and cluttering. [Figure 4.6](#) shows examples of simulated homogeneous stacks inside the simulation environment.

During the simulation, new piles are not generated after each grasp attempt, as it would be too slow. Instead, piles are always initialized with 50 objects, which are then removed one by one as interactions with the learning module occur. To avoid a deadlock in the simulation in case the last object is not graspable at all, the pile is regenerated



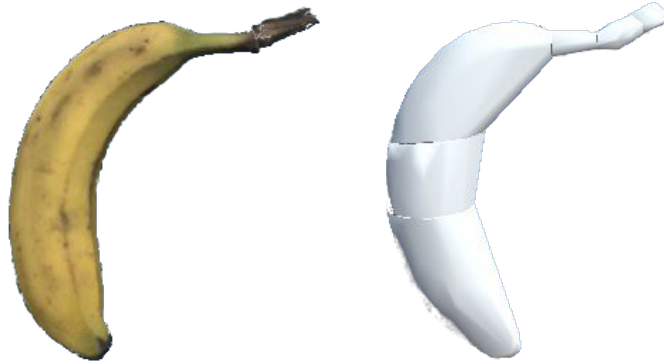


Figure 4.5: Comparison of the two models used. Left is the detailed model used for visual rendering, right is the simplified model composed of multiple convex parts used for collisions.

when it has less than 5 objects left instead of 0. Given the scale of the objects and the size of the bin they are in, the objects are very often far apart from each other when the reset happens. Therefore, the images with less than 5 objects or less are very similar, with almost no occlusion, meaning using this early reset does not reduce the diversity in the data.

#### 4.2.2 Image generation

Unlike in [Chapter 3](#), it is not possible to render multiple images with ray-tracing to get realistic images. Even if rendering an image with ray-tracing is possible in real-time with modern GPU, it would be too slow to be done multiple times in parallel for multiple agents. Instead, the rendering of the images is done through rasterization, a much faster technique than ray-tracing. The output images are not as realistic as they could be with a ray-tracing method, but it allows much faster rendering times, which are necessary to generate data to feed the neural network on-the-fly.

As the environment is entirely simulated, not only it is possible to get RGB and depth images, but also other information, without any human involvement needed. In our simulation, each of the stacks of objects is rendered 4 times in total, resulting in 4 different images:

- RGB image, showing the objects with their textures
- depth image, showing how far is each pixel from the camera

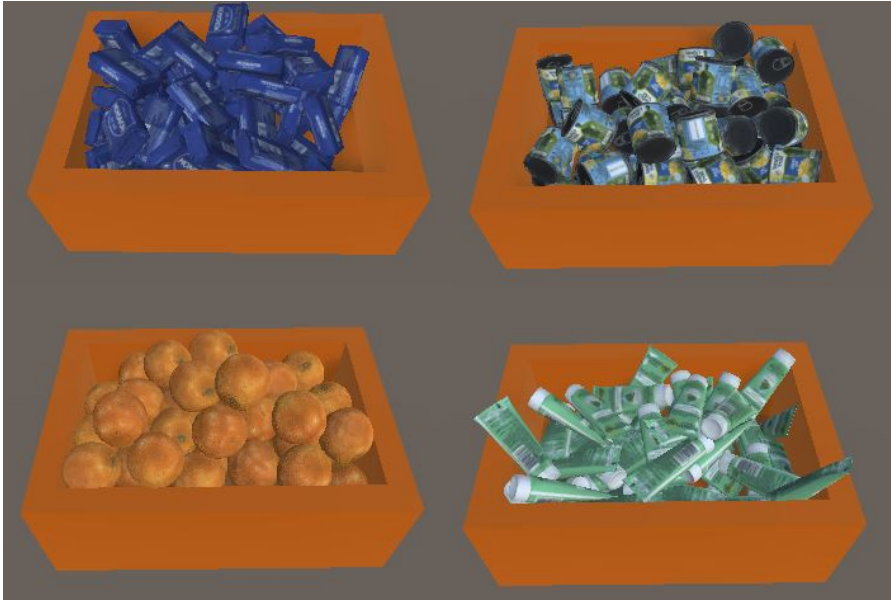


Figure 4.6: Examples of simulated stacks of objects in our simulation environment. With many objects, there is a high rate of occlusions and cluttering.

- segmentation image, where each object is colored with a unique grey value
- normal image, with three channels representing the components along  $x$ ,  $y$  and  $z$  of the normal vector at any given pixel

Examples of images from the simulation environment can be seen on [Figure 4.7](#).

### 4.2.3 Grasp simulation

Simulating stacks of objects and rendering images is not sufficient to generate data to learn grasp prediction. The environment also has to be able to determine whether or not a grasp proposition is valid. Unlike in the Dex-Net approaches [73] [74], where the grasp outcome is predicted with static analysis, we are leveraging the power of the physics engine to simulate not only the grasp but also its stability when the object is pulled out of the stack and transported away.

We implemented two of the most used kind of grippers in our environment: a vacuum suction-cup-based gripper and a parallel plate gripper. The two next sections will present each of these grippers' implementations.

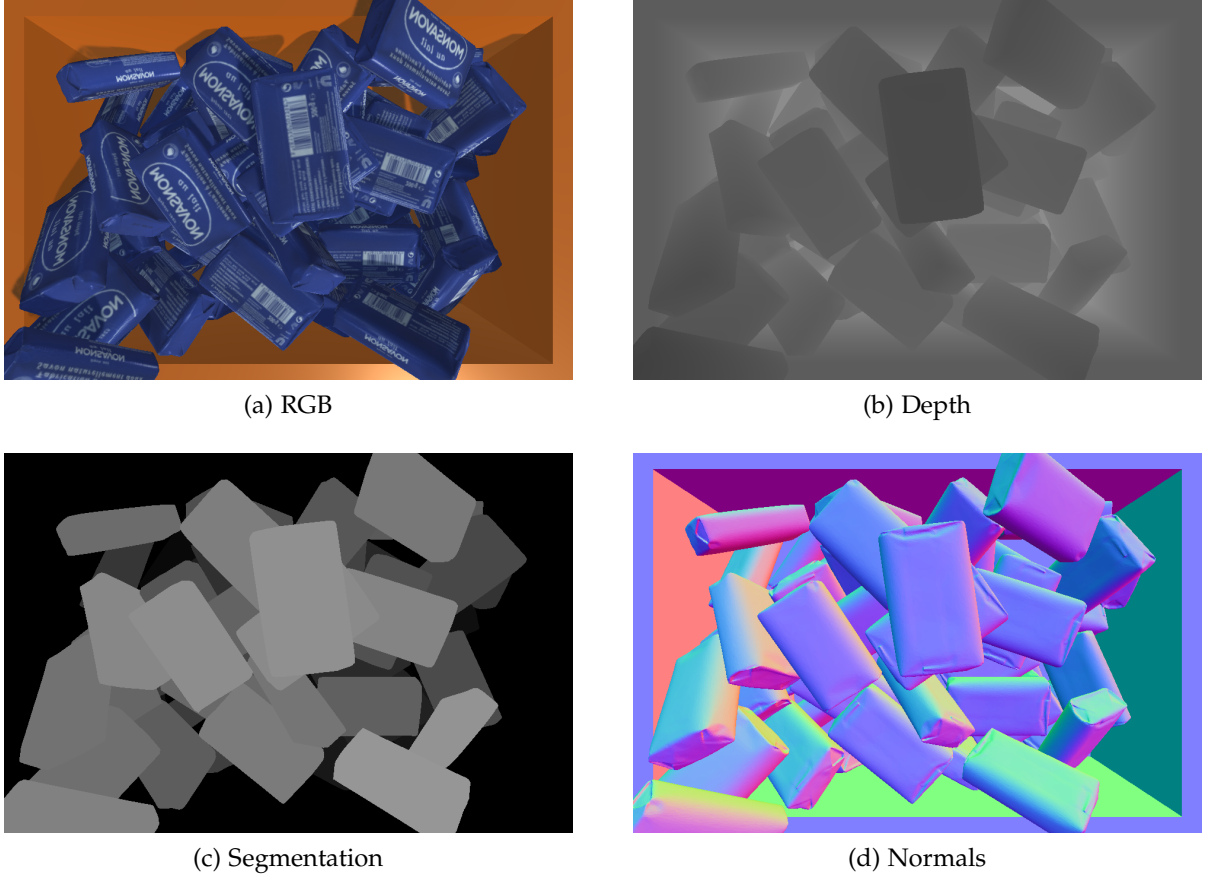


Figure 4.7: Visuals of all the output images of our rendering module

#### 4.2.3.1 Parallel-jaw gripper

A parallel-jaw gripper is composed of two parallel fingers that enclose an object. When the gripper is closing, the object lying between the fingers is pinched.

The simulation of this process is done in a very similar way to the one presented in [Chapter 3](#) by simulating the forces naturally generated by the collisions between the object and the jaws. The closing force of the jaws is set so the gripper can lift our 1 kg objects with a little margin. This means that, if the grasp is far from the object's centre of mass, the object can slip and rotate around the grasping axis. To be considered successful, an object has to be correctly extracted from the pile and moved away to a designed location outside of the box. To maximize the chances of being successful, a grasp has thus to be not only correctly aligned over an object but also to be centered on it, to avoid the object dropping during the transport phase. [Figure 4.8](#) illustrates the different phases of a grasp.

To be performed in the simulation environment, a grasp requires 8 parameters:  $\{g_x, g_y, g_z, \delta, \theta, g_{nx}, g_{ny}, g_{nz}\}$ , where  $\{g_x, g_y, g_z\}$  are the three space coordinates representing the position of the middle point

between the two fingers on axes  $x$ ,  $y$  and  $z$  respectively,  $\delta$  is the opening between the two fingers,  $\theta$  is the orientation of the jaws and  $\{g_{nx}, g_{ny}, g_{nz}\}$  is the normal along which the gripper approaches the object before closing the jaws. To be closer to real-life applications, the width of the jaws is not a grasp parameter, but rather a simulation parameter instead. Of course, multiple grippers with different widths can be used, but they can not be changed after the environment is loaded.

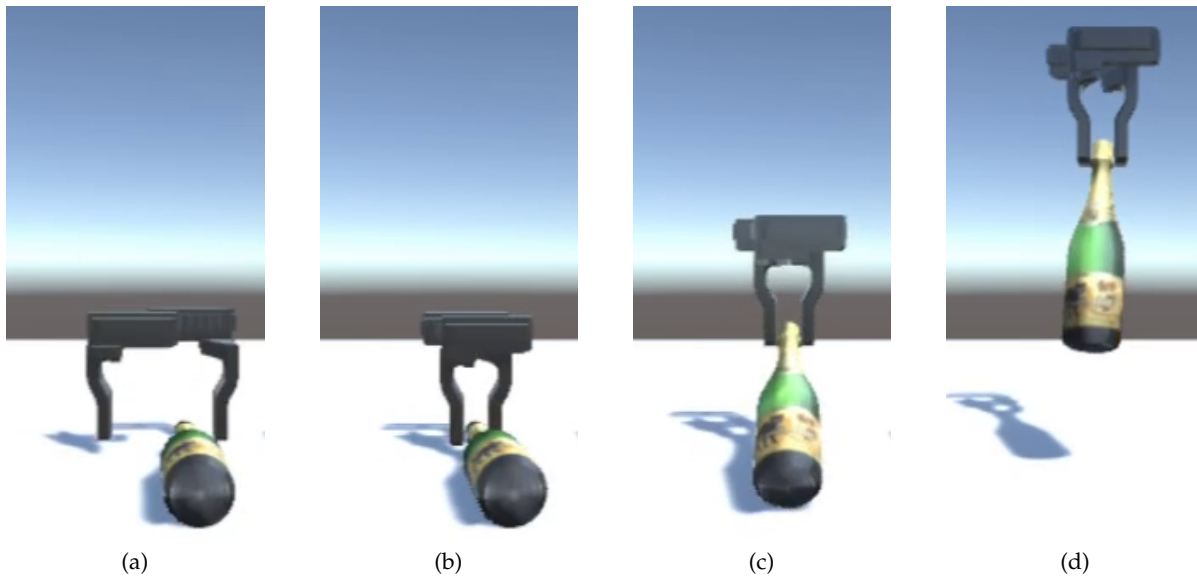


Figure 4.8: visualisation of the different steps of a parallel-jaw grasp in a stack-free environment for better visualisation. (a) the fingers are placed around the object at the designated location (b) the gripper is closed and the object is grasped (c) as the gripper is moved away from the grasping location, the object is rotated around the grasping axis due to its own weight (d) the object is successfully lifted

#### 4.2.3.2 Vacuum gripper

A vacuum gripper can be as simple as one suction cup, but more complicated structures also exist and are commercially used, as illustrated by Figure 4.9. They work in the same way, but with multiple grasping points, allowing either to catch several objects simultaneously or to catch one object more stably.

To grasp an object, a vacuum gripper uses an air pressure difference between the inside and the outside of a membrane. This pressure difference is created by a vacuum generator removing the air inside the membrane, creating a force between the gripper and the object obstructing the hole in the membrane. This force can obviously only exist if the pressure inside the membrane is very low, *i.e.* if there is no major leak between the membrane and the object. Even if there is



Figure 4.9: Examples of vacuum grippers with different numbers of suction cups and dispositions.

no air leak, the success of the grasp is not guaranteed: the object may also fall during transport. Figure 4.10 illustrates the process of a grasp using a vacuum gripper.

Contrary to the parallel-plate gripper, which is simulated only using collisions and forces resulting from them, air pressure effects can not be directly simulated inside the environment. Instead, we approximate the phenomenon using position constraints. The following of this section will describe how we simulate the grasp for one suction cup, but the same process can of course be applied to multiple suction cups simultaneously.

A suction cup grasp requires 6 parameters to be performed in our simulation environment:  $\{g_x, g_y, g_z, g_{nx}, g_{ny}, g_{nz}\}$ , where  $\{g_x, g_y, g_z\}$  are the three space coordinates representing the position of the centre of the suction cup disc on axes  $x$ ,  $y$  and  $z$  respectively, and  $\{g_{nx}, g_{ny}, g_{nz}\}$  is the normal along which the gripper approaches the object.

**AIR LEAK CHECK** During the approach phase along the normal, whenever a collision between the gripper and the objects occurs, the surface of contact between the suction cup disc and the object is estimated. To do so, we check the distance between the gripper and the collided object at  $N$  points sampled over the suction cup surface and assume that contact is indeed locally formed for each point if this distance is under 10 mm. This threshold has been chosen to model the maximum deformation of the flexible membrane of the suction cup. To sample those points inside the disc, we do not use a uniform distribution over  $r$  and  $\theta$ , as it would lead to a higher concentration on

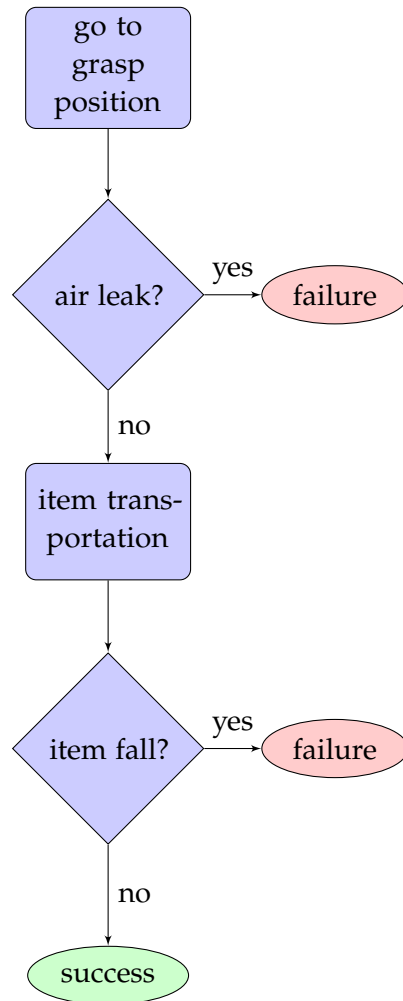


Figure 4.10: Process of a suction cup grasp. The grasp can fail either because the suction cup is badly positioned, or because the grasp is not stable enough to carry the object to the drop position.

the centre of the disc. Instead, we use the following equation to sample them along a spiral and uniformly cover the whole disc surface:

$$\begin{cases} r_i = \sqrt{\frac{i}{N}} \\ \theta_i = i\pi(1 - \sqrt{5}) \end{cases} \quad (4.1)$$

where  $i$  is the index of the  $i^{\text{th}}$  points on the disc. Figure 4.11 shows the resulting distribution of the points with  $N = 100$ , as well as two examples of distance checks inside the simulated environment.

If the estimated contact surface is over 95%, the contact is considered sufficient, with no major air leak preventing the grasp to be performed. This 95% threshold has been set empirically by observing real vacuum grippers and their behaviour regarding partial contact grasp.

