POLITECNICO DI TORINO

Master degree in Data Science and Engineering

Master Thesis

Learning2Grasp

An End-To-End Sampling Approach to Robotic Grasping in 6-DoF Pose

Politecnico di Torino

Supervisor
Prof.ssa Tatiana Tommasi

Co-supervisors:
Prof. Matteo Matteucci
Dott. Antonio Alliegro
Dott. Martin Rudorfer

Candidate
Fabio Frattin

December 2021

Learning2Grasp:An End-To-End Sampling Approach to Robotic Grasping in 6-DoF Pose
Master thesis. Politecnico di Torino, Turin.

December 2021.

Abstract

The ability to grasp objects is essential in multiple contexts, ranging from the mimic of human activities to automation of industrial tasks. Recent works based on datasets obtained through simulation have proven to achieve promising performances on unseen objects also in real-case scenarios. Nevertheless, many of them still strongly rely on approaches built on top of ad-hoc geometric heuristics to generate grasp candidates, failing in generalizing to different settings and making it hard to reproduce the same approach in different environments. Moreover, some of the heuristics require different and sometimes independent modules to reduce the redundancy of generated grasps. In this thesis, we propose a lightweight end-to-end solution for the generation of 6-DoF parallel-jaw pose grasps starting from partial view of the object which relies on completely learned sampling techniques, making it easy to translate the same approach to different datasets and settings. We start by introducing a self-supervised pre-trained feature encoder which is able to extract both local and global informations of the input shape in a combined embedding and show how this method outperforms traditional encoders. Moreover, we designed a sampling technique to suit the grasping task. We leverage pre-existing literature on learning to sample to develop a module able to select grasp contact points without imposing geometric custom contraints. This approach makes it possible to better generalize to different object types and shapes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Contextual Overview

Even though computers are basically just machines that executes instructions at a very high speed, some tasks are either too complex or too general that they cannot fit into specific and pre-designed rules. Let's assume we want to create a machine able to recognize a dozen of different animals. We should provide such machine precise and determined principles to specifically describe each of the animals. It would obviously be an excessive overhead for such a simple task.

Thinking more deeply about it, nobody ever even told us, as humans, how to tell cats and dogs apart: we simply learned what they are by seeing some **examples** of them. The biggest limitations that prevented the extension of such approach to machines have always been both the lack of a sufficient number of examples, being it several order of magnitudes higher than the one required by humans, and availability of computational power. In recent years, both limitations have been greatly smoothed out, opening whole new fields of research and opportunity. **Deep Learning**, the fields the techniques presented in this work rely on, has paved the most promising paths.

Most of the tasks we as human beings solve with great ease during the simplest day fall indeed in the aforementioned category of complex-to-define problems. **Robotics**, dealing with machines that can help and assist humans either by completely replacing them or decreasing their physical effort, is greatly benefitting from this kind of approach, especially thanks to an always increasing availability of 3D data and related architectures [3, 4, 8]. In fact, robotics needs to deal to the perception of the surrounding environment to properly explore, move and execute actions. The focus of this work is on

the application of deep learning techniques on a specific section of robotics, the manipulation of rigid objects and more specifically the ability to **grasp** them. Grasping objects is affected by several factors: object geometry, mass distribution, gripper-object friction, presence of other elements in the scene, *etc.* 3D data is a must-have for a robot to achieve awareness of the space around itself.

Even though there is increasing interest towards robotic grasping, the scientific community still lacks of established and recognized benchmarks to properly compare methods. This work is part of the BURG[1] project, whose aim is to enhance community building through enabling and sharing tools for reproducible performance evaluation.

1.2 Motivations

As underlined in the introduction, giving specific rules to solve complex tasks is likely to be limiting and makes it hard to achieve a good coverage of all the real-life scenarios. First DL approaches to robotic grasping have shown comparable performances with respect to older techniques. Nevertheless, even state-of-the-art approaches in robotic grasping still rely on custom heuristics [1] or setting-specific constraints and modeling that makes it hard to extend these techniques to environments with different characteristics. Most of these heuristics are related to identify the most promising graspable areas of an object's surface.

Recent works [12, 11] have introduced the possibility to learn how to sample key points on 3D point clouds providing a simple and task-agnostic method. The main intuition that gave the birth to this work is that it is possible to make a DL model free of custom constraints learn such contact points pattern and that learning would help into generalize better.

1.3 Main contributions

This work is the first attempt to build a lightweight completely learned grasp pose estimation pipeline free of complex heuristics with the aim of providing an almost plug and play architecture extendable to different settings. More specifically, the contributions of this work can be summarized in:

[1] https://www.chistera.eu/projects/burg

- a sampling-based end-to-end solution for visual learning of robotic grasp pose estimation whose grasp quality metrics result are comparable to current state-of-the-art solutions with much faster test execution times and lighter training;
- we are the first to use a feature encoder pre-trained on relevant self-supervised tasks [8] in a grasp setting since the same approach has shown improvements on 3D shape completion task;
- we provide an extension to an existing grasp dataset to better evaluate the comparison of different models with domain shift.

1.4 Structure of contents

Contents are organized in the following manner:

- Chapter 2 is an introduction to the 3D world, enlisting the main kind of data and the most common architecture used to learn patterns in the 3D setting;
- Chapter 3 formalize the robotic grasping task and present how the problem has been addressed so far in the literature;
- Chapter 4 is devoted to the detailed explanation of the main contribution of this work: a novel method of generating grasp poses based on differentiable learning techniques;
- Chapter 5 compares the results obtained with our method to the state-of-the-art solutions;
- Chapter 6 provides a wrap-up of the previous chapters and give suggestions to future improvements.