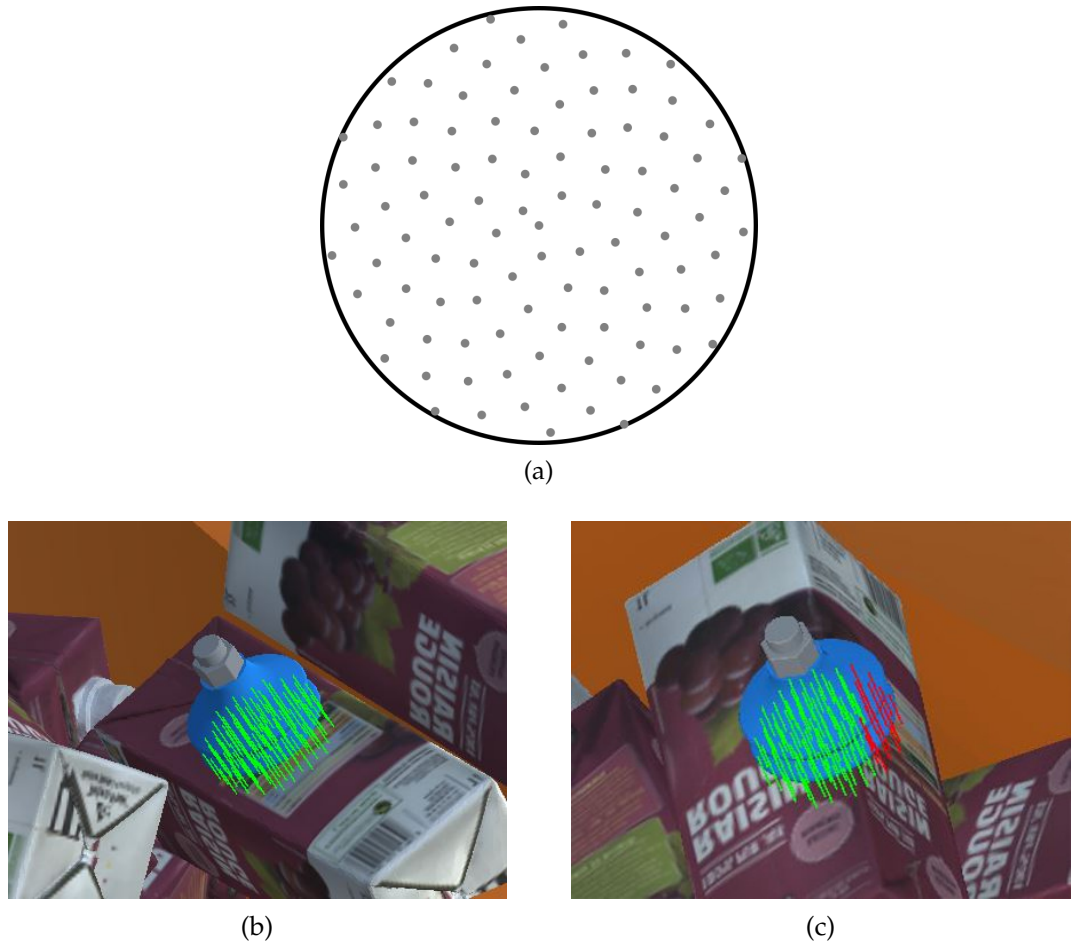


Figure 4.11: Examples of disc sampling. (a) uniform sampling of 100 points on a disc using Equation 4.1 (b) and (c) distance check in the simulation environment. Points where the suction cup is considered as locally in contact with the object are drawn in green, points where there is no contact in red.

**ROBUSTNESS CHECK** Having a good contact surface with the object does not guarantee a successful grasp outcome: the grasp also has to be stable enough to hold the object during the transportation of the object to its final position. To simulate this step of the process using the physics engine, we add a position constraint between the gripper and the object. This constraint forces the object to remain at the same position relative to the gripper from the moment the constraint is created, *i.e.* at grasp time, to the moment it is dropped to its final location.

Both force and torque applied to the object by the constraint to maintain it in position are limited. This means that if at any moment, either the force or the torque applied is above a threshold, the constraint is destroyed, and the object is dropped from the gripper. To be consistent with real-life applications, in which larger vacuum grippers can be



used to lift bigger objects, we set the thresholds relative to the suction cup radius  $R$ . Specifically, we used the following equation to set them:

$$\begin{cases} F_{max} = 50 \frac{R}{0.02} \\ \tau_{max} = 1.5 \frac{R}{0.02} \end{cases} \quad (4.2)$$

where  $R$  is expressed in meters.

As these values depend on  $R$ , multiple vacuum grippers with different properties can be created inside the simulation environment. With a smaller radius, it is easier to fulfill the surface contact criterion, but the grasp will be weaker and has to be performed on the centre of mass of the object to avoid too much torque breaking the link. On the opposite, a larger radius allows the gripper to be strong enough to grasp objects even from an off-centered position but will be more difficult to place on a flat surface to avoid air leaks.

The maximum values have not been chosen to maximize realism. Instead, they have been so grasping in the simulation environment is a challenging task, with different behaviours between multiple suction cups with different radii. Thus, the network does not only have to understand where the gripper should be sent to grasp an object but also has to carefully select the gripper that best matches the characteristics of the grasp.

#### 4.2.4 Interactive environment

We used the ML-Agents toolkit [54] to implement the two-way communication channel required to have an interactive environment. The simulation sends images and rewards corresponding to grasp outcome, and receives grasp locations, as illustrated by Figure 4.3.

To speed up data generation, and allow the generation of data from multiple objects simultaneously, the simulated robotic setup is duplicated multiple times, similarly to what is done in reinforcement learning [79]. Each of them has an independent bin, cameras, and grippers. Whenever one of these instances is idling, *i.e.* is not performing a grasp, all its cameras render the scene and the resulting images, as well as a unique identifier of this instance, are sent through the two-way communication channel, to be processed. The instance then waits for an answer as grasp coordinates. Once received, the corresponding grasp is simulated and the outcome sent back through the communication pipeline, before returning in an idling state and continuing the cycle.

Of course, the communicating agent does not have to learn from the data and can be replaced without any change in the simulation code, as long as it is able to propose grasps to the simulated robotic instances.



### 4.3 OBJECT-AWARE GRASP DETECTION NETWORK

Images of piles contain, by definition, many instances that can occlude each other. Achieving good grasping performance in these conditions requires an understanding of how objects are located in the scene.

A traditionally used approach by model-free methods is to predict object instances to then send the gripper to an estimated object centroid [36]. In this section, we present our object-aware architecture, which can predict both object instances and grasp quality estimations.

#### 4.3.1 Network architecture

The global architecture of our network can be seen on Figure 4.12. Similar to previous approaches [81] [108], our architecture is a Fully Convolutional Network (FCN), capable of estimating the quality of hundreds of potential grasps simultaneously. The following paragraphs describe each of its components in detail.

##### 4.3.1.1 U<sup>2</sup>-Net backbone

The backbone of our network is based on the recent U<sup>2</sup>-Net architecture [95]. Besides being a state-of-the-art network for Salient Object Detection, U<sup>2</sup>-Net architecture is known for its simplicity and ability to be trained from scratch, without using pre-trained weights from image classification tasks. This is important for grasp detection tasks, especially in bin-picking, as data follows a distribution very different from datasets traditionally used for pre-training, like ImageNet [22].

Just like the U-Net network [103], U<sup>2</sup>-Net is an encoder-decoder architecture. The data is processed by a set of modules, with downsampling (respectively upsampling) steps between them in the encoder (respectively in the decoder). However, instead of being sequentially chained, the output of each module of the encoder is linked not only to the next module but also to the corresponding module of the decoder, giving a U-shape to the global architecture.

In addition to this global U-shape, each individual module of the encoder and the decoder are themselves built as small U-Nets, called **Residual U-block** (or RSU), thus the <sup>2</sup> in U<sup>2</sup>-Net. These RSU blocks increase the depth of the whole architecture without significantly increasing the computational cost. Figure 4.13 presents the detailed architecture of an RSU block. An RSU block has multiple parameters defining its internal structure:

- its depth, in number of convolution blocks  $D$
- the number of filters in each convolution blocks  $M$
- the number of channels in the input features  $C_{in}$

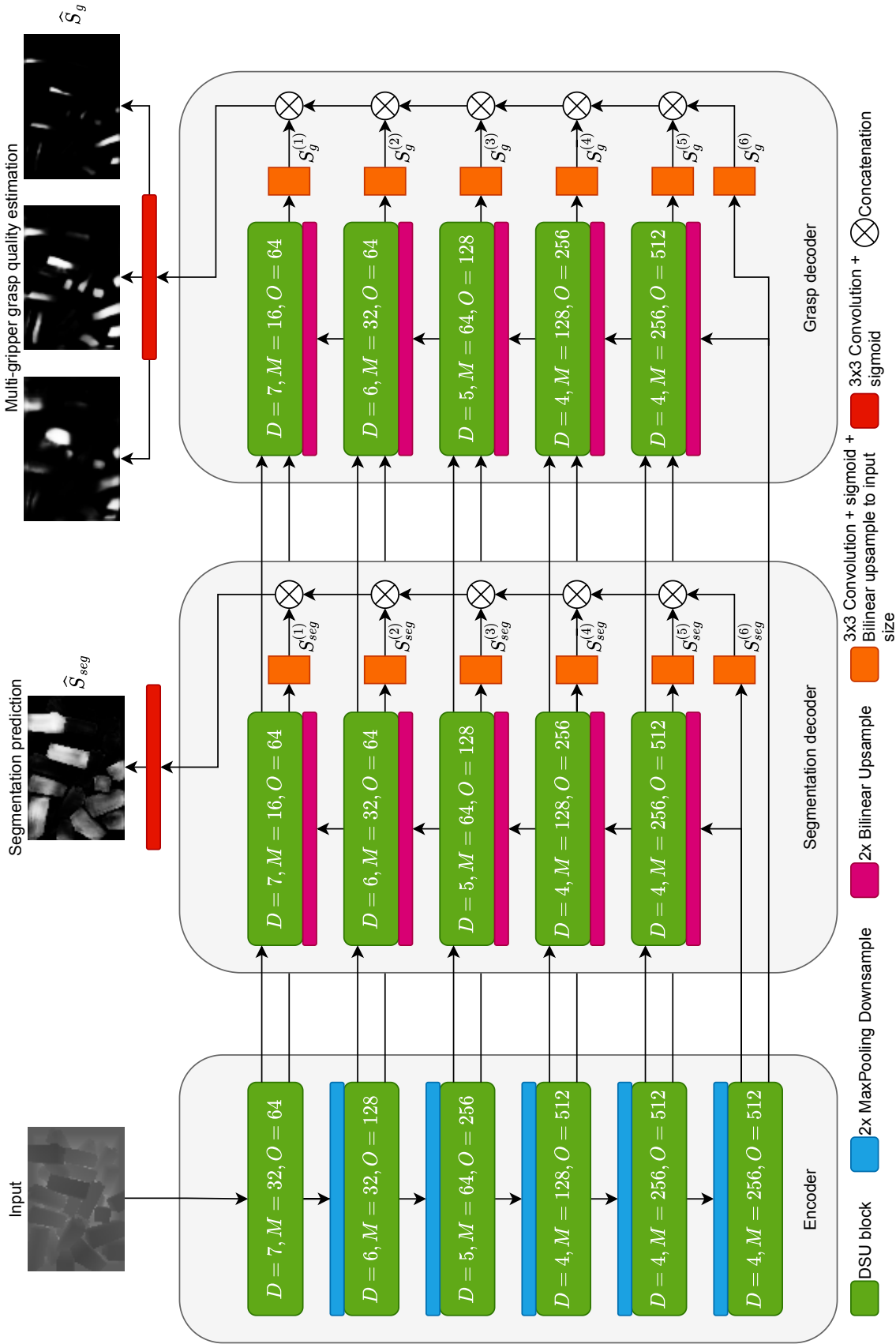


Figure 4.12: Full architecture of our network with the encoder and the two linked decoders. The decoders have both intermediate outputs at different scale levels. All these outputs are merged to form the final prediction of both the instance segmentation and the grasp quality estimation.

- the number of channels in the output features  $C_{out}$

#### 4.3.1.2 Cascaded decoders

Compared to a network used for Salient Object Detection, our network does not only need to predict a segmentation of the image for instance object estimation but also grasp quality estimation. To do so, we used a cascaded decoders approach [37]. Cascaded decoders have originally been developed for edge detections, with multiple modalities (like boundaries and occlusions) predicted by chained decoders. Instead of using only the encoder features as input, as in traditional encoder-decoder architectures, each decoder also has the output of all the previous decoders to base its prediction on. This allows the decoders to use not only the generic features extracted by the encoder but also the specific features created by the previous decoders. In our case, we use a structure with two decoders: one for the instance segmentation, and one for the grasp quality estimation. They are both identical architecture-wise, except for the top convolution, which is different depending on the considered task.

**GRASPABLE INSTANCE SEGMENTATION** Using the multi-scale features from the encoder, the first decoder task is to predict a segmentation of the instances in the image. As images can contain many objects in bin-picking scenarios, segmenting out all the instances of the objects in the image, even the small part of an almost fully occluded instance, is a quite difficult task. Knowing that kind of information is moreover not very useful for the grasping problem, as most of the time, to be successful, a grasp has to be performed on an object that is on the top of the pile.

That is the reason why we introduce a new kind of segmentation, named graspable segmentation. The goal of the decoder is not to predict a full segmentation of the image anymore. Instead, we focus on the information that is indeed useful to help the grasp prediction: instances that are on top of the pile, *i.e.* objects that do not have a part of them occluded by another one. [Figure 4.14](#) shows the difference between the two types of segmentation, while [Algorithm 2](#) details the process to compute the graspable segmentation from a depth and a full segmentation image. Thus, the problem is much more simple for the network, as the output becomes a single-channel image where each pixel has to be set to 1 if it is part of a graspable instance, and 0 otherwise.

To perform its task successfully, the network has to analyze features at different scales in the image. To help in doing so, each RSU block is followed by a 3x3 convolution, trained to predict the same segmentation at different scales. Thus, our segmentation decoder has 6 output  $\widehat{S}_{seg}^{(i)}$ , one for each scale factor. In addition, there is one more

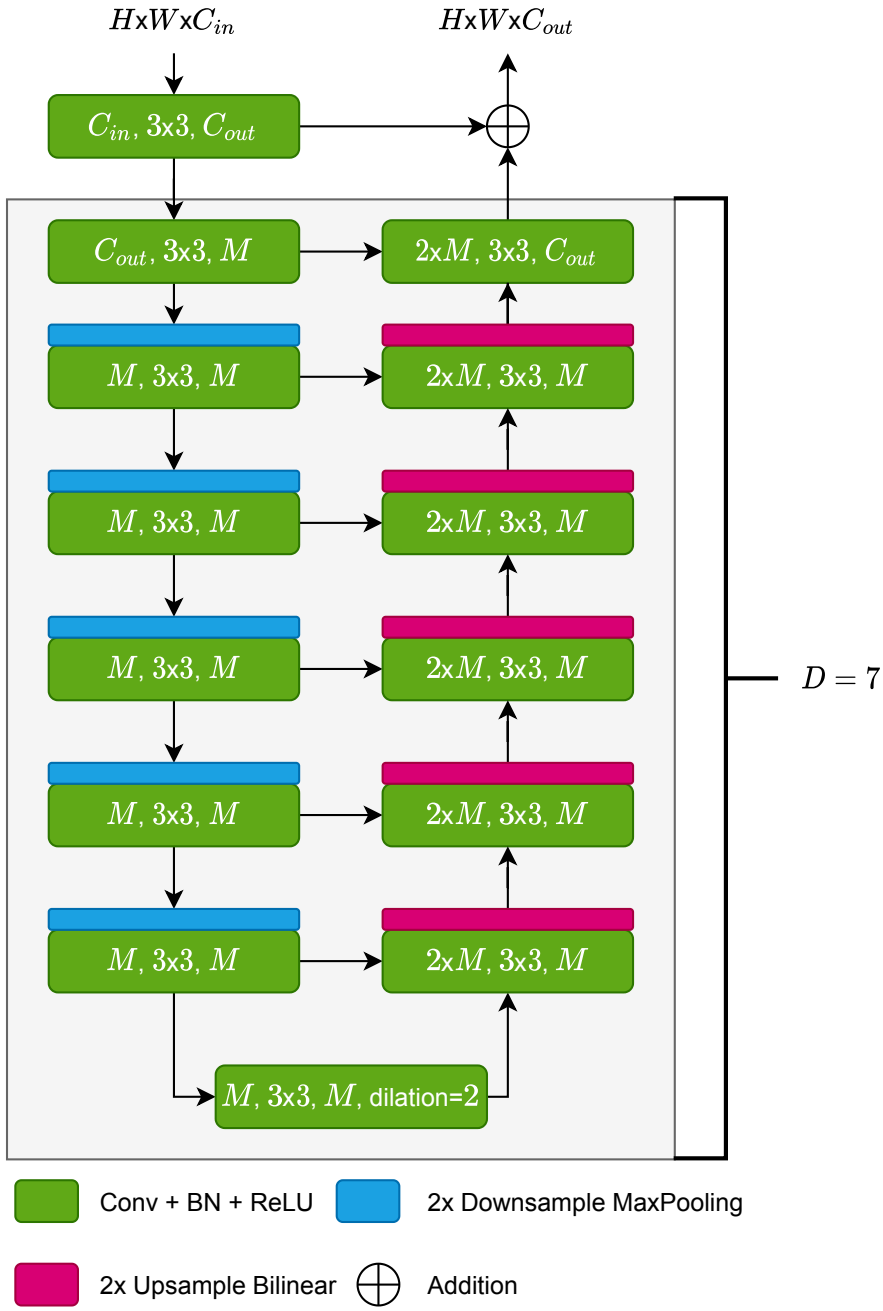


Figure 4.13: Detailed architecture of an RSU block with a depth of 7. The input features are downsampled and upsampled back to their original size so in total the dimension of the features does not change in the process. For each convolution block, the text represents input channels, kernel size, and output channels respectively.

---

**Algorithm 2:** Graspable segmentation generation algorithm

---

**Input:**  $d$ , depth image  
**Input:**  $s$ , full segmentation image  
**Output:**  $S_{seg}$ , graspable segmentation ground-truth image  
 $S_{seg} \leftarrow 0$   
**for** object  $o \in s$  **do**  
   $is\_occluded \leftarrow \text{false}$   
  **for** pixel  $p \in o$  **do**  
     $neighbours \leftarrow$  pixels adjacent to  $p$   
    **for** pixel  $p' \in neighbours$  **do**  
      **if**  $d(p') < d(p)$  **and**  $s(p') \neq s(p)$  **and**  $p' \notin background$  **then**  
         $is\_occluded \leftarrow \text{true}$   
      **end if**  
    **end for**  
  **end for**  
  **if not**  $is\_occluded$  **then**  
    **for** pixel  $p \in o$  **do**  
       $S_{seg}(p) \leftarrow 1$   
    **end for**  
  **end if**  
**end for**

---

3x3 convolution at the top of the decoder, taking as inputs all these intermediate results  $\widehat{S}_{seg}^{(i)}$  and outputting the final decoder prediction  $\widehat{S}_{seg}$ .

**GRASP QUALITY ESTIMATION** The second decoder of our architecture is the one actually responsible for estimating the grasp quality of grasps at many locations. Depending on the number and nature of the grippers the network is trained for, the output of this decoder can vary. In this section, we will describe the architecture trained to predict grasps for three suction cups of different radii, as illustrated on [Figure 4.12](#), but the top layers could be adapted for any combinations of suction cups and parallel-jaw grippers.