Part I

First part: Background

Chapter 2

Geometric Deep Learning

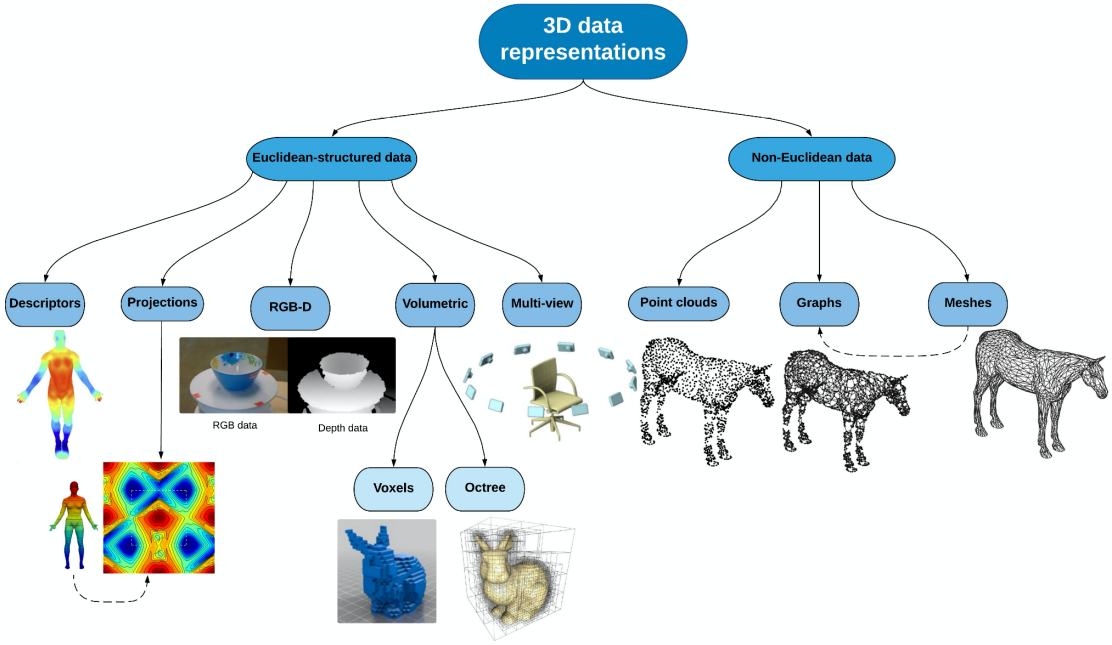

Figure 2.1. 3D data representation summary schema. The main division is about the euclidean structure of the data. Figure source: [2]

Deep Learning (DL) has proven itself able to achieve state of the art results on computer vision tasks when dealing with 2D data. This has been the case of, among many, image classification [13], segmentation [14], detection [15], localization, and so on.
The success of DL, especially through Convolutional Neural Networks (CNN),

in the 2D field has been pushed by an always increasing availability of regular-structured data (which RGB images belong to). The same was not true for 3D data until few years ago when, thanks to the advances in 3D sensing technologies, acquisition devices for this kind of data, such as RGB-D sensors or structured-light scanners, became more affordable, dramatically increasing the quantity of generated data.
Nevertheless, the regularity of input data is crucial for the functioning of classical CNN architectures, which exploit such regularity to progressively learn discriminative hierarchical features during their training. The lack of the same kind of regularity for most of the 3D data acquisition techniques makes it far from straightforward to build architectures able to perform effective shape learning, the latter being defined as the task of learning a mapping from an input geometrical signal to a multi-dimensional feature embedding [16].

2.1 3D Representations

Data captured by 3D scanning devices can be mainly grouped in two families, differing for both structure and properties: Euclidean-structured and non Euclidean-structured data.

2.1.1 Euclidean data

This kind of data preserves the properties given by its grid structure such as the global parametrization and a common system of coordinates. The most common examples that fall into this category are RGB-D data, volumetric and multi-view data, *etc.*

RGB-D

RGB-D data combines the information coming from the canonical three channels of 2D images with the corresponding **depth map**. Depth data encode the distance of the corresponding scene pixel to the camera, this way encoding the third dimension. Their popularity is mainly linked to the inexpensiveness of RGB-D sensors, combined to their simplicity and effectiveness representation, as testified by the notable results obtained in tasks like pose regression, scene reconstruction and identity recognition. Also, the availability of datasets in this type of format compared to the others is huge.

Volumetric

3D data can be modeled through **voxels**, which is the volumetric projection of the concept of *pixel*. The voxel representation leverages the characterization of the three dimensional space as a regular grid by describing how the object is distributed across the three dimensions of the scene, also including information about the point of view of the observer: a voxel can be classified as visible, occluded or self-occluded accordingly. The downside of a voxel-based representation is that, despite its simplicity and richness of information, it requires the storage of not so valuable data, since by construction it includes both occupied and non-occupied parts of the scene.

For this reason, the **octree**-based representation is usually preferred. It extends the voxel concept by enabling varying its size. Through a recursive process, it decomposes the root voxels in a hierarchical manner so that also the finest details of the observed object. Nevertheless, both types of representation share the inability of preserving some characteristics of the shape, such as the smoothness of its surface or its intrinsic properties.

Multi-view

The **multi-view** approach combines multiple 2D images of the same object captured from different view points. For each of these view, a learning function is learned separately, allowing the presence of different feature sets. These functions are later jointly optimized to represent the whole 3D shape, such that problems due to noise effect, incompleteness and occlusion, that usually arise from learning a single representation, are smoothed out.

Even though it is evident that this kind of approach would overperform a single-view based representation, the number of minimum needed views per object is task-dependent and not easy to estimate a priori, regardless of the fact that for many applications the single view is an unavoidable constraint.