As the grasp prediction decoder is located after the segmentation one in the architecture, it takes as input not only the encoder multi-scale features but also the features previously calculated by the segmentation decoder. These features are concatenated before each RSU block and then processed by all the blocks in the grasp prediction decoder.

Just like for the graspable segmentation, the grasp parameters are predicted at each scale in the decoder, concatenated, and then fed to a convolution layer outputting the final prediction. In total the decoder has 7 outputs: 6 intermediate  $\widehat{S}_g^{(i)}$  and one final  $\widehat{S}_g$ . Each one

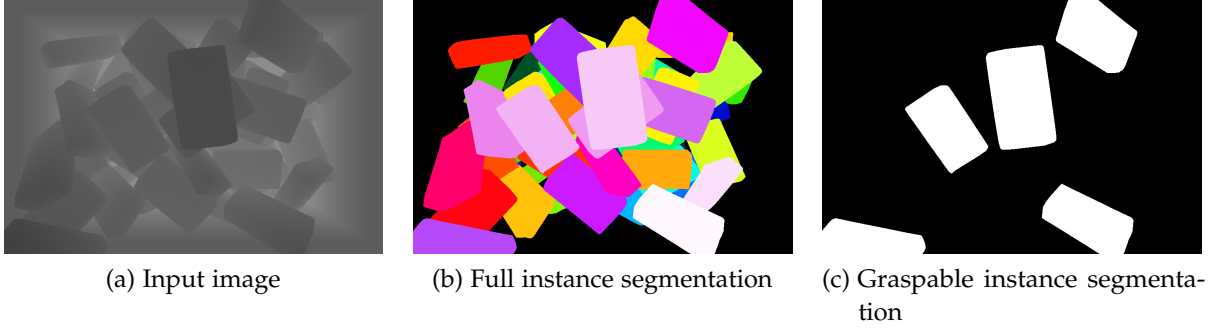


Figure 4.14: Comparison of two types of segmentation applied on the same input depth image (a). (b) full instance segmentation, each pixel is colored depending on the instance it belongs to (c) our graspable segmentation, only the pixels belonging to the instances that are on the top of the pile are set to 1.

of these outputs is a 3 channels images. Each one of these channels corresponds to one of the suction cups considered. For example, the value of each pixel of the first channel represents the estimation of the success likelihood of a grasp attempt with the first gripper at the location of this pixel.

#### 4.3.2 Self-supervised grasp selection

The output of the network is composed of one segmentation map  $\widehat{S}_{seg}$  and one grasp quality estimation map  $\widehat{S}_g$ . From these two images, one must extract only one position to actually send the gripper to. To do so, we first multiply each channel of  $\widehat{S}_g$ , *i.e.* the success estimation for each gripper, with the segmentation map. As both  $\widehat{S}_{seg}$  and  $\widehat{S}_g$  have values between 0 and 1, the result of this multiplication is a new image,  $\widehat{S}_{final}$ , of the same shape as  $\widehat{S}_g$ , also with values between 0 and 1. Figure 4.15 illustrates this process. Then, a pixel is sampled from this resulting image, with a probability depending on its value: the closer to 1 its value is, the more often it is selected. Using this approach rather than selecting the pixel with the highest value creates some variability in the data generation: even if a pixel is predicted with a small score because such configuration has never been seen before, it has a small chance of being picked, therefore expanding the dataset with this new situation.

Once a pixel is selected, it has to be transposed into a full suction cup grasp  $\{g_x, g_y, g_z, g_{nx}, g_{ny}, g_{nz}\}$ .  $g_x$ ,  $g_y$ , and  $g_z$  are determined by both the projection of the selected pixel in the world space and its value in the depth image. The three orientation parameters  $\{g_{nx}, g_{ny}, g_{nz}\}$  are determined by the normal at the considered pixel. In the simulation, the normal can directly be taken from the rendered normal image. However, such information is often not available for real setups. In

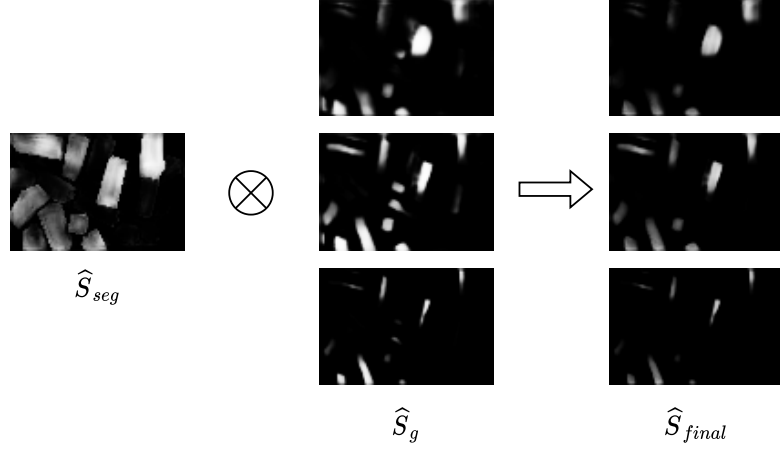


Figure 4.15: Example of multiplication of the two outputs  $\hat{S}_{seg}$  and  $\hat{S}_g$  from our network to obtain a segmentation and grasp success combined estimation. Pixels with a value close to 1 in the combined image must be part of both a top object and at good grasp locations.

this case, the normal is instead estimated from the depth image using the following equation:

$$n_{x,y} = \text{normalize} \left( -\frac{d_{x+1,y} - d_{x-1,y}}{2}, -\frac{d_{x,y+1} - d_{x,y-1}}{2}, 1 \right) \quad (4.3)$$

where  $d_{x,y}$  (respectively  $n_{x,y}$ ) represents the depth value (respectively the normal) at pixel  $(x, y)$ .

#### 4.3.2.1 Network training

The training of our architecture is made through two different losses: one for the graspable segmentation,  $\mathcal{L}_{seg}$ , and one for the grasp success estimation  $\mathcal{L}_g$ :

$$\mathcal{L} = w_{seg} \mathcal{L}_{seg} + \mathcal{L}_g \quad (4.4)$$

where  $w_{seg}$  is a weight to balance the importance of the segmentation loss compared to the grasp quality estimation loss. In practice, we set  $w_{seg}$  to 0.05 in all our trainings.

Both the losses are based on cross-entropy, defined as:

$$CE(p, q) = p \log(q) + (1 - p) \log(1 - q) \quad (4.5)$$

**GRASPABLE SEGMENTATION LOSS** For the graspable segmentation loss, the ground truth is a full image  $S_{seg}$  of the same size as the input image  $H \times W$ , in which the value of a pixel is set to either 0 or 1. At each scale level of the network, the output  $\hat{S}_{seg}^{(i)}$  is also an image of size

$H \times W$ , taking values between 0 and 1. The final output of the decoder  $\widehat{S}_{seg}$  shares the same format. The graspable segmentation loss is thus:

$$\mathcal{L}_{seg} = -\frac{1}{HW} \sum_{x=1}^W \sum_{y=1}^H \left( CE \left( S_{seg_{x,y}}, \widehat{S}_{seg_{x,y}} \right) + \sum_{i=1}^6 w_{seg}^{(i)} CE \left( S_{seg_{x,y}}, \widehat{S}_{seg_{x,y}}^{(i)} \right) \right) \quad (4.6)$$

where  $w_{seg}^{(i)}$  is a weight associated to the output at the  $i^{th}$  scale level.

This weighted sum of cross-entropy losses ensures that the inference of the network is consistent with the ground-truth at each level, and for each pixel in the image. In practice, all the  $w_{seg}^{(i)}$  are often set to 1, avoiding a laborious hyperparameter search.

**GRASP SUCCESS ESTIMATION LOSS** Contrary to the graspable segmentation loss, the ground truth for the grasp success estimation loss is not a full segmented image, but rather the reward obtained when attempting to grasp an object at one given position, with a given tool. Formally, let  $(x, y)$  be the pixel coordinates of the grasp attempt,  $t$  the index of the tool used for this grasp, and  $r \in \{0, 1\}$  the outcome of the grasp attempt (0 if the grasp was a failure, 1 if it succeeded). The grasp success estimation loss is then:

$$\mathcal{L}_g = - \left( CE \left( r, \widehat{S}_{g_{t,x,y}} \right) + \sum_{i=1}^6 w_g^{(i)} CE \left( r, \widehat{S}_{g_{t,x,y}}^{(i)} \right) \right) \quad (4.7)$$

where  $w_g^{(i)}$  is a weight associated to the output at the  $i^{th}$  scale level. Just like for the graspable segmentation, all the  $w_g^{(i)}$  are often set to 1 in practice.

#### 4.4 EXPERIMENTS

The goal of this section is to explore the abilities of our self-supervised object-aware grasp detection neural network. Specifically, we want to answer the following questions:

1. What is the performance of our approach, and how does it perform compared to other methods?
2. What is the influence of each part of our architecture on the final performance of the method?

We first detail the procedure we use to train and evaluate the network in [Section 4.4.1](#). Then, [Section 4.4.2](#) and [Section 4.4.3](#) present the experiments we made to answer (1). Finally, analysis to answer (2) are performed in [Section 4.4.4](#).



#### 4.4.1 Training details

##### 4.4.1.1 Images used

Our network is a Convolutional Neural Network, which means that it takes images as inputs.

**IMAGE NATURE** Depth sensors are now very common and almost as cheap as RGB cameras. Depth information is also required to transform a grasp from pixel coordinates to world coordinates. Moreover, in an industrial context, processed objects can sometimes have no texture information at all, as can be seen on [Figure 4.16](#). In this case, a depth image carries more information than an RGB one. For these reasons, we have decided to use only the depth information as input for our training. The input of the network is then a single channel  $H \times W$  image, with values between 0 and 1 representing the distance to the camera: 0 means very close to it, and 1 very far.



Figure 4.16: Example of a pile of textureless objects. An RGB image does not contain a lot of information to work on for this kind of industrial parts.

**IMAGE RESOLUTION** Our network follows an encoder/decoder architecture, with the output images having the same resolution as the input one. Thus, the size of the input image is a crucial factor for two reasons:

- it has a significant impact on the inference time and the memory usage of the network
- it directly controls the distance in world coordinates between two pixels in the output map: as the size of the bin does not change, the more pixels there are in the output image, the less

distance there is between two of them, and the more precise the grasp position can be determined.

Table 4.1 presents different characteristics of the network regarding the chosen input resolution. Given these numbers and the fact our smallest suction cup has a radius of 1 cm, the chosen input resolution for our training is 64x96. While 128x192 seems to be a better choice, training with this resolution is 25% slower than with 64x96. As a full training session lasts almost two days due to our deep backbone, these 25% represents 12 additional hours of training. 64x96 is therefore a good trade-off between precision in world coordinates and training time. The memory usage for a batch of 8 is also low, thanks to the architecture of the RSU blocks, making our network compatible with lower-ends GPUs with less memory, even for training.

Input image resolution	Training iteration time (s)	Training memory usage (Gb)	Output resolution in world coordinates (cm)
32x48	0.41	2.1	1.25
64x96	0.40	2.6	0.63
128x192	0.50	4.9	0.31
192x288	0.81	9.1	0.21

Table 4.1: Characteristics of the network for different input resolutions. Time is measured for one full forward, backward and weights update iteration on a GTX 1080Ti. For both the time and the memory usage, a batch of size 8 is considered.

#### 4.4.1.2 Optimizer details

The weights of the network are updated using the AdamW optimizer [69], with a learning rate of  $1e^{-4}$ . We do not use any pre-trained weights, all layers are initialized using the Kaiming method [39] and are trained from scratch for 40 epochs. An epoch is defined as one complete training step over the whole replay memory buffer. The replay buffer is composed of 90000 data samples, that are sent in batches of size 8 to the network. Before the training starts, the replay memory is filled up by gathering data using uniform sampling over all the pixels of all the considered tools. At the end of each epoch, the oldest 10000 data in the replay memory are replaced with the newest ones, gathered during the training of the previous epoch.

#### 4.4.1.3 Grippers

For our training, we considered a simulated robot with three different suction cups, with three different radii  $R$  in  $\{1, 2, 3\}$  cm. The strength of their simulated vacuum is defined by Equation 4.2. Figure 4.17

shows the size of these suction cups compared to a typical pile of objects in our simulation environment. The output of the network is therefore a  $H \times W \times 3$  image, the  $i^{\text{th}}$  channel being associated with the corresponding  $i^{\text{th}}$  radius size. For extension of the network to parallel-plate gripper, see [Section 4.5](#).



Figure 4.17: The three suction cups used to train the network in the context of a typical object pile in our simulation environment.

#### 4.4.1.4 *Evaluation metrics*

For all compared methods, the evaluation procedure is the same. We use the same simulation environment with different objects than the one used at training. Then, we use the currently studied method to sample grasps on a succession of a large number of images (depending on the method, but a typical order of magnitude is tens of thousands of grasps, but less for humans). Evaluating the process on such large sets of images allows us to see many situations, from full bins with many objects to almost empty bins with only a few instances. The grasp is executed in the simulation environment, under the exact same conditions as during data generation for training: when the gripper successfully grasps the object and carries it away to a drop position, it is considered a success. The final score of the method is thus the percentage of such successful grasps among all the attempts.

#### 4.4.2 *Quantitative analysis*

As the performance in the simulation environment can not be directly linked to a performance in a real-world application due to the reality gap, we compare our method to multiple other approaches inside our

environment. The same set of objects is used for all methods requiring a training step. All approaches are also evaluated using the same set of objects, but different from the ones used from training.

#### 4.4.2.1 *Testing grasp selection*

At evaluation time, we use a method similar to the one presented in [Section 4.3.2](#) to select a grasp from a grasp quality estimation map. In [108], the authors used an argmax approach: the grasp with the highest estimated quality is selected to be performed. This approach can be dangerous in bin-picking tasks: if this predicted grasp does not modify the image (because the predicted location fails to grasp the object for example), the picking task enters a blocked state, from which it can not escape, as the same grasp will be predicted over and over.

To solve this problem, we instead use a thresholded version of the grasp selection method presented in [Section 4.3.2](#). Given a grasp quality estimation map, we first set all the values inferior to  $T$  times the maximum value to 0. Then, the final grasp is sampled according to their relative values: a higher value has more chances to be selected than a lower value. The overall operation results in the selection of a high-quality grasp, but with some variation, preventing the process to fall in an infinite loop. In our experiments, we set  $T$  to 0.9. A study of the influence of  $T$  on the performance can be found in [Section 4.4.4](#).

#### 4.4.2.2 *Compared methods*

**RANDOM SELECTION** The first baseline we compared our method to is a random selection based one. To get a grasp location from the image, we simply sample a pixel over the output image of size  $H \times W \times 3$  with a uniform probability distribution. While the performance of a random grasp planner can not be expected to be very high, it is still relevant information to have, as it defines a lower bound that any other algorithm should exceed.

**INDUSTRIAL BASELINE** In addition to the random comparison, we also compare our approach to an analytic approach used in the industry. Developed at Siléane, and used in bin-picking applications, it is based on image analysis techniques, searching for potential areas in the image where a suction cup can successfully grasp an object. An example of detections can be seen on [Figure 4.18](#). As the approach is only based on geometric criteria, we add a post-processing step to remove all grasps that are predicted on non-object parts of the image (for example on the borders or the bottom of the box containing the objects). It is not a data-based approach, and it has not been developed to succeed in the simulation environment. Therefore, its performances in the simulation do not necessarily reflect how good it is when used

on real data due to the reality gap, but it is an interesting performance to compare our method to.

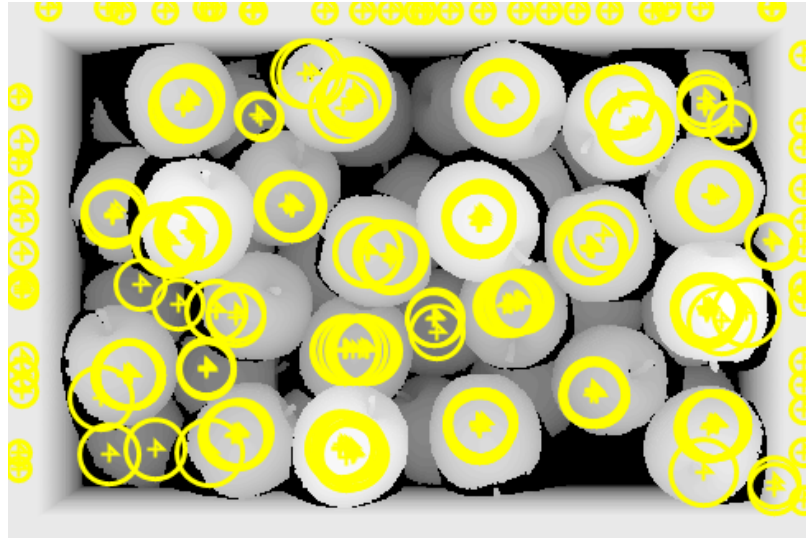


Figure 4.18: Detection results of the industrial baseline on an image from our simulation environment. Detections on the border of the box containing the objects are removed in a post-processing step.

**HUMAN CLICK AND PICK** To get an idea of how difficult the task in the simulation environment is, we also evaluate human performance. To do so, a variation of the simulation environment has been developed. Instead of sending the images to a learning component, it displays it to a human operator, asking him to select a suction cup and click on one pixel to send the gripper to. Because  $64 \times 96$  is quite a small image for human eyes, the image is upscaled to a  $640 \times 960$  resolution using a nearest-neighbour interpolation, *i.e.* without adding new information. An example of our human interface can be seen on [Figure 4.19](#). As it is difficult to ask a human to perform several thousand clicks while staying focused on the task, we instead asked a dozen different people to play with this simulation environment. Some of these people were familiar with bin-picking, while others did not have any knowledge about it. However, they were all explained their task in the same way before the testing session begins.