2.1.2 Non-Euclidean data

Non-Euclidean data distinguishes itself for the lack of: (1) global parametrization, (2) a common system of coordinates and (3) a vector space structure. For these reasons, it is not straightforward to extend the existing 2D DL techniques to this type of data. The major obstacle that preclude the use of architectures such as CNN to non-euclidean structured data is that the convolutional operator is not well-defined in this space, forcing its replacement with other types of modules, such as the pooling operator [3]. Non-Euclidean

data has also been addressed as *Geometric Data* [17] and by extension the adaptation of DL techniques from the 2D to the 3D world have been labeled as *Geometric Deep Learning*. The main focus of this work revolves around the **point-cloud** encoding, even though also meshes are here presented, being one of the most popular representation.

Point Clouds

The formal definition of a point cloud is an *unordered* set of N vectors x_i in a D dimensional space:

$$\Phi = \bigcup_{i=1}^{N} x_i, \text{ with } x_i \in \mathbb{R}^D \tag{2.1}$$

Each vector representing a point in the space. The simplest use-case of point clouds is the 3-dimensional space, the three dimensions representing their location in the xyz cartesian system, but some more information can be added by extending the space with new dimensions, such as colors, normals, rigidity, *etc.* By construction, they are unstructured, being continuously distributed in the space, and any permutation of their ordering does not affect their spatial distribution, being each point independent from the others.

The latter leads to one of the main drawback of the point clouds: the ambiguity about surface information due to lack of connectivity and relations between points. Also, the first constraint for any DL model working with point clouds is the *permutation invariance*. The function learned by such models should be robust to permutation, more formally the model must be able by construction to learn a function f:

$$\begin{gathered} f\colon \mathbb{R}^{NxD} \to \mathbb{R}^T \\ \pi : \text{permutation function} \\ f(x_1, x_2, ..., x_n) \equiv f(x_{\pi 1}, x_{\pi 2}, ..., x_{\pi n}) \text{ with } x_i \in \mathbb{R}^D \end{gathered} \tag{2.2}$$

In a subsequent section the major challenges that these types of architecture have to face will be addressed more deeply.

Meshes

A 3D mesh is composed of a set of polygons (the faces). Each polygon is described as a set of vertices, that end up describing the mesh coordinates in the space. Like in a graph, the vertices are connected by edges. Even though

they are more structured than point clouds, the convolutional operator is still not defined for this kind of data. Some effort have been made in the last years in this direction, for example trying to exploit the graph properties of the meshes to define a convolution-like operator on top of the Laplacian eigen-decomposition [18].

2.2 Learning Architectures on Point Cloud

The method and the experiments performed in this thesis are solely related to point cloud data. Here are presented the main architectures presented in the literature. The major challenges that every network dealing with point clouds have to face are:

- **permutation invariance**: being the point cloud an unordered set of points, the learning function should be invariant to permutations (as stated in 2.2).
- **transformation invariance**: rotating and translating points all together should not alter the task output.

2.2.1 PointNet

PointNet [3] is the first architecture able to consume point clouds in an end-to-end fashion. The authors presented a network that enables the solution of different tasks: Shape Classification, Semantic Segmentation and Part Segmentation. It receives as input directly the xyz coordinates of the points, even though it is possible to extend the approach to an higher number of dimension.

First, it maps each input point to an higher dimensional space through a shared MLP. Secondly, it achieves the *permutation invariance* property thanks to a **pooling** symmetric function layer: it could either be a max or an average aggregation. The pooling layers represent the mirror of the 2D convolutional layers in the pc non-Euclidean domain. The aggregation output is a *global descriptor* for the entire shape, encapsulating information coming from each of the input points.

The *transformation invariance* property is harder to achieve. While the permutation invariance can be obtained through a simple mathematical operation (a position-invariant pooling), the rotation/translation invariance does not have a direct counterpart. PointNet tries to be robust in this sense by exploiting two spatial transformer networks:

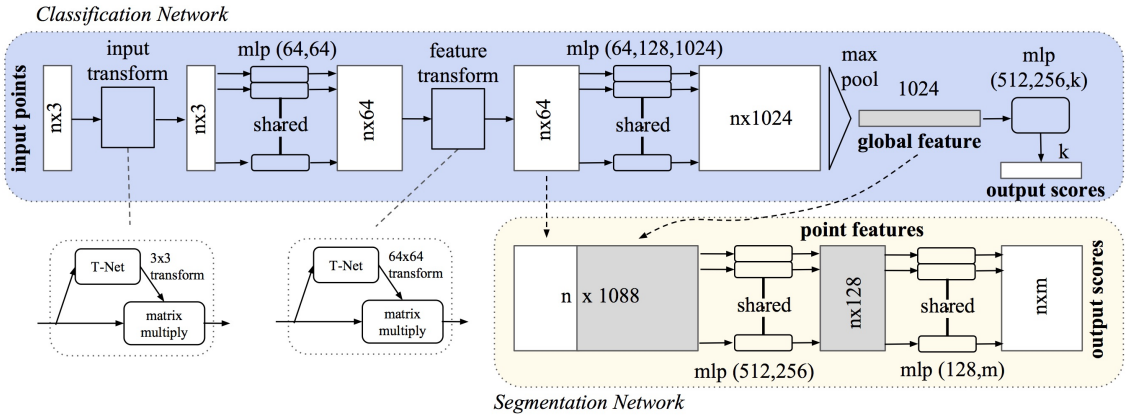

Figure 2.2. PointNet architecture: classification and segmentation networks. Input is raw x, y, z point cloud data of size n × 3. Raw PointCloud. 3 × 3 transformation matrix is applied to input Point Cloud. If enabled also a 64×64 transformation matrix is applied regularizing features from different point clouds. Point features are aggregated by max pooling. All layers (except last one) include batch normalization and ReLU; Dropout is used in the final MLP. Fig. source: [3].