**OTHER LEARNING METHOD** To get a performance comparison with another data-driven approach, we trained the recent GG-CNN2 network [81]. This network is also a CNN, with a grasp quality estimation head. As our simulation environment has three suction cups, we duplicated the last convolution filter two times to get three grasp quality estimation maps, one for each of our suction cups. As the authors provide weights pre-trained on depth images, we used them as starting point. Then, trained the network on images from our simulation environment in two ways:

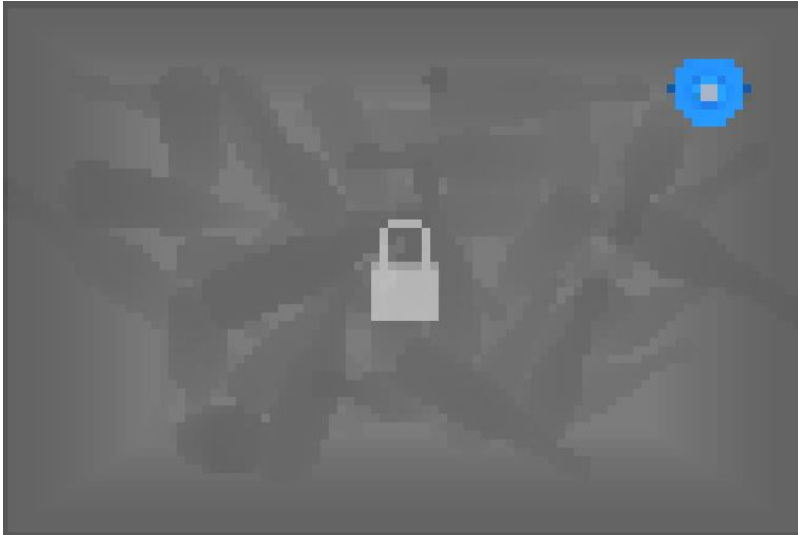


Figure 4.19: Human interface for our click and pick simulation. A lock is displayed as long as one grasp is being performed. When it is removed, the human operator knows he has to click on a pixel to perform the next grasp. When switching the selected suction cup, a preview of the one that will be used is displayed for a short time in the top right corner.

- by following the official code provided with the paper <sup>1</sup>, *i.e.* in a supervised way, using pre-generated images from our simulation environment.
- by following our self-supervised approach, *i.e.* by progressively generating data using the partially trained network during the training.

At test time, we used the same approach as for our network to sample a grasp from the predicted grasp quality map, as explained in Section 4.4.2.1. For both ways, we independently trained the network 5 times, and the reported performance is the average of all the training.

#### 4.4.2.3 Performance comparison

Table 4.2 reports the results of the methods presented in Section 4.4.2.2. For each method, we measured the success grasping rate, *i.e.* the number of predicted grasp that indeed picked an object divided by the total amount of attempted grasp, and the inference time, *i.e.* the time the method takes to analyze the image and select one grasp location. As all the grasps are performed in the same simulated environment, the execution time of the grasp by the robot is not considered here: it is the same for all methods. All reported times are measured for one image on the same Windows 10 computer, with an Intel I7-7700 CPU and an Nvidia GTX 1080 Ti GPU.

<sup>1</sup> <https://github.com/dougsm/ggcnn>



Method	Success Rate	Inference Time (ms)
Random	3.25%	0.003
Industrial Baseline	16.10%	128
Human (average)	50.98% $\pm$ 6.07	1200
Human (best)	59.44%	860
GG-CNN2 [81] (supervised)	22.46% $\pm$ 6.05	5.2
GG-CNN2 [81] (self-supervised)	43.40% $\pm$ 0.47	5.2
Ours	62.77% $\pm$ 0.21	78.1

Table 4.2: Performances and inference time of multiple grasp detection methods.

The first thing that can be said about the overall results is that the gap between simulated physics and reality is important. We could indeed expect humans to have a much better performance than 50.98% (even if they are not experts in bin-picking), and the industrial baseline to be at least at 80% instead of 16.10%.

This gap can be explained by two main reasons: first, all our objects, including the container box, are not deformable, and second, the joint between the grasped object and the gripper is a strict position constraint. These two facts combined mean that, whenever the grasped object collides with the box, *i.e.* when the displacement done during one simulation time-step makes its volume intersecting the container one, an infinite force has to be applied to maintain the grasp position constraint. Obviously, this infinite force breaks the thresholds defined by Equation 4.2, resulting in a failed grasp attempt. A more detailed description of this behaviour is available in Appendix B. In real-life, objects and suction cups joints are not so rigid and allow some compliance in case of such small collisions. The simulation environment is therefore more challenging than real-life. Performances of different methods inside the simulation can nevertheless be compared to each other, to see how good they perform in these adversarial conditions.

By comparing the performances of the two training of the GG-CNN2, we can directly see the advantage of the self-supervised approach compared to the pre-generated data used in a classical supervised way: the successful grasp rate almost doubles, from 22.46% to 43.40%. This is due to the fact that the more the network is trained, the more it

can generate data bringing more information than the pre-generated ones. For example, once the network has learned that sending the gripper on a pixel belonging to the containing box always fails, keeping many data samples of this kind is not useful, as they don't bring new information. However, replacing them with new adversarial examples, where the network made a mistake predicting a success, helps it to correct the weights to prevent similar mistakes in the future.

From the last two lines of the table, we can also see that our deeper object-aware architecture performs better than the shallower GG-CNN2 one, with an average success rate of 62.77% against only 43.40%. This shows that thanks to its more complex structure, our deeper architecture can extract more precise information, and thus to better estimate the grasp outcome at each position. Obviously, having a deeper neural network, with more layers, has a drawback: the inference time is higher, *i.e.* it takes more time to get a grasp position from the sensor image. It takes 78.1ms for our full architecture with the graspable segmentation decoder, more than 15 times slower than the 5.2ms of the GG-CNN2. While this prevents our network to be used at very high frame-rates, it can be used in industrial bin-picking applications, as it is still almost 40% faster than the currently used industrial baseline.

Another interesting fact to note is that our full architecture outperforms the best human in our test set, with a success rate of 62.77% against 59.44%. This means that our network has indeed learned to consider not only the shape of the surface at the proposed grasping point but also its surroundings and the dynamics involved in the extraction of the object afterward, like a human would do.

#### 4.4.2.4 Segmentation performances

Even if we do not explicitly use the output of the first decoder to create a binary segmentation of the input, we can still artificially create such an image to evaluate the segmentation performance. To do so, we apply a binarization threshold to  $\hat{S}_{seg}$ : pixels with predicted values below the threshold are assigned to 0, while the others are set to 1. The result is a binary image that can be compared to its ground-truth counterparts  $S_{seg}$ . By varying this threshold between 0 and 1, we can have a good understanding of the network's behaviour when presented images of the unknown objects.

**EVALUATION METRICS** We compute two indicators measuring the performance of our graspable segmentation decoder:

- the IoU, or intersection over union, representing how well the predicted segmentation instances overlap the ground-truth ones.
- the precision and recall. Precision indicates how good is the model to return only instances that are indeed indicated as



graspable in the ground-truth, while recall represents the ability of the network to detect all graspable instances.

The way we compute the graspable segmentation ground-truth is quite strict: a single occluded pixel of an instance can switch it all from graspable to non-graspable. A very small change in the input depth image can thus cause a very large modification of the ground-truth image, making it hard for the network to generate a prediction for lightly occluded instances. To measure the impact of this, we generate variations of the ground truth, that we also compare to the prediction. Specifically, we create ground-truth images where more instances are defined as graspable by adding a tolerance factor on the occlusion of the objects. For example, with a 10% tolerance, objects that are less than 10% occluded will still be considered graspable. Details on how we estimate the occlusion of an instance can be found in [Appendix B](#).

**RESULTS** Results obtained with a threshold varying between 0 and 1 can be found on [Figure 4.20](#). As can be seen on the IoU curve with 0% tolerance, the network prediction does not match the ground-truth perfectly, with only a top IoU of 0.487 achieved for a threshold value of 0.296. However, we can see that part of the mismatch is due to how our graspable segmentation ground-truth is defined. By increasing the occlusion tolerance, we can see that the IoU of the same network increases, with for example a peak IoU of 0.637 when considering objects that are only 10% occluded. This confirms that the network tends to predict more graspable instances than there actually is in the ground-truth.

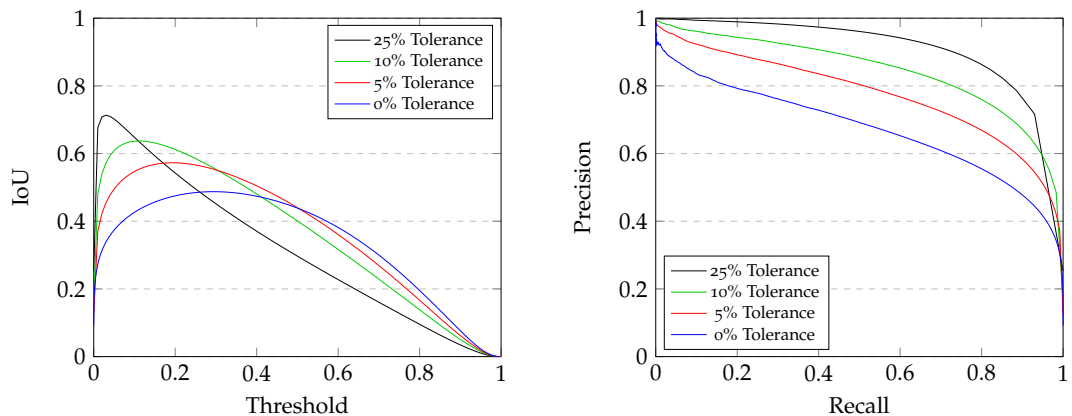


Figure 4.20: Plot of the IoU (left) and the precision-recall curve (right) of our network for various binarisation thresholds, and multiple level of tolerance in the objects occlusion level for the ground-truth.

This phenomenon is also visible on the precision-recall curves. With a 0% occlusion tolerance, the precision decreases quite rapidly when the recall increases, meaning that more instances are misclassified as graspable. However, the higher the tolerance is, the slower is this

decrease in precision. With a 10% occlusion tolerance on the ground-truth, the network predicts 80% of the pixels set to 1 in the ground-truth with a precision of 0.728.

Even if it tends to predict more instances than the ones in the ground-truth, the graspable segmentation decoder of our network is still capable of predicting useful segmentation information. In fact, predicting more instances can sometimes be an advantage in a bin-picking context. With many objects, all of them may be at least slightly occluded. In this case, the ground-truth is, per definition, fully black. However, it is preferable that our network predicts non-zero segmentation values, as we could not be able to select a grasp otherwise.

#### 4.4.3 Qualitative analysis

Figure 4.21 shows the visual outputs obtained from the same input depth image for three different networks:

- our full architecture, with its graspable segmentation decoder
- our architecture deprived of the graspable segmentation decoder
- the smaller GG-CNN2 network

All networks have been trained using our self-supervised learning approach, with the same simulation environment. Multiple things can be inferred from these images.

First, we can see that the output of the graspable segmentation (the sixth column) matches the desired ground-truth pretty well. Almost all the pixels of the instances marked as not occluded, *i.e.* set to 1 in the ground-truth, have values close to 1 in  $\hat{S}_{seg}$ . This means that the network has integrated the notion of non-occluded object, and is able to correctly infer it, even for new objects with similar, but different, shapes. Due to how the graspable segmentation ground-truth is defined, a minor change in the depth image, for example the occlusion of only one pixel of an object, can cause a large change in the ground-truth, setting all the pixels to 0 instead of 1. For this reason, we can also see some additional objects, not set to graspable in the ground-truth, but predicted with higher values in  $\hat{S}_{seg}$ . These additional segmented objects are however not a real problem, as they are still some lightly occluded objects and thus good candidates for grasping.

Through these images, one can also understand the difference in performances between the networks using the U<sup>2</sup>-Net backbone, and the smaller GG-CNN2. The GG-CNN2 network does not have enough layers to extract a good representation of a grasping location. There is almost no difference in the predictions for the smallest and the medium suction cups, revealing it couldn't learn how different they

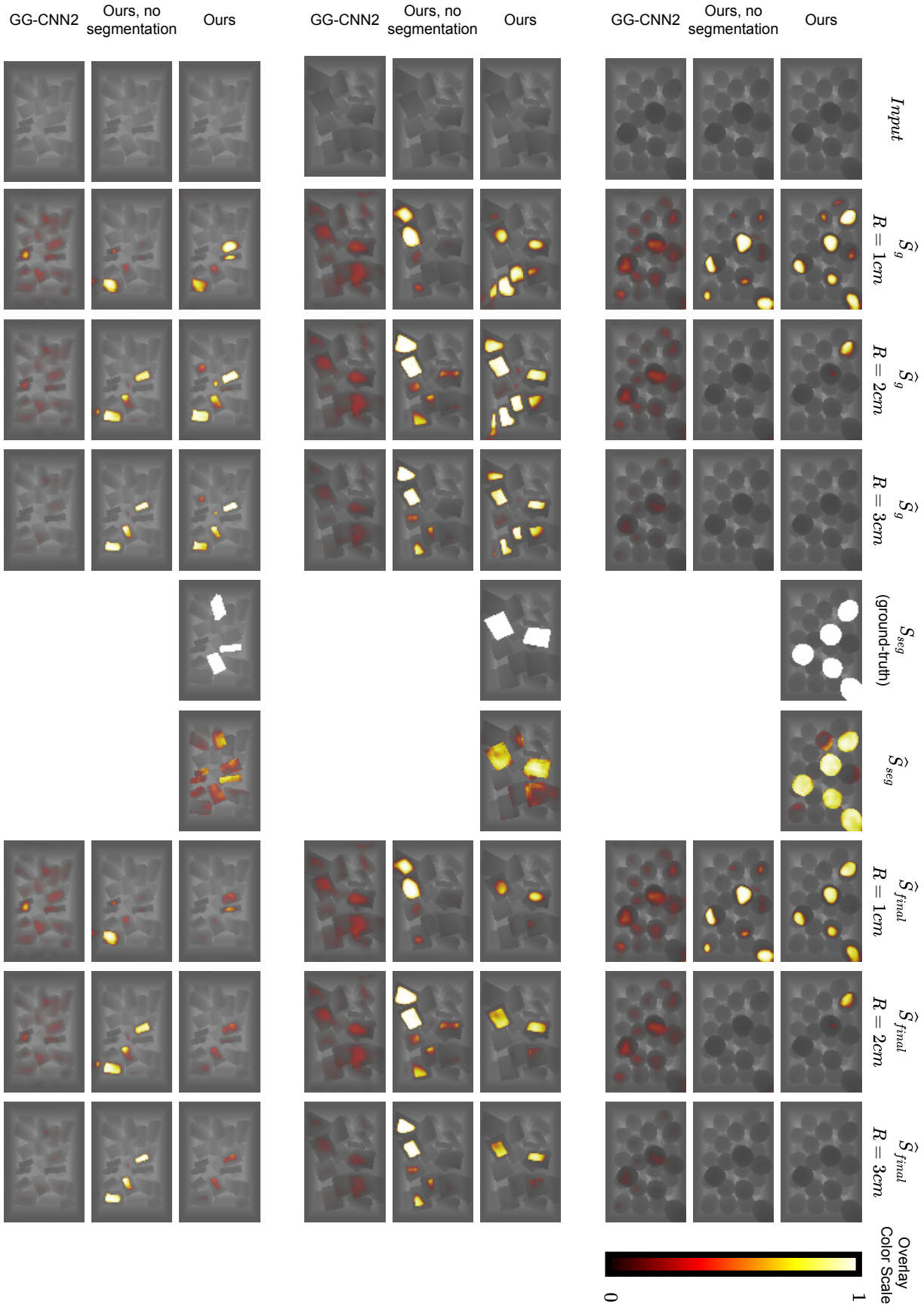


Figure 4.21: Examples of input and outputs of three architectures: our full network, our network deprived of its segmentation decoder, and the smaller GG-CNN2 network. For the networks without segmentation, grasp outputs are copied under the final prediction for easier comparison, i.e.  $\hat{S}_{final} = \hat{S}_g$ .

behave when used for grasping in the simulation environment. Moreover, the grasp quality estimations generally have lower values, with a maximum value under 0.5, showing that the network does not have a very high level of confidence that proposed locations are indeed correct ones.

On the other hand, the U<sup>2</sup>-Net based architectures can output specific quality maps for each of the suction cups. This is particularly visible for the second input image, for which each suction cup radius has different grasping opportunities: the smallest can't grasp objects that are too occluded, while the largest one can, but has to be located farther from the borders of the objects because of its size. This is also visible with the spherical objects: due to the size of the grippers and the curvature of the object, medium and large suction cups can't fit the surface and thus fail to fit the air leak criterion for a successful grasp, as described in [Section 4.2.3.2](#).

By comparing  $\hat{S}_{final}$  for the first two lines for each image, one can understand how the use of the graspable segmentation changes the behaviour of the network, pushing it to output higher grasp quality on non-occluded objects. Graspable segmentation does not only forces the network to focus on non-occluded objects but also helps it to see grasp opportunities that are otherwise ignored. This is visible on the last image, with a very thin box placed vertically. In this configuration, the box is very easily graspable, as it can't be occluded (otherwise it would fall on its side due to the weight of the occluding box), and as the extraction will be performed along the vertical, meaning no chance of collision with the containing box, or other objects. However, only the smallest suction cup is able to perform such a grasp, as the other ones are too wide to fit on the thin side of the box. In this specific case, the network trained with graspable segmentation has indeed understood that case, while the one without the segmentation decoder failed to predict a good grasp quality on such locations.

#### 4.4.4 *Parameters influence*

Our network architecture is composed of several components. In this section, we evaluate the influence of different parameters on the final success rate of the network.

##### 4.4.4.1 *Ablation studies*

To determine which parts of our proposed architecture are crucial for grasp quality estimation, we perform multiple training of our network deprived of some parts. In details, we studied the influence of four different parts of the architecture: the use of the multi-scale loss, the presence or absence of the segmentation decoder during the training phase, its use or not during the testing phase (*i.e.* whether  $\hat{S}_{final}$  or  $\hat{S}_g$  is used to select the grasp), and the influence of the cascaded

decoders architecture over the same architecture without link between the decoders. For each test, we trained each network 5 times and get the average performance between the 5 runs. Results are presented in [Table 4.3](#).

Segmentation decoder during training	Segmentation decoder during testing	Segmentation decoder linked to grasp decoder	Multi-scale loss	Performance
✗	✗	✗	✗	60.79% ± 1.47
✗	✗	✗	✓	60.62% ± 0.87
✓	✗	✗	✓	41.39% ± 0.98
✓	✓	✗	✓	57.03% ± 0.65
✓	✗	✓	✓	60.38% ± 0.60
✓	✓	✓	✓	62.77% ± 0.21

Table 4.3: Performances of our architecture deprived of some of its components. Each reported performance is the average and standard deviation between 5 independent training with the same configuration.

**MULTI-SCALE LOSS** To see the influence of the multi-scale loss, we trained two networks with all  $w_g^{(i)}$  set to 0 for  $i \in [1..6]$  to remove the multi-scale loss, and set to 1 to use it. These two results are in lines 1 and 2 of [Table 4.3](#). As we can see, the multi-scale loss does not directly improve the performance of the network, as the network using a multi-scale loss has a performance of 60.62%, very similar to the one without which successfully grasps an object in 60.79% of the attempts. However, the presence of the multi-scale loss brings stability in the training, reducing the variability in the performances, with a reduced standard deviation of 0.87 instead of 1.47.

**SEGMENTATION DECODER DURING TRAINING** Simply adding the graspable segmentation task to the network does not help the network to learn better grasp quality estimation. As we can see comparing the second and third lines of [Table 4.3](#), this leads to a large drop in performance, from 60.62% for the model without the graspable segmentation task, to only 41.39% for the network with two independent decoders for graspable segmentation and grasp quality estimation. This drop in performance is not very surprising: the network is asked to perform a new task that is not directly linked to the previous one.

**SEGMENTATION DECODER DURING TESTING** If the graspable segmentation decoder is present in the architecture, it can be used at inference for the final prediction, as presented in [Section 4.3.2](#). Be-

tween lines 3 and 4, the trained architecture is the same. The only difference is that in the first case,  $\hat{S}_g$  is directly used to sample a grasp, while in the second,  $\hat{S}_g$  is multiplied by  $\hat{S}_{seg}$  before the grasp is selected from the quality estimation. Using the graspable segmentation at inference has two main effects: first, it improves the overall performance, from 41.39% to 57.03%, and second, it decreases the variability in trained networks, from 0.98 to 0.65. Having a low variability is crucial, as it avoids having to train a network multiple times to get a good one.

**LINKED CASCADED DECODERS** The last parameter we tested in our ablation studies is the connection between the graspable segmentation and the grasp quality estimation decoders. The third and fifth lines of [Table 4.3](#) show the performance of the same architecture with only one difference: in the network of the fifth line, the grasp quality estimation decoder does not only have the encoder outputs as inputs, but also the outputs from the segmentation decoder. This link strongly connects the two tasks for the network, increasing the final performance, from 41.39% to 60.38%.

These ablation studies show that the presence of the graspable segmentation task in the network is not sufficient to improve the grasp performance. This segmentation task must be linked to the grasp quality estimation task both in the architecture and in the final inference to get our best performance. Moreover, the presence of this segmentation task helps the trainings to be consistent, with a very small variability in the network’s performances. This approach can thus be used in real-life applications without having to train the same network multiple times to get a good performance.

#### 4.4.4.2 Inference threshold influence

Our network outputs only a grasp quality estimation map for each possible location. To select a grasp from this map, we use an approach with a threshold  $T$ , as described in [Section 4.4.2.1](#).  $T$  can take any value between 0 and 1, 0 being the grasp selection method used at training, *i.e.* a random selection weighted by the values of the network. When  $T$  equals 1, the selection becomes equivalent to an argmax selection.

[Figure 4.22](#) presents the grasp success rate of our trained network for multiple values of  $T$ . As expected, the grasp success rate increases with the value of  $T$ : locations with lower predicted quality are discarded, and better locations are therefore selected more often. However, a too high value can reduce the success rate due to the deadlock that can occur when the image is not changed after a failed grasp attempt. For these reasons, we selected an intermediate value of 0.9 for  $T$ . It allows the network to achieve peak performance, without being totally

blocked by letting the possibility to select any location with a predicted grasp quality close to the maximum value.

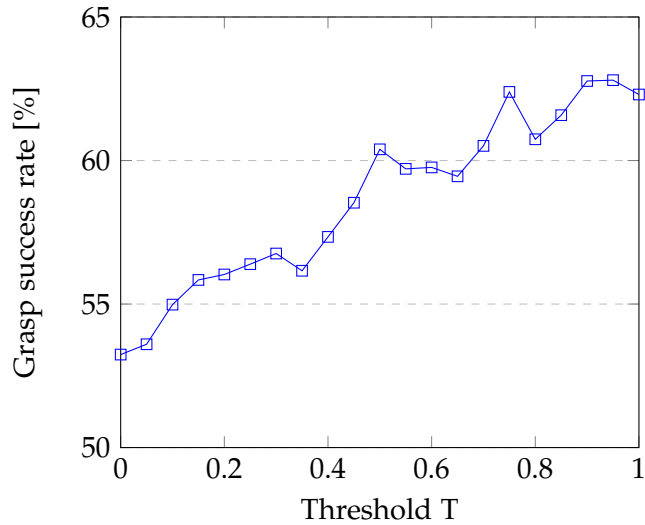


Figure 4.22: Plot of the grasp success rate of our network versus the value of the threshold  $T$  applied to the estimated grasp quality map.

#### 4.5 ADAPTATION TO NEW SITUATIONS

The goal of this section is to answer the following question: **how does our network adapt when faced with new situations?** To answer this question, we conducted two sets of experiments: one using modified versions of our simulation environment, and one using a real robotic setup. In both cases, the objective is to see the performance of the network when faced with situations it was not trained on, and how it can be efficiently adapted to them.

##### 4.5.1 Modified simulation

The easiest way to test the adaptability of our network is by modifying the simulation environment, and see how quickly it can adapt to these new conditions.

###### 4.5.1.1 Simulation modifications

We tested four different types of modifications inside our simulated world: the object that has to be grasped, the data available for training, the image returned by the simulated depth sensor, and the nature of the gripper end-effector.

**CHALLENGING OBJECT** For our adaptation experiments, we wanted an object different from the ones seen in training. To do so, we have



carefully chosen a model previously seen in [Chapter 3](#) that has properties presenting a challenge in our simulated context. Specifically, this object is a bottle model as shown on [Figure 4.23](#). We scaled this object so that around 50 of them could fit inside our simulation box. The final simulated object is then 25 cm high, and approximately 6.4 cm wide, and is challenging for multiple reasons:

- Compared to a more spherical or cubic shaped object, this model, with its elongated shape, is more prone to be partially occluded by multiple other instances when piled up
- the thinner head part of the bottle makes it harder to grasp: grippers with larger radius do not have a surface large enough to create a successful grasp, while smaller suction cups positioned at this location would fail due to the torque being created by the distance to the centre of mass
- due to the air leak criterion described in [Section 4.2.3.2](#), the cylindrical shape and its curvature impose a small suction cup radius. Given the scale of the object in the simulation environment, any suction cup with a radius over 1.5 cm would thus have no possibility at all to grasp this object



Figure 4.23: Bottle model used for our network adaptation experiments. Its shape and size have been carefully selected to ensure a challenging task in the simulation environment.

**DATA AVAILABLE FOR TRAINING** Unlike in simulated environments, it is very difficult to get a segmentation ground-truth in real-life. Even if our graspable segmentation is much easier and quicker to create for a human than a full segmentation, it is still a limiting factor, preventing the fine-tuning procedure to be fully automatized. Therefore, we also performed our fine-tuning procedure using only the grasp outcome. In details, we set  $w_{seg}$  to 0 in [Equation 4.4](#). In



addition, as we want to preserve the segmentation abilities of the network, all the layers of the encoder and the first decoder are frozen, and thus not updated. Only the weights of the last decoder, estimating grasp quality, are updated with the gradient from the grasp loss  $\mathcal{L}_g$ .

**NOISY SENSOR** In real-life applications, depth sensors do not return perfect images, but noisy ones. Depending on the technology used, this noise can have various profiles, but they always have one thing in common: missing data. For some pixels, the depth sensor is not able to determine the distance to the camera. In this situation, the pixel is often filled in with a predetermined value (either 0 or 1 for example). This placeholder value can be troubling for a network that did not see any of such images during its training.

To simulate this phenomenon, we added a post-processing step to our image capture pipeline. This step uses the normal map to replace the value of some pixels with 1. To get a noise effect similar to what can be seen on real depth images, these pixels are selected on the angle between their normal and the vertical. Thus, the depth information of all pixels with a normal forming an angle of more than  $60^\circ$  is discarded. Examples of this process can be seen on [Figure 4.24](#). While not perfect, this quite simple process is sufficient to evaluate the behaviour of the network when adapted to images with missing data points.

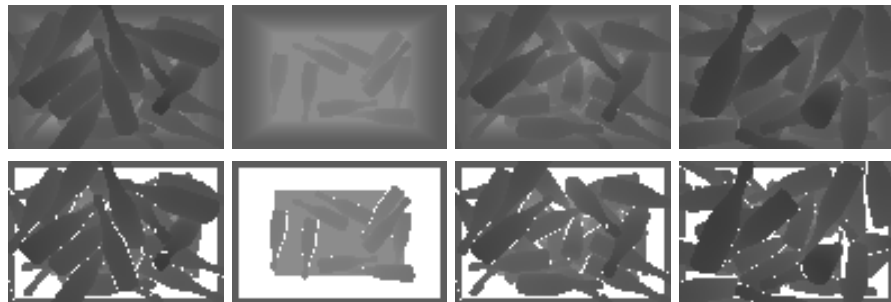


Figure 4.24: Comparison between perfect depth images (top row) and, noisy depth images, after applying our noise post-processing effect. The depth value of all pixels having a normal too far from the vertical is discarded and replaced with 1 instead.

**CHANGE IN SUCTION CUPS RADIUS** Our base neural network is trained for suction cups with a radius of size 1, 2, and 3 cm. It is unlikely that all the applications it will be used for would use the exact same set of grippers. So, to evaluate if our network can quickly learn to adapt to new grippers, we changed the suction cups to have a radius of 0.5, 1, and 1.5 cm, in this order.

**PARALLEL-PLATE GRIPPER** A grasp with a parallel-plate gripper is not defined by a radius, but rather by an opening, and an orientation.

To adapt our architecture to this new type of gripper, we followed the approach of [108], and discretised the orientation space into 6 possible orientations:  $0^\circ$ ,  $-30^\circ$ ,  $30^\circ$ ,  $-60^\circ$ ,  $60^\circ$  and  $90^\circ$ . The opening of the gripper is constant, and set to 9 cm, thus leading to 6 possible grasps at each location. As we are dealing with homogeneous piles of objects without too complicated shapes, this quite coarse set of possible grasps is sufficient.

To adapt to this grasp representation, only the top layers of the grasp quality estimation decoder have to be changed, from 3 channels representing the radius to 6, one for each orientation. These modified layers have to be initialized before the fine-tuning process begins. Using the weights trained with the suction cups could be an option, as a good location for a vacuum-based gripper could also be potentially good for a parallel-plate gripper. However, doing so would focus the network attention on suction cups grasps during the data generation process. That's why we instead reinitialized the replaced layers using the Kaiming method [39], as if they were trained from scratch.

#### 4.5.1.2 *Fine-tuning procedure*

The ultimate goal of our method is to be applied to real-life problems rather than simulation ones. We thus adapted our self-supervised training procedure to add constraints consistent with real problems.

The main issue when working with real robotic setups is data gathering time. Usually, only one robotic cell is available, and a whole grasping cycle (including data acquisition, data processing, and grasp planning and execution) can take several seconds. As consequence, we removed all the parallelization inside the simulation environment: only one pile of objects is being simulated at any time. In addition, we also limited the number of data that can be acquired at each epoch. Considering an average of 10 seconds for a full grasping cycle as a typical value, we set the number of data samples gathered for each epoch at 120, *i.e.* the equivalent of 20 minutes of real-time. The full length of the fine-tuning procedure is 72 epochs or the equivalent of one full day of self-supervised data collection for a real robot.

Unlike the full training procedure, in which the network is evaluated only at the end, and on unseen objects, we evaluate the performance of the network at the end of each epoch, on new images of the object it is currently fine-tuned for. In this way, it is also possible to determine how quickly the network adapts to the new conditions. This is crucial information, as in some industrial contexts, the production should not be stopped for training more than a few hours, and thus the full 72 epochs fine-tuning is not feasible in totality.

#### 4.5.1.3 Adaptation results

Due to how they are simulated in different ways, the grasping rates with suction cups and with a parallel-plate gripper can't be directly compared. Thus, we present the results from their respective fine-tuning experiments separately.

**SUCTION CUPS** Figure 4.25 presents the evolution of the performance of our network during the fine-tuning procedure for four situations, with the challenging new object. One of these situations is our unmodified simulation environment. This will act as a baseline for the bottle object, allowing us to compare the performances achieved by the network when presented with an altered version of the environment.

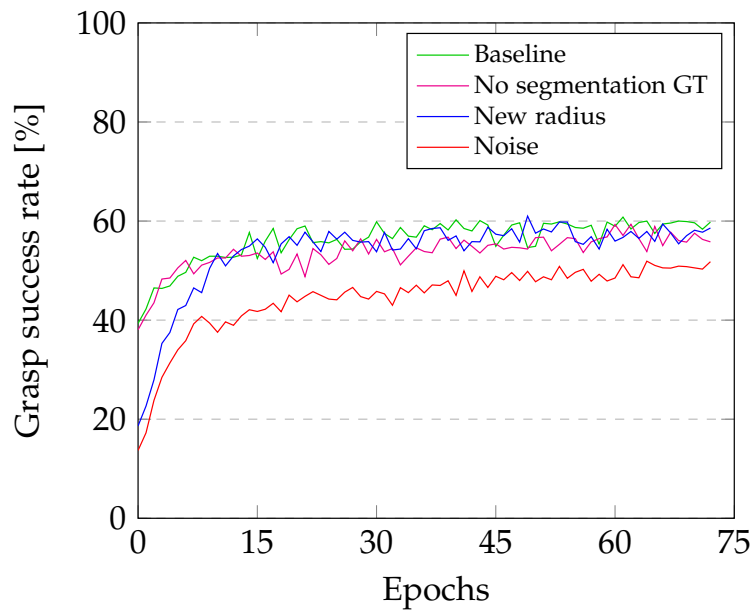


Figure 4.25: Evolution of the grasp accuracy of our network during the fine-tuning process for different variations of the simulation environment.

From the curve of the baseline, we can already see that our base network does not directly generalize well to this particular new object, with a grasping rate of only 39.25% before the fine-tuning. However, the network is able to quickly adapt to this new shape, reaching performance of 55% in only 15 epochs. 15 epochs in our fine-tuning procedure are the equivalent of only 5 hours of automatized trials by a robot, or only 1800 grasp attempts. The performance plateau around 59% is reached in 30 epochs, or 10 hours. This quick adaptation makes this approach viable in the industry: the learning of a new reference can be automatized, and quickly performed when the robot is not in production (for example during one night).

When the segmentation ground-truth is not available during the fine-tuning operation, the training curve still follows the same path, with a small drop in the final reached performance, from 59% to 57.55%. This means that the segmentation decoder is quite able to generalize to new objects, even without fine-tuning. However, the final grasp decoder requires an adaptation step to increase the overall performance from 39.15% to 57.55%.

Modifications of the grippers causes, as one could expect, a massive drop in the initial performance, from 39.25% for the baseline to 18.65%. However, this does not change the peak performance the network can reach, nor the number of grasp attempts needed to reach this peak performance. This shows that even if our base network is trained using some radius for the gripper, it can still be used on applications with different end-effector configurations, without any loss of performance after fine-tuning (assuming the new configuration allows to grasp the considered object).

When the input data distribution is changed, for example by adding pixels with missing data, the performance also drastically drops. When applied on noisy inputs, our base network trained on perfect depth images has a grasping rate of only 13.7%. By applying our fine-tuning process, the performance rapidly increases, reaching 37.55% after only 10 epochs (or 1200 grasp attempts). It then continues to grow slowly to 51.3% at the end of the 72 epochs. This 7.7 points gap with the baseline can be explained in part by the introduction of the noise: the network is asked to extract the same information, but with missing pixels, *i.e.* from fewer input data. The number of epochs necessary to reach a top performance is also larger: as the input data distribution is modified, not only the top convolution layers have to be adapted, but also the ones inside the encoder, to learn a better internal representation ignoring the missing data.

**PARALLEL-PLATE GRIPPER** Results of the fine-tuning of the base network to a network predicting the grasp quality for the 6 orientations of a parallel-plate gripper are presented on [Figure 4.26](#). As we can see, even with randomly initialized top layers for the grasp decoder, the network still has a performance of 23.9% before the fine-tuning begins. Due to the segmentation decoder being trained, initial grasps are indeed not uniformly distributed over the whole image, but rather only over the graspable objects.

As long as the fine-tuning process runs, the performance of the network increases, until it reaches a plateau around 73% after 40 epochs. Fine-tuning for a completely new kind of gripper is thus possible. However, reaching the top performance requires more time and trials than when keeping the same gripper, but with slightly modified parameters.

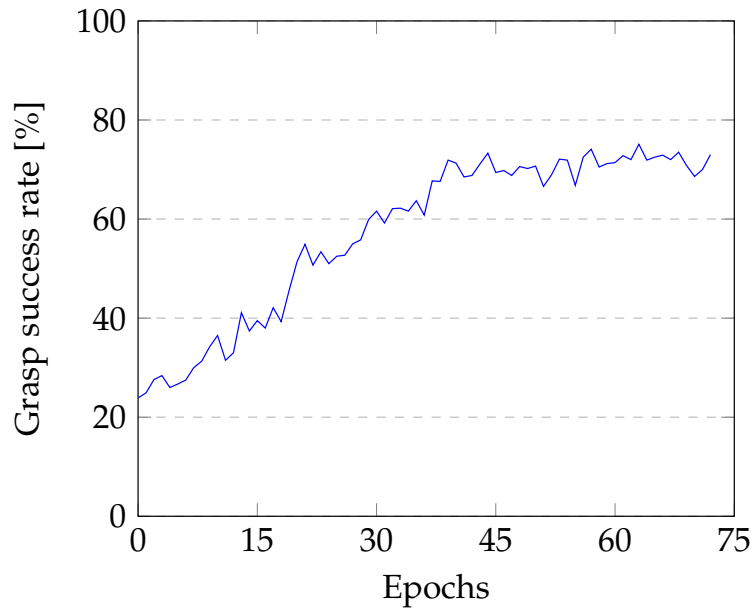


Figure 4.26: Evolution of the grasp accuracy of our network during the fine-tuning process when suction cups are replaced by a parallel-plate gripper with 6 possible orientations.