- **STN3d**: it learns an affine transformation matrix that is applied to shape input via dot product in an attempt to align different input shape.

- **STNkd**: it produces and alignment in the feature space through a feature transformation matrix.

As shown in 2.2, classification and segmentation branches share most of the network architecture, which can be considered as a *feature extractor*, both at **local**-wise level (the coordinates of each point are mapped to an higher dimensional space) and at **global**-wise level, since the aggregation produces a 1-dimensional vector. Starting from these two kind of descriptors, different tasks can be performed: the classification task takes as input the global feature and tries to predict the shape class, while the segmentation task needs to combine both local and global information to predict the segment each point belongs to. Classification and segmentation are only two of the different task that can be performed, but they are the perfect example of how both the embeddings (local and global) are suitable to different task.

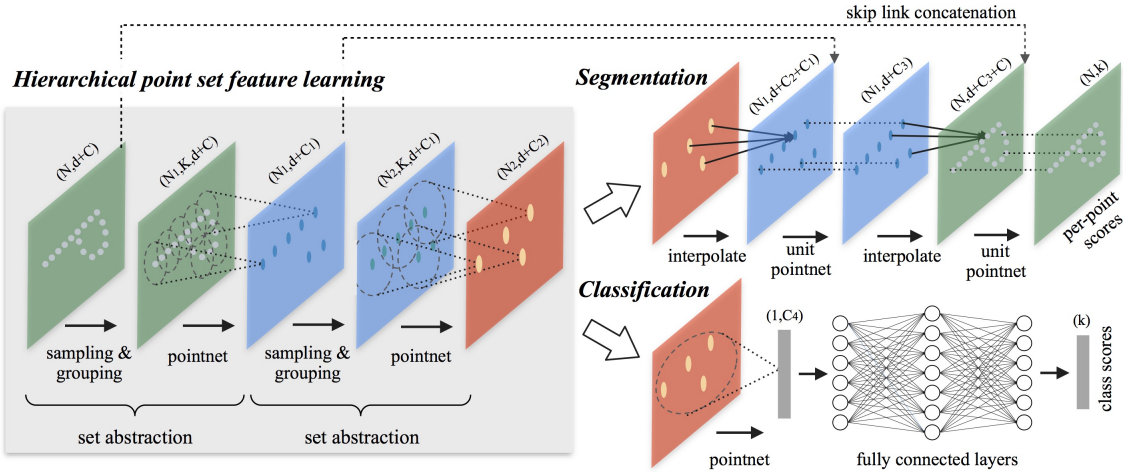

Figure 2.3. PointNet++ architecture. Hierarchical feature learning is introduced to learn features at various scales. Sampling and grouping layers define and sample neighbourhoods at various sizes which are fed into a PointNet module architecture for local pattern extraction. Fig source: [4].

2.2.2 PointNet++

PointNet++ [4] exploits the ideas introduced by PointNet with the aim to extract information in a hierarchic fashion by recursively applying PointNet itself on nested partitions of the input point set. PointNet then simply becomes a functional block in a bigger architecture. At a high-level, it applies different techniques to overcome the major limitation of PointNet, which is the absence of correlation between points in their neighbourhood. Moreover, it proposes novel layers to adapt the learning to point clouds with non-uniform distribution densities.

The learning process happens by iterating two main processes:

- **partitioning**: a partition is a **neighbourhood ball** in the underlying Euclidean space, defined by a centroid location and the scale. The coverage of the whole shape is ensured by sampling the centroids through farthest point sampling (FPS).
- **abstracting**: through PointNet, each local region is abstracted by its centroid and local encoding.

The recursive application of abstractive operators to small local neighbourhood and the further grouping of such features into larger units mirrors the

CNN behavior in 2D space, overcoming the absence of direct convolutional layers for unstructured 3D data like point clouds.

2.2.3 DeCo

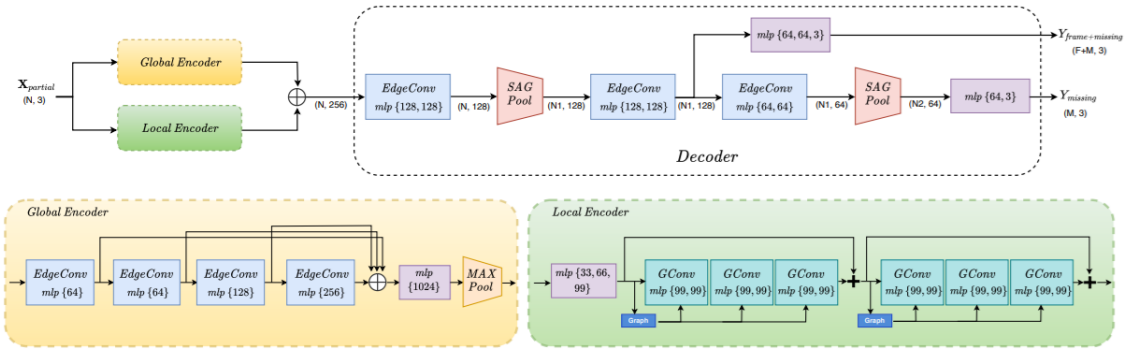

Figure 2.4. DeCo point cloud completion architecture. Global and local encoders extract semantic and geometric information, respectively, from the partial point cloud by pretraining with contrastive and denoising pretext tasks. The decoder converts this information into the points of the missing part. EdgeConv [5] and GConv [6] are graph convolutional layers, SAG Pool [7] is a graph pooling method. $\bigoplus$ denotes concatenation, $+$ denotes summation. Fig. source: [8].

DeCo [8] is a novel architecture that is originally conceived to leverage the power of self-supervised approach to perform 3D point cloud completion. The training is conducted in two stages: self-supervised pretext and downstream task training. More specifically, the architecture presents two branches, each leveraging a different self-supervised task:

- Global encoding by **conrasting**: the goal of contrastive learning [19] is to learn an embedding by enforcing closeness among similar samples and distance among dissimilar ones. When in a self-supervised setting, data augmentation has proven to be a fair method to generate several positive and negative pairs starting from unlabeled data. By promoting similarity between augmented samples generated from the same shape, it learns to capture a global understanding of the latter, regardless of the specific application of the transformation. The building block of this branch are edge-convolutional layers [5].

- Local encoding by **denoising**: since it mostly relies on low-level geometric characteristics naturally shared by multiple classes, detached from the global semantic of the shape, denoising is a great fit for catching information at a local level. The base building block of this branch are graph convolutional layers [6], which subsequently aggregates neighbourhood-like features in an always updated high dimensional space.

The DeCo feature extractor does not require labeled data since it completely relies on self-supervised techniques. For this reason, given the high availability of unlabeled data, it can be pre-trained on large un-annotated datasets to be later fine-tuned on the specific supervised task.

Chapter 3

Robotic grasping

Manipulation of objects is involved in the majority of activities we have to perform every day. **Robotic grasping** deals with the implementation of artificial systems to mimic the object handling of the human body (hands in particular). The great interest in developing robots able to operate in dynamic and unstructured environments comes both from industry (bin-picking, professional services) and from daily life (household environment). The necessity of automatically configurable devices, with a flexible and customizable grasp pipeline, has been addressed by *learning* based approaches able to configure for the given task with little human intervention. The great challenge is then to build methods prone to generalize to novel objects. This chapter serves as an introduction to the robotic grasping problem, its definition and the major challenges it has to face. Among the different classes of existing approaches, it is performed a deeper focus on the **learning-based** ones, category which the method presented in this thesis belongs to.

3.1 Grasp problem formalization

Literature shows different ways in which the grasp problem has been approached and formalized, also depending on the input data the specific method is fed with. In this work, we stick with the definition used in [10].

Generally, a grasp is defined as the pose in which the gripper closes its fingers, i.e. when the gripper makes contact with the object. Given a grasp, we still need to identify trajectories how the robot can move to that grasp pose and how it removes the object from the scene, but these problems are usually detached and there are established solutions available. Therefore the main challenge is to find grasp poses which allow the gripper to lift and hold

Figure 3.1. Robotic arm equipped with a 2-finger parallel-jaw gripper. The grasp task is usually intended as lifting and holding the object in a stable pose. Fig. source: [9].

the object. More formally, a grasp $\mathbf{g} \in SE(3) \times \mathbb{R}^n$ fully describe both pose and configuration of the chosen gripper, n being the number of degrees of freedom. To simplify and reduce the search space, most of the recent works relies on two reasonable assumptions:

- the usage of a 2-finger *parallel-jaw* gripper, which reduces the degrees of freedom to 1.
- consider the fingers to be fully opened before the grasp motion is actually performed, which further reduce the grasp space to $SE(3)$.

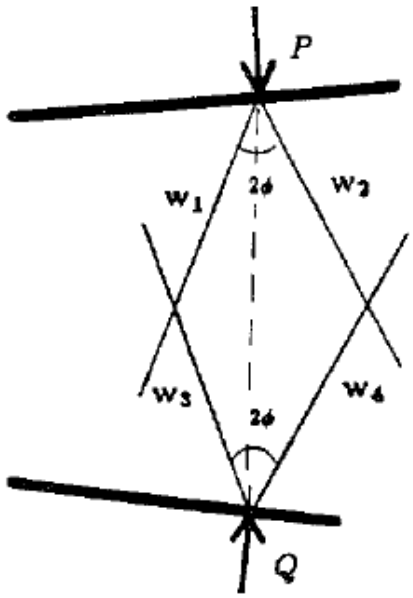

Figure 3.2. An antipodal grasp. The segment PQ needs to fall into both *friction cones* defined at P and Q.

Depending on the specific method, the $SE(3)$ space can be parametrized in diverse settings. In this work, we rely on the parametrization presented in [1]. The grasp is formulated as $\mathbf{g} = (\mathbf{x}, \phi) \in SE(3)$, where $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$ is the center of the two parallel jaws and $\theta \in [-\pi, \pi]^3$ is the Euler angle vector encoding the orientation of the gripper.

To better suit a learning based approach, some variations of this parametrization are usually preferred. First of all, in order to avoid the singularities of an Euler angle representation, the orientation of the grasp pose is encoded in *unit quaternions* [20]. The same information can be embedded in a slightly different re-parametrization $(\mathbf{c_1}, \mathbf{c_2}, \theta)$, where $(\mathbf{c_1}, \mathbf{c_2})$ is the contact points pair, belonging to the object surface, and $\theta \in [-\pi, \pi]$ is the *pitch* orientation of the grasp approach direction (displayed in Fig. 3.3).