#### 4.5.2 Transfer to real-life applications

As explained in previous sections, our simulation environment does not perfectly match real conditions. To evaluate the reality gap and see to what extent learning on simulated data can improve predictions on real data, we tested our approach on a real robotic environment.

##### 4.5.2.1 Testing details

To evaluate our network, we used a Fanuc robotic arm, equipped with a vacuum-based suction cup. Data acquisition is performed using a depth sensor from Photoneo, located above the grasping area. A photo of our complete real setup can be seen on [Figure 4.27](#). Just like in the simulation, the objects are placed in a rectangular bin, of similar size as the one in our simulated environment. Using this setup, we tested our network on two different tasks, also illustrated on [Figure 4.27](#).

The first task is a semi-organized object pile: many objects, identical in shape, are placed in the box, in a regular pattern. We used egg boxes, as they are easy to acquire in large quantities, light enough to be grasped by the suction cup, and have an interesting shape. Grasping an egg box is quite easy: if the gripper is on the top flat part of the box, the grasp is almost guaranteed to be a success. However, the more complex parts on the sides and back of the box make it impossible to successfully grasp it in any other location. Boxes then have to be extracted from the pile in order, with non-occluded first. Otherwise, occluding boxes could overturn and become impossible to grasp.

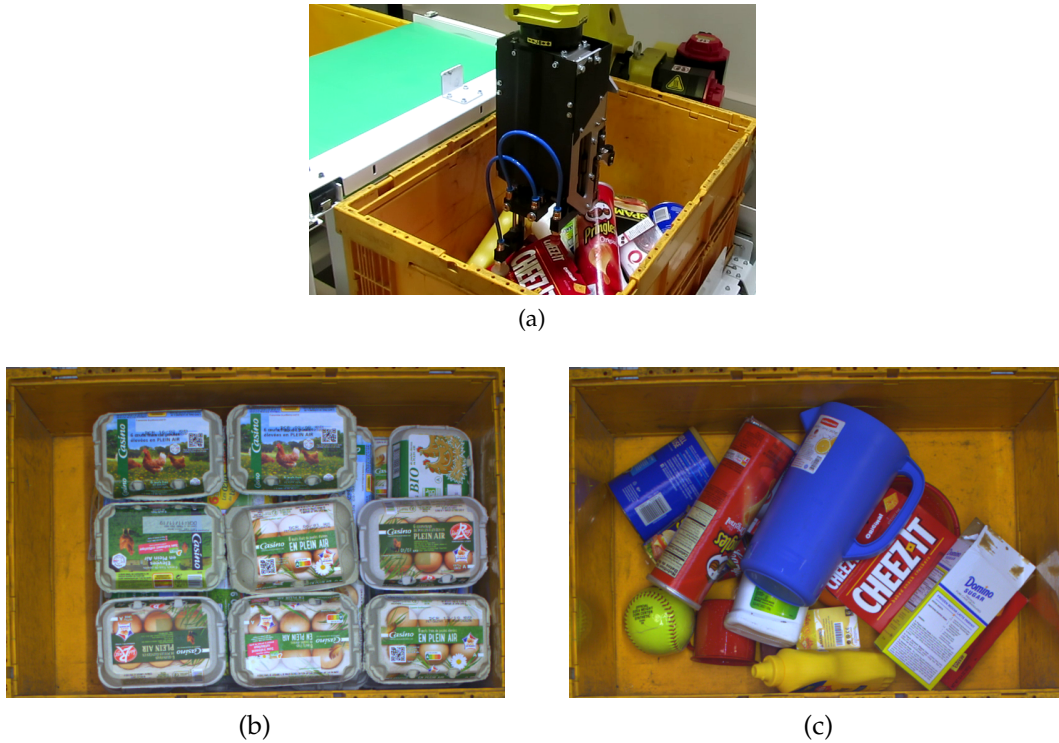


Figure 4.27: Images of our real robotic setup. (a) The global setup, the robotic arm controls a vacuum-based suction cup to grasp objects in the orange bin. The depth sensor is located above the grasping area, pointing downwards. (b) Example of RGB image for our easier semi-organized bin-picking task. (c) Example of RGB image for the more complex heterogeneous bin-picking problem.

The second task is a more common bin-picking application. Using objects from the YCB dataset [15]. Objects are randomly placed inside the bin, creating a heterogeneous pile with various shapes and scales. This task is much more difficult, as not all surfaces are suitable for a suction cup, and occlusion rates are much higher than in a semi-organized pile. Moreover, as our network has only seen homogeneous piles during training, this task is a real challenge, and thus a suitable one to evaluate the generalization abilities of our approach.

For both tasks, we evaluated two networks: our full architecture, with the graspable segmentation decoder and the grasp quality one, and the same architecture trained without the segmentation decoder, *i.e.* a simple encoder-decoder architecture with a U<sup>2</sup>-Net backbone. This will allow us to measure the impact of the graspable segmentation decoder in real-cases. Both networks are fully trained on the same objects, as described in Section 4.4.1. Then, a fine-tuning process is applied, still in simulation with the same objects, but with some artificial noise added to the input depth, as explained in Section 4.5.1.1. This fine-tuning step with noise is crucial, as otherwise, the network would have to deal with the holes in the real data, without having seen some during its training. The robotic system only uses one suction



	<b>Ours (no segmentation)</b>	<b>Ours (with segmentation)</b>
<b>Egg boxes</b>	97.4%	97.7%
<b>YCB objects</b>	89.7%	92.3%

Table 4.4: Performance of our network, with and without the segmentation decoder, on the two presented real tasks.

cup radius, we thus merge the three predictions in one map by taking for each pixel the maximum predicted value between the three radii.

#### 4.5.2.2 Real applications results

Table 4.4 presents the accuracy of our network, with and without the graspable segmentation decoder, on the two tasks presented in the previous section.

**EGG BOXES TASK** As expected, both networks have a very good performance on the simple egg boxes problem, with an average of only one failed grasp attempt for each session of 40 objects grasped, resulting in a grasping rate of around 97.5%. For most of these failed attempts, the reason is that the suction cup is sent above an edge instead of the centre of the object, preventing a good vacuum to be formed. Besides these few failed attempts, the networks are always able to correctly predict a good location for the suction cup, as we can see on Figure 4.28. However, the adaptation is not complete. While the grasp predictions are correct and most of the time centered on the objects, the segmentation has some objects only partially labeled. This is not an issue for the overall performance, as long as at least one object is correctly segmented as graspable.

**HETEROGENEOUS BIN-PICKING** The problem of heterogeneous bin-picking is more complex than the egg boxes one, especially for a network trained only on homogeneous images generated in simulation. Our network still achieves good performances, with a 92.3% success rate for the version with the segmentation decoder, and 89.7% for the version without. This shows once more that adding the segmentation decoder helps the network to focus on more interesting instances.

Due to the nature of the images, most of the time, only one instance is non-occluded. This is confirmed by the predictions of the segmentation decoder, as can be seen on Figure 4.29.  $S_g$  and  $S_{final}$  are very close, the multiplication with the segmentation only resulting in removing grasp predictions on other objects, with more occlusions. This is visible in the first column of the network with segmentation: the SPAM can is predicted with a high grasp quality score, but not segmented as a graspable object, and thus the quality estimation on

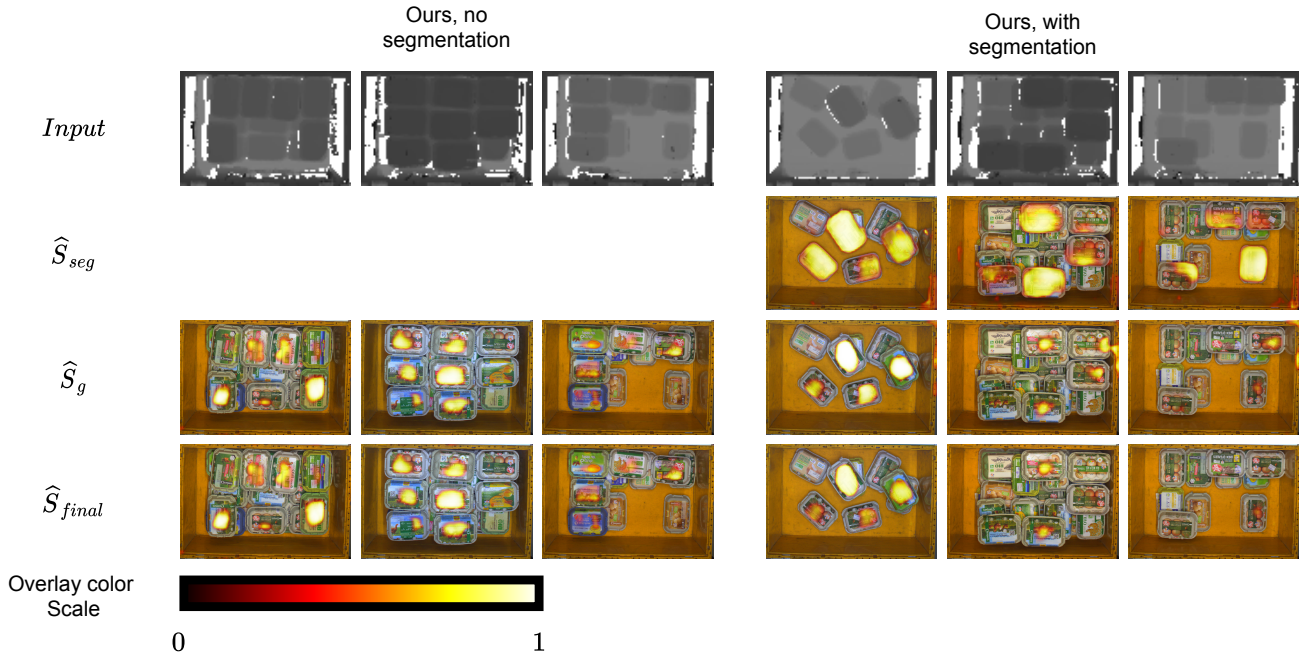


Figure 4.28: Outputs of our network for multiple egg boxes configurations.  $\hat{S}_{final} = \hat{S}_g$  when there is no segmentation prediction from the network. Results are displayed over the RGB image for better visualisation, however, the network only uses the depth as input.

this object is lowered in the final prediction  $\hat{S}_{final}$ . The effect of the segmentation decoder is also visible when comparing the prediction for the water pot: the object-aware network is able to understand the shape of this large object much better than the other, trained without object instance notions. This results in a grasp quality higher on the centre of the object, compared to a grasp quality higher on the border of the object.

Figure 4.30 shows some closer look on some grasp attempted by the robotic setup. Most of the failed cases can be categorized into three categories:

- badly considering two close objects as one and thus predicting a bad grasp on the junction between the two
- predicting a grasp on a surface that is not vertical enough, resulting in a grasp failing when the object is extracted
- ignoring an object that is in the corner of the container. As these objects are very difficult to grasp in the simulation due to the collision mechanism, the network often ignores them in real-life. In this kind of situation, like the last image of the second row in Figure 4.30, no grasp with high quality is predicted, and thus a grasp with low estimated quality is selected.



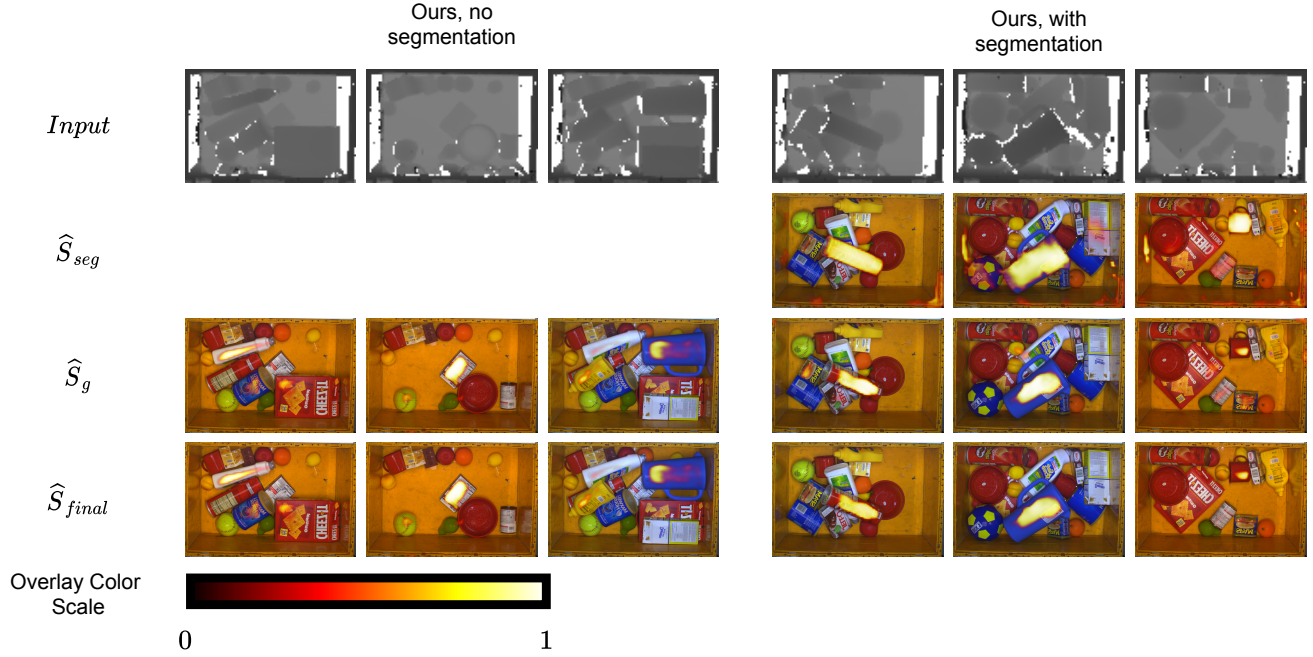


Figure 4.29: Outputs of our network for multiple heterogeneous bin-picking configurations.  $\hat{S}_{final} = \hat{S}_g$  when there is no segmentation prediction from the network. Results are displayed over the RGB image for better visualisation, however, the network only uses the depth as input.

Despite the scarcity of these cases, as evidenced by the overall grasping rate of 92.3%, they could still be problematic for industrial use cases. However, it is important to note that the network was trained only on simulated data of homogeneous piles, quite far from this heterogeneous pile situation, with many objects having unknown shapes. It is thus reasonable to think that the network could perform better when fine-tuned on a few images of such piles.

## 4.6 CONCLUSION

### 4.6.1 Summary

In this chapter, we proposed a novel object-aware fully convolutional neural network architecture for grasp detection in piles. The proposed architecture is composed of one encoder and two decoders, for graspable segmentation and grasp quality estimation respectively. The whole architecture is trained in a self-supervised way, using simulation.

Specifically, we first detailed the self-supervision process used during training, before describing our architecture. It distinguishes itself from state-of-the-art approaches [36] [131] by combining graspable



Figure 4.30: Close look at some grasps predicted by our network. Blue circle is the location of the suction cup. They are displayed over the RGB image for better visualisation. Some allow the network to successfully grasp the object (top row), while others fail at this task (bottom row).

instance segmentation and grasp quality estimation in one network trained end-to-end. We detailed the simulation environment we developed specifically for grasp detection in piles, for both vacuum-based suction cups, and parallel-plate gripper. We then conducted multiple experiments to demonstrate the advantage of using our network with combined prediction, as well as ablation studies to highlight the role of each component of our architecture. Last but not least, we showed that our network trained in simulation can be quickly and successfully adapted to new situations, both in simulation and real-life robotic setups.

#### 4.6.2 Contributions

Unlike state-of-the-art approaches for grasp detection in piles, which rely on either predicting instance segmentation, and then grasp location, or directly grasp quality, jointly predicting them improves the final grasping rate. As a result, **our proposed approach, using instance segmentation to filter out grasp candidates outperforms other approaches, only based on grasp quality estimation.**

Our network can be trained in a **self-supervised way, interacting with our simulated environment, allowing it to learn from more relevant data**, and removing the need of using manual annotations or

heuristics to generate many annotated samples prior to the training. This allows the training of much deeper, and thus more performant, neural networks.

Our experiments and ablation studies showed that both of these contributions are important. The network with the best performance is our deep U<sup>2</sup>-Net based architecture, trained in a self-supervised way, and using both the decoders to output graspable instance segmentation and grasp quality estimation.

As networks trained on synthetic data tend to perform worse on real ones, due to the reality gap, we also conducted a set of experiments to show that **our network is able to quickly adapt to new situations, either in simulation or in real-life, by performing a self-supervised fine-tuning**. This makes our network suitable for real-life industrial applications, such as bin-picking tasks.

# 5

---

## CONCLUSION

---

This chapter concludes our work, and is organized as follows: first, we summarize our work in [Section 5.1](#). Then, we recapitulate our contributions in [Section 5.2](#). Finally, we discuss the research directions that could be explored for future work in [Section 5.3](#).

### 5.1 SUMMARY

In this work, we addressed the problem of model-free robotic grasping from images. In details, the goal is to extract a location from an image where a robotic system can send an end-effector to successfully grasp an object. State-of-the-art approaches are typically trained using either manually annotated real images, or data gathered through robotic trial and error. However, such data gathering methods are not scalable, making it difficult using the full power of very deep neural networks. We thus propose to leverage physics simulation to generate the amount of data required to train such architectures.

In [Chapter 2](#), we reviewed the state-of-the-art for robotic grasping, and more especially for model-free grasping. Grasp detection is usually performed by learning a grasp quality estimator, whose role is to rank grasp candidates by potential success likelihood. The robotic system can then select the highest ranked one, and perform the grasp.