3.1.1 Antipodality

An intrinsic physical and geometric property of grasps is *antipodality*. We will not go much in details here since the formal definition has already been widely discussed in the early literature [21, 22, 23]. In a nutshell, the **antipodal** property requires the two contact points of a grasp to be on opposite sides

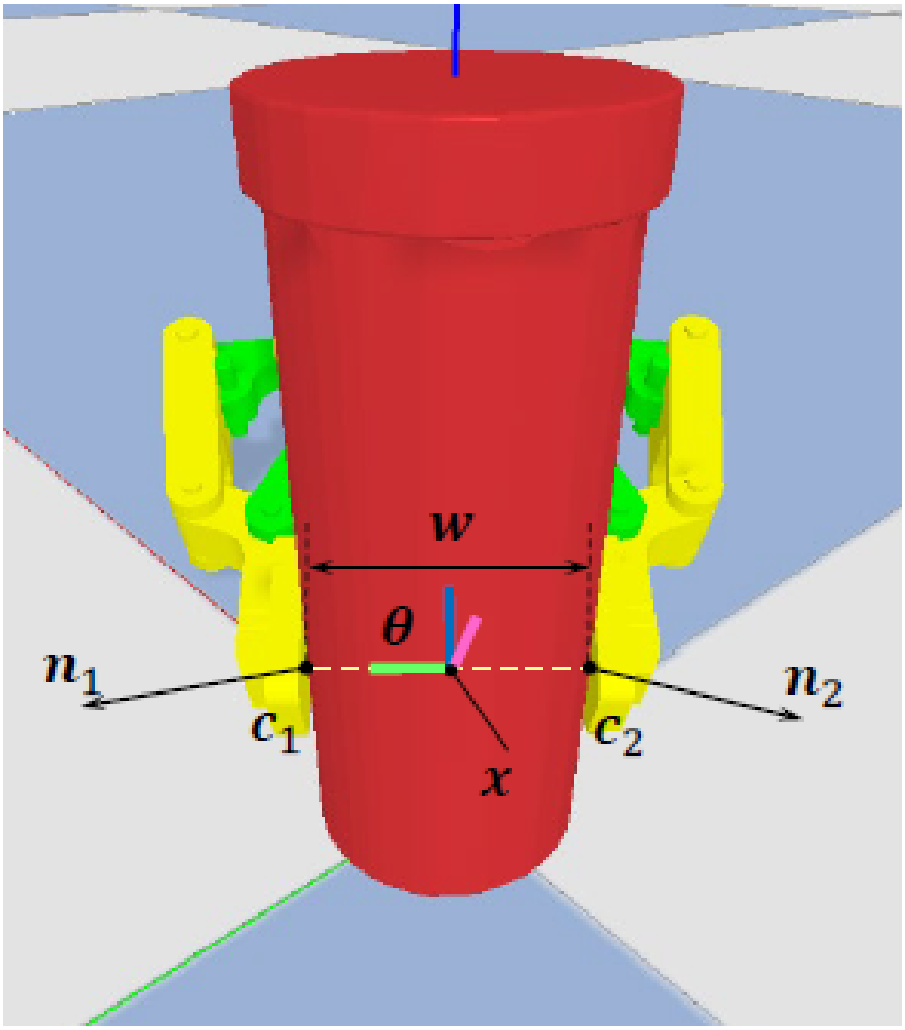

Figure 3.3. Illustration of the 6-DoF grasp parameterization. The two contact points are denoted as $\mathbf{c}_1$ and $\mathbf{c}_2$, with the corresponding normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$. A grasp candidate is parameterized by gripper center $\mathbf{x}$ and gripper orientation θ. An additional freedom of gripper opening width w is sometimes used in the literature. Fig. source: [1].

of the shape of an object as shown in Fig 3.2, which make it quite unlikely to have them both visible at the same time.

3.1.2 Evaluation metrics

The success of a grasp depends on the specific underlying task the method is trying to perform. The basic grasping task requires the lifting and holding

of the object for some seconds, to later return to the original state.

Typically, an effective grasp pipeline should generate not only successful, but also *diverse* grasps, such that the object can be grasped even in cluttered scenes or given spatial constraints. Therefore, measures other than the simulation or real-world testing have been introduced to evaluate a **set** of generated grasps for a specific object.

These measures rely on **distances** between grasps. In [24] is suggested a weighted metric. Let $\mathbf{g}, \mathbf{h}$, with $\mathbf{x_g}, \mathbf{x_h} \in \mathbb{R}^3$ being their centers and $\mathbf{q_g}, \mathbf{q_h} \in \mathcal{S}^3$ their orientations as unit quaternions. Such distance is defined as:

$$\rho(\mathbf{g}, \mathbf{h}) = \omega \|\mathbf{x_g} - \mathbf{x_h}\|_2 + \arccos(|\langle \mathbf{q_g}, \mathbf{q_h} \rangle|) \tag{3.1}$$

With ω set such that a translation of 1mm in the centers distance is equal to a rotation of 1°.

Coverage

It is interesting to understand how much the grasp a method generates ($\mathcal{X}$) covers the full set of successful reference grasp ($\mathcal{R}$). Such reference set could either be human-annotated or generated in simulation. A grasp in the reference set can be considered to be "covered" if, among the generated grasps, there is at least one grasp "close enough" to it. Formally, the coverage metric is defined as:

$$\text{cov}(\mathcal{X}, \mathcal{R})_\epsilon = \frac{|\{\mathbf{g} \mid \mathbf{g} \in \mathcal{R} \wedge \exists\, \mathbf{x} \in \mathcal{X} : \rho(\mathbf{g}, \mathbf{x}) \leq \epsilon\}|}{|\mathcal{R}|} \tag{3.2}$$

where $\epsilon \in \mathbb{R}$ is the distance threshold. Typical values in literature range from 3.5mm to 5mm, which make such measure sensitive to variations of the evaluation method and thus to be used carefully. In our work, we consider 3.5mm as reference value.

Precision

Precision is defined by the ratio of successful grasps among all the ones the method is able to generate. Depending on the specific application, a typical trade-off is the one between precision and coverage. The more grasps are generated, the more likely is that the shape is well covered but also that some grasps are not successful since some regions of the object may be inherently difficult to grasp.

Rule-based precision When it is not convenient (or worse, not even doable) to evaluate the generated grasps through a simulator, some distance-based rules have proven to be a good proxy to evaluate a method. For example, [1] counts as *success* a generated grasp if, according to a threshold, is close enough to a positively annotated grasp in the reference set.

3.2 Learning-based robotic grasping

Data-driven approaches have gained popularity in recent years, mainly thanks to an always increasing availability of sources of data. [25] further divides this class of methods into:

- **model-based**: there is some prior knowledge of the object (typically a CAD or a previously scanned model). The learning happens through a three-stage process: (1) object poses estimation, (2) grasp pose determination and (3) grasp execution path planning to actually pick the object without collision. Relying on a prior knowledge, they are usually not able to generalize to novel objects.
- **model-free**: they lack of prior knowledge about the objects, thus the object pose estimation step is skipped. On the other hand, not imposing constraints on object modeling allows for a better generalization power.

The approach later presented in this thesis, along with the most recent works in grasp learning, falls in the second category, and more specifically in the *discriminative* class.

3.2.1 Model-free formalization

The tasks involved in the visual grasp learning process can be formalized as either learning a **grasp scoring function** [26] or a **grasp regression function** [27].

The first one can be formalized as:

$$\Omega : \mathbb{R}^{n\times 3} \times (\mathbb{R}^3 \times [-\pi, \pi]^3]) \rightarrow [0,1] \tag{3.3}$$

which, given the point cloud and a grasp, predicts a score which can be seen as its success probability. The aim of such scoring function is to create a **rank** among all the candidate grasps and to later execute the ones with the highest score.

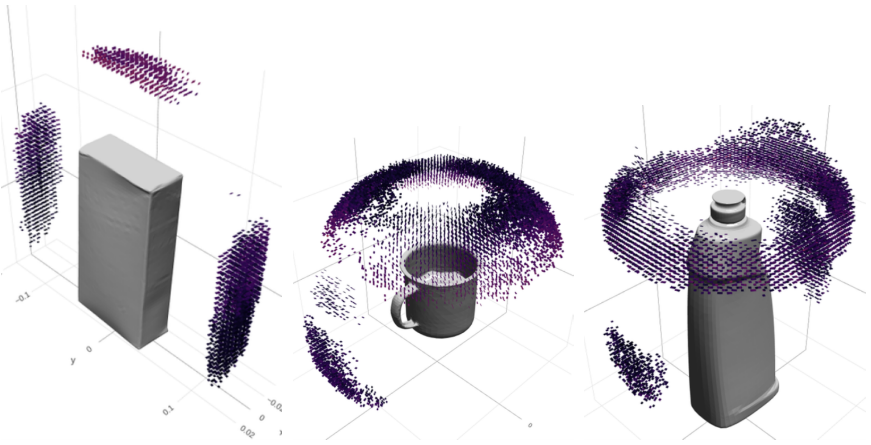

Figure 3.4. Objects (sugar box, mug and bleach cleanser) and the corresponding successful grasps obtained through a simulator. For each object, the grasp distribution is discontinuous and presents multiple modes. Fig. source [10].

There are several ways to obtain the previously mentioned grasp candidates, but at a high level the regression function has to take as input the point cloud and outputs one or multiple grasps as formalized in eq. 3.4.

$$\Phi : \mathbb{R}^{n\times 3} \rightarrow \mathbb{R}^3 \times [-\pi, \pi]^3 \tag{3.4}$$

As eq. 3.3 and eq. 3.4 indicates, the robotic grasping task is reduced to learning mapping functions from observed geometric properties of the shape surface to physically and semantically sensible grasp proposals.

The major identified challenges are:

1. point cloud data encoding does not include physical properties like mass density distribution and surface material, as well as friction coefficient between the gripper and the object surface;

2. successful grasps are often distributed along multiple modes (fig. 3.4) of the object surface and the optimal configuration depends on external factor such as the surrounding environment;

3. grasp annotation cannot be continuous and, depending on the specific source of data, may be sparse;

4. as every deep-learning based approach, it requires a huge quantity of data, which implies the usage of grasp simulators and therefore of techniques to transfer the knowledge to the real-world (see section 3.2.2)

3.2.2 Simulation and sim-to-real transfer

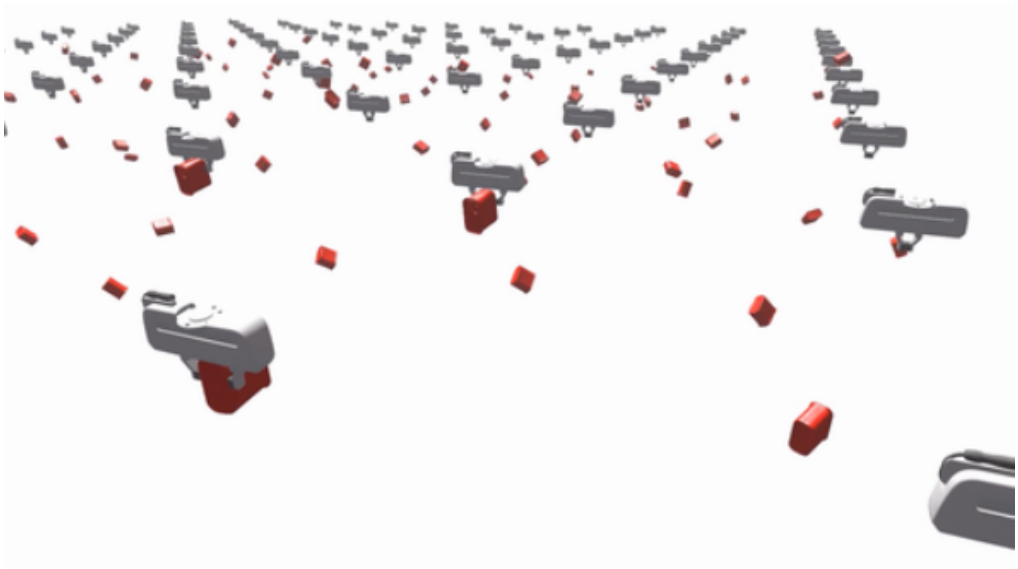

Figure 3.5. Grasps performed by a parallel-jaw gripper with the FleX simulator. Fig. source [10]