In [Chapter 3](#), we studied the problem of grasping in the case of isolated objects. Specifically, we first described a simulation pipeline developed to generate large-scale annotated dataset from 3D models, as well as our synthetic Jacquard dataset that we created using this method. We also extended a state-of-the-art approach by adding an explicit correlation between the predicted grasp and its estimated quality by the network. Finally, we compared a network using this correlation, and trained on our large data to its state-of-the-art counterpart.

The method proposed in [Chapter 3](#) does not generalize directly to images with many objects, with high rates of occlusion. We presented in [Chapter 4](#) a novel architecture, designed specifically to deal with the grasping problem in such conditions, a problem also referred to as bin-picking. We explained how we used self-supervision to train the network using an interactive real-time simulation environment. We also detailed the object-aware architecture we proposed to use on

images of piles of objects, before analyzing the results of the experiments we made to compare it to other approaches. Finally, we showed that this object-aware network, while trained in simulation, was able to generalize well to new unknown situations, through automatic self-supervised fine-tuning.

## 5.2 CONTRIBUTIONS

Our contribution for model-free robotic grasping is three-fold: **a large-scale synthetic dataset for robotic grasping, an extension of state-of-the-art approaches for isolated object grasping, correlating grasp prediction and quality estimation, and a novel object-aware network that can be trained in a self-supervised way for bin-picking problems.**

Specifically, we proposed in [Chapter 3](#) our large-scale synthetic Jacquard dataset. Being orders of magnitude larger than previously manually annotated dataset, we also showed that, due to its size and diversity, it can improve performance of neural networks on real images, despite being totally synthetic. In addition, the presented method is fully automatized, meaning that, providing some compute power, more data can be easily generated if needed.

Explicitly using the correlation between the grasp prediction and its quality estimation when training a network improves its performance. As demonstrated in [Chapter 3](#), using our scorer module on top of a state-of-the-art grasp detection method increases the performance of a network from 88.1% to 92.4% on a set of real objects.

For bin-picking problems, simulation can also be used to train deep neural networks, as we showed in [Chapter 4](#). Our object-aware network is able to jointly predict graspable instance segmentation and grasp quality estimation. The segmentation helps the network to focus on objects that are easy to grasp, *i.e.* those that are not occluded, while the grasp quality estimation finds local areas suitable for the gripper mounted on the robot. Training this object-aware network is possible in simulation, for which segmentation data are easy to obtain. Despite this simulation only training, we demonstrated that our network can quickly adapt to new problems and situations, either simulated or in real-life.

## 5.3 PERSPECTIVES

In this section, we present some future research directions for the presented work.

**PHYSICS SIMULATION** All the work presented here are based on rigid objects simulation. In real-life, objects are never totally rigid, but rather deformable to some extent. Simulating interactions between

a gripper and a deformable object is possible [129] [107]. Therefore, simulating this deformation could lead to an increased level of realism, reducing the reality gap between the simulation and the real-world. This is particularly true for domains for which objects are very often deformable, such as the food industry.

Simulation could also be extended to get tactile information from the gripper. Simulating tactile sensors is possible [91] [56] and could open the path to add force control learning for a parallel-jaw gripper.

**NEW MODALITIES** Our object-aware model bases its grasp predictions on two information: object segmentation and grasp quality estimation. However, more modalities correlated to the final grasp success could also be used. Examples of such modalities are normals or material estimation. More decoders could be added in our architecture, each used to predict one of these modalities. The final grasp prediction would then be determined based on all these modalities. Material estimation would be useful if, for example, a grasp is physically possible on some parts of the object, but not suitable, because the material on this location is more fragile.

**BETTER TRANSFER FROM SIMULATION TO REAL** After training in simulation, we either directly apply our network on real images, or use a fine-tuning to adapt it. This approach could be improved to reduce the size of the reality gap, and thus facilitate the adaptation. Reducing the reality gap in the images can be performed with GAN trying to match the simulated images and real ones [31] [11]. Another way to reduce the reality gap is called domain randomization [121] [48] [49] [83]: instead of looking for more realism in the simulated data, we try to expand it to be as much varied as possible, hoping that simulated data distributions would eventually include real data ones.

Beside changing the data used by the network, it is also possible to modify the way the network is trained, using meta-learning. Meta-learning aims at replacing a network learning a task with a network learning to learn a task [29] [85]. In our case, a task would for example be an object. The network would thus be presented with multiple objects during the training procedure, and learn to quickly adapt to new ones. Then, when presented with new real objects, the network would be able to adapt its weights in only a few trials.





---

## BIBLIOGRAPHY

---

- [1] Umar Asif, Mohammed Bennamoun, and Ferdous Sohel. "Model-free segmentation and grasp selection of unknown stacked objects." In: *European Conference on Computer Vision*. Springer. 2014, pp. 659–674.
- [2] Umar Asif, Jianbin Tang, and Stefan Herrer. "GraspNet: An Efficient Convolutional Neural Network for Real-time Grasp Detection for Low-powered Devices." In: *IJCAI*. Vol. 7. 2018, pp. 4875–4882.
- [3] Umar Asif, Jianbin Tang, and Stefan Herrer. "Densely supervised grasp detector (DSGD)." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 8085–8093.
- [4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.
- [5] Lars Berscheid, Pascal Meißner, and Torsten Kröger. "Robot learning of shifting objects for grasping in cluttered environments." In: *arXiv preprint arXiv:1907.11035* (2019).
- [6] Lars Berscheid, Thomas Rühr, and Torsten Kröger. "Improving data efficiency of self-supervised learning for robotic grasping." In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 2125–2131.
- [7] Antonio Bicchi and Vijay Kumar. "Robotic grasping and contact: A review." In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 1. IEEE. 2000, pp. 348–353.
- [8] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Blender Institute, Amsterdam. URL: <http://www.blender.org>.
- [9] J. Bohg, A. Morales, T. Asfour, and D. Kragic. "Data-Driven Grasp Synthesis—A Survey." In: *IEEE Transactions on Robotics* 30.2 (2014), pp. 289–309. DOI: [10.1109/TR0.2013.2289018](https://doi.org/10.1109/TR0.2013.2289018).
- [10] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. "Unsupervised pixel-level domain adaptation with generative adversarial networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3722–3731.



- [11] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. "Using simulation and domain adaptation to improve efficiency of deep robotic grasping." In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4243–4250.
- [12] Romain Brégier, Frédéric Devernay, Laetitia Leyrit, and James L Crowley. "Symmetry aware evaluation of 3d object detection and pose estimation in scenes of many parts in bulk." In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 2209–2218.
- [13] Romain Brégier, Frédéric Devernay, Laetitia Leyrit, and James L Crowley. "Defining the pose of any 3d rigid object and an associated distance." In: *International Journal of Computer Vision* 126.6 (2018), pp. 571–596.
- [14] Eric Brown, Nicholas Rodenberg, John Amend, Annan Mozeika, Erik Steltz, Mitchell R Zakin, Hod Lipson, and Heinrich M Jaeger. "Universal robotic gripper based on the jamming of granular material." In: *Proceedings of the National Academy of Sciences* 107.44 (2010), pp. 18809–18814.
- [15] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. "Benchmarking in manipulation research: Using the Yale-CMU-Berkeley object and model set." In: *IEEE Robotics & Automation Magazine* 22.3 (2015), pp. 36–52.
- [16] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. "Shapenet: An information-rich 3d model repository." In: *arXiv preprint arXiv:1512.03012* (2015).
- [17] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation." In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818.
- [18] Fu-Jen Chu, Ruinian Xu, and Patricio A Vela. "Real-world multiobject, multigrasp detection." In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3355–3362.
- [19] *Cornell Grasping Dataset*. <http://pr.cs.cornell.edu/grasping/rectdata/data.php>.
- [20] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. <http://pybullet.org>. 2016–2019.

- [21] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. "Supervised learning." In: *Machine learning techniques for multimedia*. Springer, 2008, pp. 21–49.
- [22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
- [23] Amaury Depierre, Emmanuel Dellandréa, and Liming Chen. "Jacquard: A large scale dataset for robotic grasp detection." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3511–3516.
- [24] Amaury Depierre, Emmanuel Dellandréa, and Liming Chen. "Optimizing correlated graspability score and grasp regression for better grasp prediction." In: *arXiv preprint arXiv:2002.00872* (2020).
- [25] Yukiyasu Domaie, Haruhisa Okuda, Yuichi Taguchi, Kazuhiko Sumi, and Takashi Hirai. "Fast graspability evaluation on single depth maps for bin picking with general grippers." In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1997–2004.
- [26] Zhikai Dong, Sicheng Liu, Tao Zhou, Hui Cheng, Long Zeng, Xingyao Yu, and Houde Liu. "PPR-Net: point-wise pose regression network for instance segmentation and 6d pose estimation in bin-picking scenarios." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 1773–1780.
- [27] Mohamed El-Shamouty, Kilian Kleeberger, Arik Lämmle, and Marco Huber. "Simulation-driven machine learning for robotics and automation." In: *tm-Technisches Messen* 86.11 (2019), pp. 673–684.
- [28] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. "The pascal visual object classes (voc) challenge." In: *International journal of computer vision* 88.2 (2010), pp. 303–338.
- [29] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1126–1135.
- [30] David Flavigné and Véronique Perdereau. "A learning-free method for anthropomorphic grasping." In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 2985–2990.

- [31] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. "Domain-adversarial training of neural networks." In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2096–2030.
- [32] Ross Girshick. "Fast r-cnn." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.
- [33] Corey Goldfeder, Matei Ciocarlie, Hao Dang, and Peter K Allen. "The columbia grasp database." In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 1710–1716.
- [34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.
- [35] Ben Goodrich, Alex Kuefler, and William D Richards. "Depth by Poking: Learning to Estimate Depth from Self-Supervised Grasping." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 10466–10472.
- [36] Matthieu Gard. "Generic instance segmentation for object-oriented bin-picking." PhD thesis. Lyon, 2019.
- [37] Matthieu Gard, Emmanuel Dellandréa, and Liming Chen. "Deep Multicameral Decoding for Localizing Unoccluded Object Instances from a Single RGB Image." In: *International Journal of Computer Vision* (2020), pp. 1–29.
- [38] Di Guo, Fuchun Sun, Huaping Liu, Tao Kong, Bin Fang, and Ning Xi. "A hybrid deep architecture for robotic grasp detection." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1609–1614.
- [39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.
- [40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [41] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Pascal Fua, and Nassir Navab. "Dominant orientation templates for real-time detection of texture-less objects." In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 2010, pp. 2257–2264.

- [42] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. "Multi-modal templates for real-time detection of texture-less objects in heavily cluttered scenes." In: *2011 international conference on computer vision*. IEEE. 2011, pp. 858–865.
- [43] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes." In: *Asian conference on computer vision*. Springer. 2012, pp. 548–562.
- [44] Heiko Hirschmuller. "Stereo processing by semiglobal matching and mutual information." In: *IEEE Transactions on pattern analysis and machine intelligence* 30.2 (2007), pp. 328–341.
- [45] Tomáš Hodaň, Jiří Matas, and Štěpán Obdržálek. "On evaluation of 6D object pose estimation." In: *European Conference on Computer Vision*. Springer. 2016, pp. 606–619.
- [46] Tomáš Hodan, Pavel Haluza, Štěpán Obdržálek, Jiri Matas, Manolis Lourakis, and Xenophon Zabulis. "T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects." In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 880–888.
- [47] *Jacquard Dataset Website*. <https://jacquard.liris.cnrs.fr/>.
- [48] Stephen James, Andrew J Davison, and Edward Johns. "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task." In: *arXiv preprint arXiv:1707.02267* (2017).
- [49] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12627–12637.
- [50] Eric Jang, Coline Devin, Vincent Vanhoucke, and Sergey Levine. "Grasp2vec: Learning object representations from self-supervised grasping." In: *arXiv preprint arXiv:1811.06964* (2018).
- [51] You-Cyuan Jhang et al. *Training a performant object detection ML model on synthetic data using Unity Perception tools*. <https://blogs.unity3d.com/2020/09/17/training-a-performant-object-detection-ml-model-on-synthetic-data-using-unity-perception-tools/>. 2020.
- [52] Yun Jiang, Stephen Moseson, and Ashutosh Saxena. "Efficient grasping from rgb-d images: Learning using a new rectangle representation." In: *2011 IEEE International conference on robotics and automation*. IEEE. 2011, pp. 3304–3311.

- [53] Edward Johns, Stefan Leutenegger, and Andrew J Davison. "Deep learning a grasp function for grasping under gripper pose uncertainty." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4461–4468.
- [54] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. "Unity: A general platform for intelligent agents." In: *arXiv preprint arXiv:1809.02627* (2018).
- [55] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation." In: *arXiv preprint arXiv:1806.10293* (2018).
- [56] Zhanat Kappassov, Juan-Antonio Corrales-Ramon, and Véronique Perdereau. "Simulation of tactile sensing arrays for physical interaction tasks." In: *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE. 2020, pp. 196–201.
- [57] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1521–1529.
- [58] Kilian Kleeberger and Marco F Huber. "Single Shot 6D Object Pose Estimation." In: *arXiv preprint arXiv:2004.12729* (2020).
- [59] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey." In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.
- [60] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.
- [61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Communications of the ACM* 60.6 (2017), pp. 84–90.
- [62] Sulabh Kumra and Christopher Kanan. "Robotic grasp detection using deep convolutional neural networks." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 769–776.
- [63] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), pp. 436–444.
- [64] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps." In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 705–724.

- [65] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "Epn: An accurate o (n) solution to the pnp problem." In: *International journal of computer vision* 81.2 (2009), p. 155.
- [66] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436.
- [67] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *arXiv preprint arXiv:1509.02971* (2015).
- [68] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context." In: *European conference on computer vision*. Springer. 2014, pp. 740–755.
- [69] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization." In: *arXiv preprint arXiv:1711.05101* (2017).
- [70] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models." In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.
- [71] Jeffrey Mahler and Ken Goldberg. "Learning deep policies for robot bin picking by simulating robust grasping sequences." In: *Conference on robot learning*. PMLR. 2017, pp. 515–524.
- [72] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kröger, James Kuffner, and Ken Goldberg. "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards." In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 1957–1964.
- [73] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Mathieu Aubry, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics." In: (2018).
- [74] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. "Dex-Net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

- [75] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. "Learning ambidextrous robot grasping policies." In: *Science Robotics* 4.26 (2019).
- [76] Khaled Mamou, E Lengyel, and AK Peters. "Volumetric hierarchical approximate convex decomposition." In: *Game Engine Gems 3*. AK Peters/CRC Press, 2016, pp. 141–158.
- [77] Andrew T Miller, Steffen Knoop, Henrik I Christensen, and Peter K Allen. "Automatic grasp planning using shape primitives." In: *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*. Vol. 2. IEEE. 2003, pp. 1824–1829.
- [78] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602* (2013).
- [79] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning." In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.
- [80] Douglas Morrison, Peter Corke, and Jürgen Leitner. "Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach." In: *arXiv preprint arXiv:1804.05172* (2018).
- [81] Douglas Morrison, Peter Corke, and Jürgen Leitner. "Learning robust, real-time, reactive robotic grasping." In: *The International Journal of Robotics Research* 39.2-3 (2020), pp. 183–201.
- [82] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. "6-dof graspnet: Variational grasp generation for object manipulation." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 2901–2910.
- [83] Fabio Muratore, Christian Eilers, Michael Gienger, and Jan Peters. "Data-Efficient Domain Randomization With Bayesian Optimization." In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 911–918.
- [84] Anh Nguyen, Dimitrios Kanoulas, Darwin G Caldwell, and Nikos G Tsagarakis. "Detecting object affordances with convolutional neural networks." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 2765–2770.
- [85] Alex Nichol, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms." In: *arXiv preprint arXiv:1803.02999* (2018).

- [86] Matthias Nieuwenhuisen, David Droeschel, Dirk Holz, Jörg Stückler, Alexander Berner, Jun Li, Reinhard Klein, and Sven Behnke. "Mobile bin picking with an anthropomorphic service robot." In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 2327–2334.
- [87] Nvidia Corporation. *PhysX*. Version 4.1. URL: <https://github.com/NVIDIAGameWorks/PhysX>.
- [88] John Oberlin, Maria Meier, Tim Kraska, and Stefanie Tellex. "Acquiring object experiences at scale." In: *AAAI-RSS Special Workshop on the 50th Anniversary of Shakey: The Role of AI to Harmonize Robots and Humans*. 2015.
- [89] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. "Grasp pose detection in point clouds." In: *The International Journal of Robotics Research* 36.13-14 (2017), pp. 1455–1473.
- [90] Maxime Petit, Amaury Depierre, Xiaofang Wang, Emmanuel Dellandréa, and Liming Chen. "Developmental bayesian optimization of black-box with visual similarity-based transfer learning." In: *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE. 2018, pp. 161–168.
- [91] Zachary Pezzementi, Erica Jantho, Lucas Estrade, and Gregory D Hager. "Characterization and simulation of tactile sensors." In: *2010 IEEE Haptics Symposium*. IEEE. 2010, pp. 199–205.
- [92] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours." In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 3406–3413.
- [93] Alberto Pretto, Stefano Tonello, and Emanuele Menegatti. "Flexible 3D localization of planar objects for industrial bin-picking with monocular vision system." In: *2013 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2013, pp. 168–175.
- [94] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space." In: *Advances in neural information processing systems*. 2017, pp. 5099–5108.
- [95] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R Zaiane, and Martin Jagersand. "U2-Net: Going deeper with nested U-structure for salient object detection." In: *Pattern Recognition* 106 (2020), p. 107404.



- [96] Yuzhe Qin, Rui Chen, Hao Zhu, Meng Song, Jing Xu, and Hao Su. "S4g: Amodal single-view single-shot se (3) grasp detection in cluttered scenes." In: *Conference on robot learning*. PMLR. 2020, pp. 53–65.
- [97] Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. "Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6284–6291.
- [98] Joseph Redmon and Anelia Angelova. "Real-time grasp detection using convolutional neural networks." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 1316–1322.
- [99] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.
- [100] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.
- [101] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks." In: *Advances in neural information processing systems*. 2015, pp. 91–99.
- [102] José Jeronimo Rodrigues, Jun-Sik Kim, Makoto Furukawa, Joao Xavier, Pedro Aguiar, and Takeo Kanade. "6D pose estimation of textureless shiny objects using random ferns for bin-picking." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 3334–3341.
- [103] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.
- [104] Cesare Rossi and Sergio Savino. "An underactuated multi-finger grasping device." In: *International Journal of Advanced Robotic Systems* 11.2 (2014), p. 20.
- [105] Reuven Y Rubinfeld and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.
- [106] Anis Sahbani, Sahar El-Khoury, and Philippe Bidaud. "An overview of 3D object grasp synthesis algorithms." In: *Robotics and Autonomous Systems* 60.3 (2012), pp. 326–336.

- [107] Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey." In: *The International Journal of Robotics Research* 37.7 (2018), pp. 688–716.
- [108] Vishal Satish, Jeffrey Mahler, and Ken Goldberg. "On-policy dataset synthesis for learning robot grasping policies using fully convolutional deep networks." In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1357–1364.
- [109] Manolis Savva, Angel X Chang, and Pat Hanrahan. "Semantically-enriched 3D models for common-sense knowledge." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2015, pp. 24–31.
- [110] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. "Robotic grasping of novel objects using vision." In: *The International Journal of Robotics Research* 27.2 (2008), pp. 157–173.
- [111] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017).
- [112] Karun B Shimoga. "Robot grasp synthesis algorithms: A survey." In: *The International Journal of Robotics Research* 15.3 (1996), pp. 230–266.
- [113] Jun Shintake, Vito Cacucciolo, Dario Floreano, and Herbert Shea. "Soft robotic grippers." In: *Advanced Materials* 30.29 (2018), p. 1707035.
- [114] Xin Shu, Chang Liu, Tong Li, Chunkai Wang, and Cheng Chi. "A self-supervised learning manipulator grasping approach based on instance segmentation." In: *IEEE Access* 6 (2018), pp. 65055–65064.
- [115] Shuran Song, Andy Zeng, Johnny Lee, and Thomas Funkhouser. "Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations." In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4978–4985.
- [116] Felix Spenrath and Andreas Pott. "Gripping point determination for bin picking using heuristic search." In: *Procedia CIRP* 62 (2017), pp. 606–611.
- [117] Felix Spenrath and Andreas Pott. "Using Neural Networks for Heuristic Grasp Planning in Random Bin Picking." In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2018, pp. 258–263.

- [118] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. "Implicit 3d orientation learning for 6d object detection from rgb images." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 699–715.
- [119] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [120] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. "Real-time seamless single shot 6d object pose prediction." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 292–301.
- [121] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 23–30.
- [122] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.
- [123] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. "Deep object pose estimation for semantic robotic grasping of household objects." In: *arXiv preprint arXiv:1809.10790* (2018).
- [124] André Ückermann, Robert Haschke, and Helge Ritter. "Real-time 3D segmentation of cluttered scenes for robot grasping." In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE. 2012, pp. 198–203.
- [125] Unity Technologies. *Unity3D*. Version 2020.1. URL: <https://unity.com/>.
- [126] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." In: *arXiv preprint arXiv:1509.06461* (2015).
- [127] Kentaro Wada, Shingo Kitagawa, Kei Okada, and Masayuki Inaba. "Instance Segmentation of Visible and Occluded Regions for Finding and Picking Target from a Pile of Objects." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 2048–2055.
- [128] Yupei Wang, Xin Zhao, and Kaiqi Huang. "Deep crisp boundaries." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3892–3900.

- [129] Lazher Zaidi, Belhassen-Chedli Bouzgarrou, Laurent Sabourin, and Youcef Mezouar. "Interaction modeling in the grasping and manipulation of 3D deformable objects." In: *2015 International Conference on Advanced Robotics (ICAR)*. IEEE. 2015, pp. 504–509.
- [130] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4238–4245.
- [131] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching." In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3750–3757.
- [132] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. "Tossingbot: Learning to throw arbitrary objects with residual physics." In: *IEEE Transactions on Robotics* (2020).
- [133] Xinwen Zhou, Xuguang Lan, Hanbo Zhang, Zhiqiang Tian, Yang Zhang, and Narming Zheng. "Fully convolutional grasp detection network with oriented anchor box." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7223–7230.



# A

---

## ISOLATED OBJECT GRASPING

---

### A.1 BACKGROUND AUGMENTATION

In [Chapter 3](#), we trained our network to detect grasp opportunities on synthetic images of isolated objects. As we want the network to be generic, we augmented the data by replacing the default constant background with a new one. This new background is randomly picked from a list with varied textures, as illustrated by [Figure A.1](#). All the images have been selected to ensure a large diversity in both color and textures.

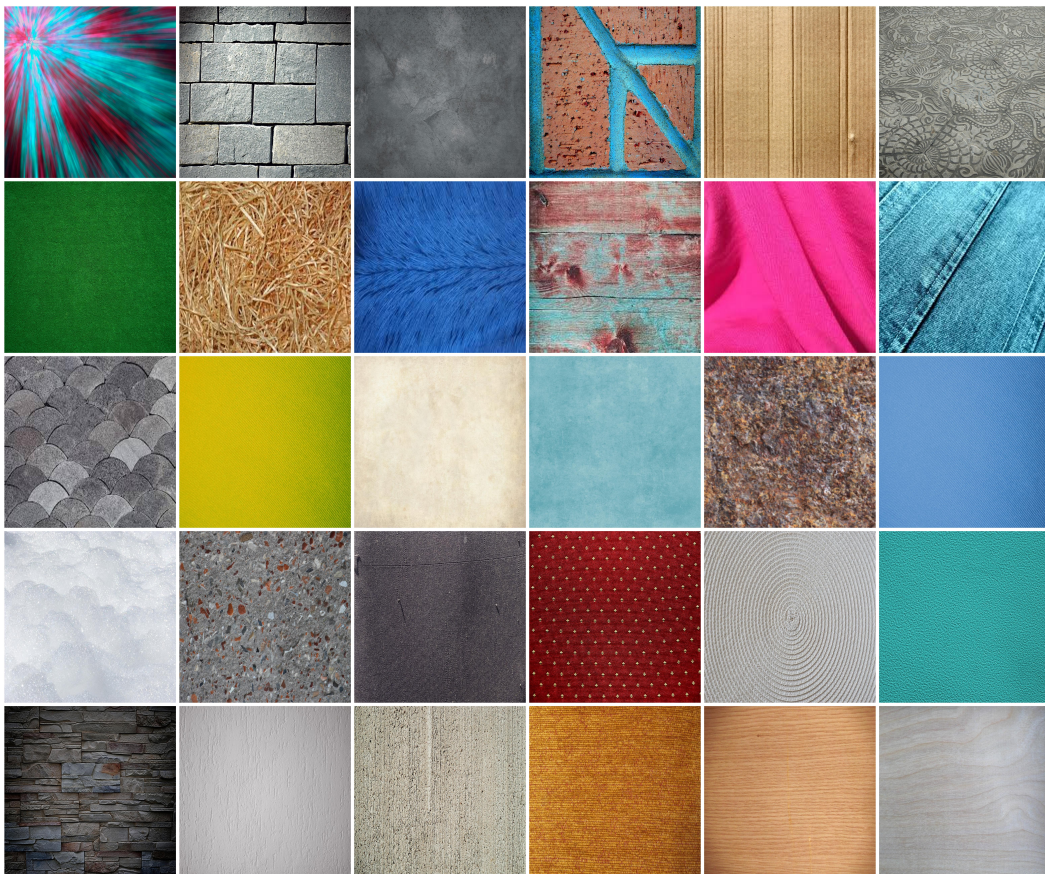


Figure A.1: All background images used during our data augmentation process. Images have been chosen to be very diverse in color and texture.



# B

---

## END-TO-END BIN-PICKING

---

### B.1 RIGID COLLISION SIMULATION

In [Chapter 4](#), we simulate a unmovable box containing many instances of rigid objects. These instances can be picked up by a simulated suction cup. To emulate the vacuum force created by this kind of gripper, a position constraint is created between the gripper and the object being picked up. This constraint creates a force and a torque that maintain the object in place at any time step. If either the required force or torque is over the thresholds defined by [Equation 4.2](#), the joint breaks, and the object is dropped. While this simulation behaviour is quite close to real-life in most cases, it is not when collisions with rigid and unmovable objects happen. Such an example of a collision is illustrated by [Figure B.1](#).

During one simulation time step, the movement of the gripper pushes the object inside the containing box. As the box is unmovable and rigid, the physics engine does not allow the object to penetrate it, and instead it stays in contact with it. However, the link with the gripper has to maintain the position of the object relative to the gripper by creating a force to push it inside the containing box. This infinite force breaks the thresholds defined by [Equation 4.2](#) and thus the object is dropped.

This collision issue does not only happen with the box, but also with other objects. When one object collide with another, there are two possibilities: either the second object can move freely in the desired direction, or it is in contact with an obstacle in that direction. In the first case, the obstacle will be moved by the force of the joint, within the limits of the thresholds. But in the second case, the infinite force created by the obstacle will propagate to the grasped object, breaking the link with the gripper.

When such cases happen in real-life, it does not always lead to a failed grasp. Contrary to simulation, real objects are very rarely fully rigid. In fact, most of the time, the suction cups are made of soft plastics that can easily bend to absorb such small collisions. This compliance makes them much more robust than the one in our simulation.



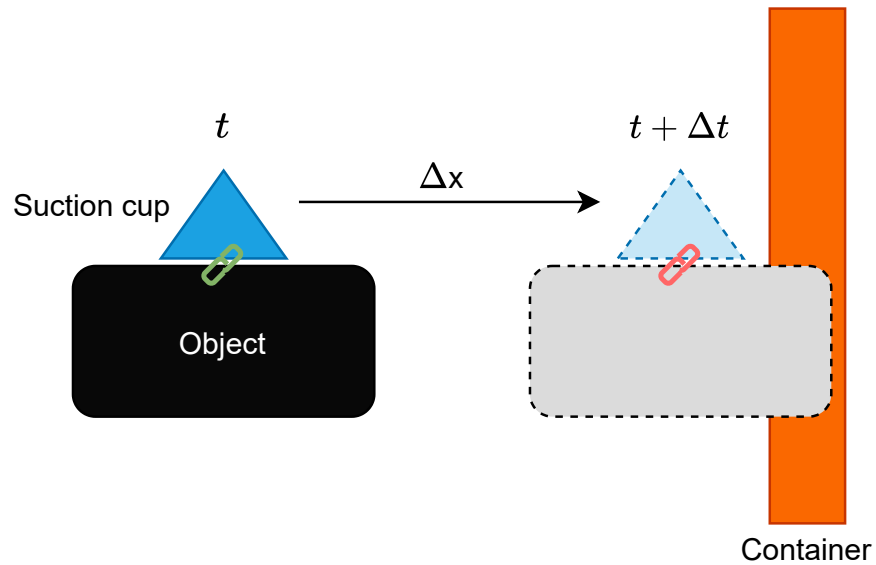


Figure B.1: Collision between an object carried by a simulated suction cup and the containing box. Due to the collision, the link between the object and the gripper is broken.

## B.2 GRASPABLE INSTANCE SEGMENTATION

In [Chapter 4](#), we introduced the concept of graspable segmentation, computed using [Algorithm 2](#). To evaluate the performance of our network, we also used a slightly different definition, with an adaptable tolerance to occlusion. [Algorithm 3](#) presents the modified algorithm used to generate ground-truth with different tolerance levels. As it is not possible to estimate the occluded surface of an object only from the depth image and the full segmentation, we instead compute the number of pixels of the border of the instance that are under or over their neighbours. While the border can not be directly linked to the surface, this is a good indicator of how much an instance is occluded.

The result of this algorithm for multiple values of tolerance is illustrated by [Figure B.2](#). As can be seen, the more tolerant we are with occlusion, the more instances are set as graspable. The border criterion seems to be a good approximation, as the instances that are progressively set to graspable when we increase the tolerance look indeed not too much occluded.

---

**Algorithm 3:** Graspable segmentation with occlusion tolerance algorithm

---

**Input:**  $d$ , depth image  
**Input:**  $s$ , full segmentation image  
**Input:**  $T \in [0, 1]$ , tolerance level  
**Output:**  $S_{seg}$ , graspable segmentation ground-truth image  
 $S_{seg} \leftarrow 0$   
**for** object  $o \in s$  **do**  
   $border\_pixels \leftarrow 0$   
   $occluded\_border\_pixels \leftarrow 0$   
  **for** pixel  $p \in o$  **do**  
     $is\_border \leftarrow \text{false}$   
     $is\_occluded \leftarrow \text{false}$   
     $neighbours \leftarrow$  pixels adjacent to  $p$   
    **for** pixel  $p' \in neighbours$  **do**  
      **if**  $s(p') \neq s(p)$  **then**  
         $is\_border \leftarrow \text{true}$   
        **if**  $d(p') < d(p)$  **and**  $p' \notin background$  **then**  
           $is\_occluded \leftarrow \text{true}$   
        **end if**  
      **end if**  
    **end for**  
    **if**  $is\_border$  **then**  
       $border\_pixels += 1$   
    **end if**  
    **if**  $is\_occluded$  **then**  
       $occluded\_border\_pixels += 1$   
    **end if**  
  **end for**  
  **if**  $\frac{occluded\_border\_pixels}{border\_pixels} \leq T$  **then**  
    **for** pixel  $p \in o$  **do**  
       $S_{seg}(p) \leftarrow 1$   
    **end for**  
  **end if**  
**end for**

---

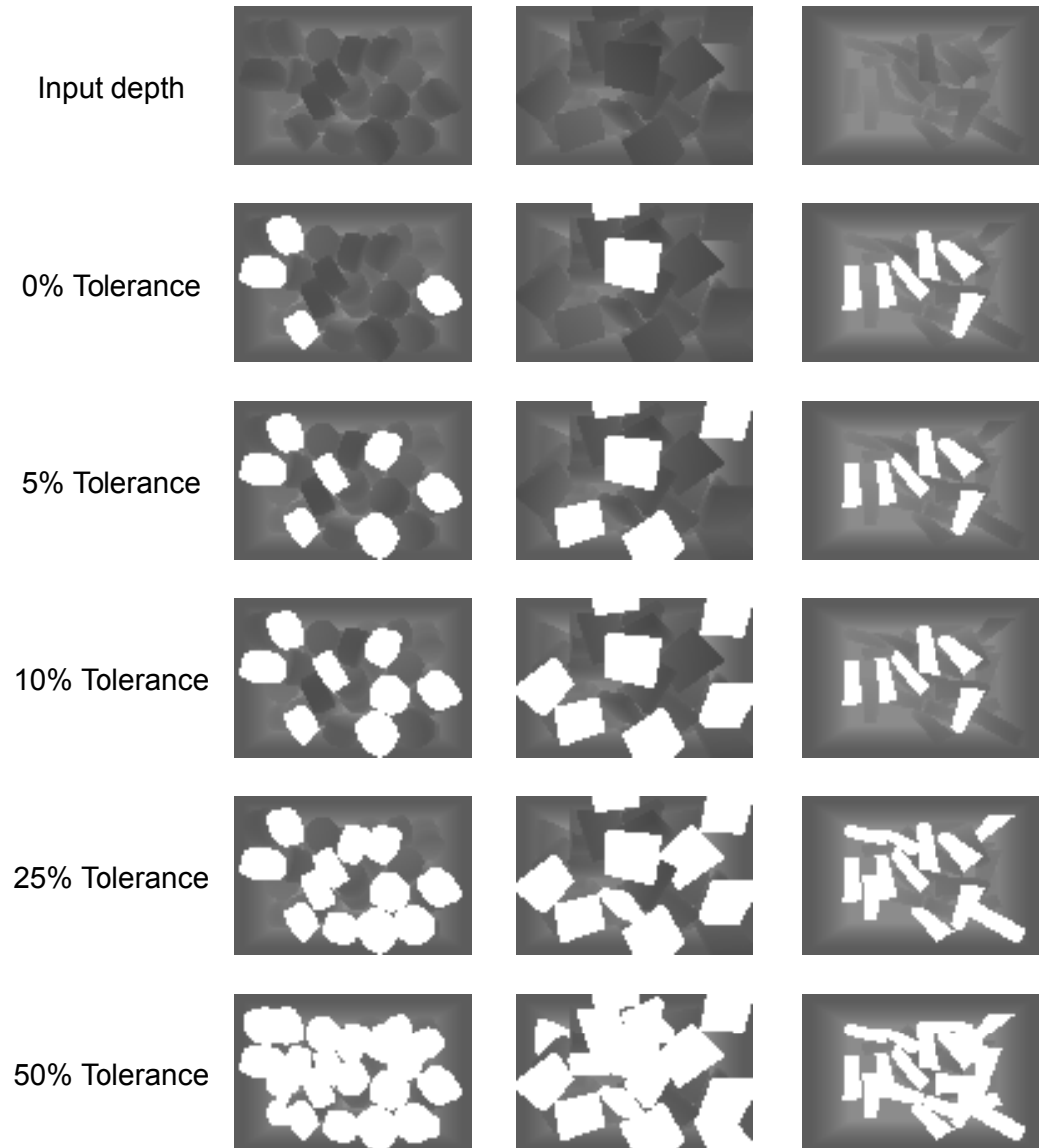


Figure B.2: Examples of graspable segmentation ground-truth images with multiple values of occlusion tolerance. Top images are the input depth. Segmentation is overlaid in white on top of the depth images.

## **AUTORISATION DE SOUTENANCE**

Vu les dispositions de l'arrêté du 25 mai 2016,

Vu la demande du directeur de thèse

Monsieur L. CHEN

et les rapports de

M. J. PETERS

Professeur - TU Darmstadt - FB-Informatik - FG-IAS - Hochschulstr. 10 - 64289 Darmstadt

et de

M. S. CALINON

Directeur de Recherche - Idiap Research Institute - Centre du Parc - Rue Marconi 19  
CH-1920 Martigny - Switzerland

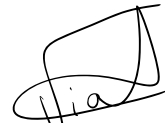
**Monsieur DEPIERRE Amaury**

est autorisé à soutenir une thèse pour l'obtention du grade de **DOCTEUR**

**Ecole doctorale InfoMaths**

Fait à Ecully, le 7 mai 2021

P/Le directeur de l'E.C.L.  
Le directeur des Etudes



Gregory VIAL