Deep learning approaches have proven to have a better generalization ability with respect to other methods. Despite these advantages with respect to performance and test time robustness, the drawback is the high demand of data for training. A barely sufficient amount of data to properly train the networks requires expensive and time-consuming resources. Real world grasp datasets have been collected [28, 29, 30], but to such process is not scalable nor flexible and invariant to changes in the setup (such as changing the gripper or moving the camera).

Simulation has proven to be the best alternative to real-world data. The most common physics simulator are PyBullet [31], Blender [32], FleX [33], just to name a few. Being fast and parallelizable, simulations overcome the scalability approaches presented by real robotic arms. There are obviously disadvantages, ranging from the least problematic, such as license costs and configurations needs, to the inability to fully reproduce the properties of the real world and thus often requiring specifically designed methods to transfer the acquired knowledge to real applications.

The sim-to-real world transfer mainly happens leveraging techniques largely used in visual learning, such as **domain randomization** and **domain adaptation**.

Domain Randomization Works by applying various randomizations to the observations in order to prevent the model from learning the simulation environment and make it see the real world as just another different variation. Concretely, it may happen by randomizing the textures, the colors of the object and the background, the camera placement and lightning, *etc.*

Unsupervised Domain Adaptation Domain Adaptation consists in learning a predictor in the presence of a shift between train and test distribution [34]. It can either happen by learning auxiliary self-supervised tasks on both domains simultaneously to induce alignment between source and target domain [35, 36], or by adversarial learning [37], pushing the model to extract task relevant, yet *domain invariant*, features. The major downside of these approaches is the need for real, yet unlabeled, data from the target domain.

Part II

Second part: Learning2Grasp

Chapter 4

Method

This chapter is devoted to the introduction of *Learning2Grasp*, a novel and completely learned pipeline to generate grasp poses starting from a partial view of the object based on a sampling approach. Indeed, most of the recent works on robotic grasping either rely on some kind of geometric heuristic to build a set of candidate graspable points [26, 38], or they sample a very large quantity of grasp candidates and subsequently train a neural network architecture to prune most of them [1]. Also generative methods such as [9] need to evaluate and even refine the generated candidates to make them successful in a real-time scenario. The work here presented aims to overcome these limitations by **learning to sample** contact points from the partial view without imposing any geometric or custom spatial constraint to ease the process of extending the approach to different datasets and settings.

4.1 Method rationale

As addressed in Chapter 3, the grasping task comprises both estimating the gripper pose and finding trajectories to execute such pose without colliding with the environment or the object itself. Typically, a single camera is only able to capture a *partial* view of the object. For the intrinsic antipodal property (section 3.1.1), usually only one of the two contact point defining the grasp is on the visible surface of the object under consideration, the other lying on the hidden side of it. Following the parametrization introduced in 3.1 and indicating with $\mathcal{P}$ the partial view point cloud, for the generated grasp $\mathbf{g} = (\mathbf{c}_1, \mathbf{c}_2, \theta)$ either $\mathbf{c}_1 \in \mathcal{P}$ or $\mathbf{c}_2 \in \mathcal{P}$. Such constraint leads to subsequently subdivide the pose estimation task into two phases:

1. select contact candidates on the visible surface;
2. for each of such candidates, regress the second contact point (and the relative orientation) to build the grasp.

Phase 2 can be easily reduced to a regression task, for which many learning-based solutions are available.

On the other hand, the same is not true for the phase 1. Defining $\mathcal{C}$ the set of points in $\mathcal{P}$ for which exists at least one successful grasp involving such point, phase 1 of the process basically requires to select a set $\mathcal{C}^* \subset \mathcal{P}$ that best represents $\mathcal{C}$. In other words, it can be intended as a **sampling** operation. One naive approach could be to randomly sample the source point cloud. More sophisticated approaches leverage task-related sampling techniques like *furthest point sampling* [39], but there is always the need to design each time a different and usually complex heuristics to specifically fit the task under analysis.

More recent learning-based approaches [11, 12] have proven to produce comparable performances in the same setting the heuristics were designed for, with the benefit of being extendable to any kind of downstream task. The main novelty of our work is to make phase 1 learnable as well, in order to free the grasp pose estimation task of environment and domain custom constraints. We suppose such approach have huge potential since objects typically show **patterns** in the distributions of contact points on their shape (ex: grasp a box from its side, grasp a bowl along the rim, *etc...*), and we aim to learn this pattern by exploiting state-of-the-art neural network architectures.

4.1.1 Learning to Sample

Formally, the **task-related** sampling problem requires, given a point set $\mathcal{P} = \{p_i \in \mathbb{R}^3, \quad i = 1, ..., n\}$, a sample size $k \leq n$ and a task network T, to find a subset $\mathcal{S}^*$ of k points that minimizes the task network's objective function f:

$$\mathcal{S}^* = \arg\min_{\mathcal{S}} f(T(\mathcal{S})), \quad \mathcal{S} \subset \mathcal{P}, \quad |\mathcal{S}| = k \leq n \tag{4.1}$$

A specific case of such formulation is when the network T is an identity and the objective function is directly designed and applied to the sampling output. The major issue is that the sampling operation is *not differentiable*, thus the gradient with respect to the sampled value cannot be calculated. As

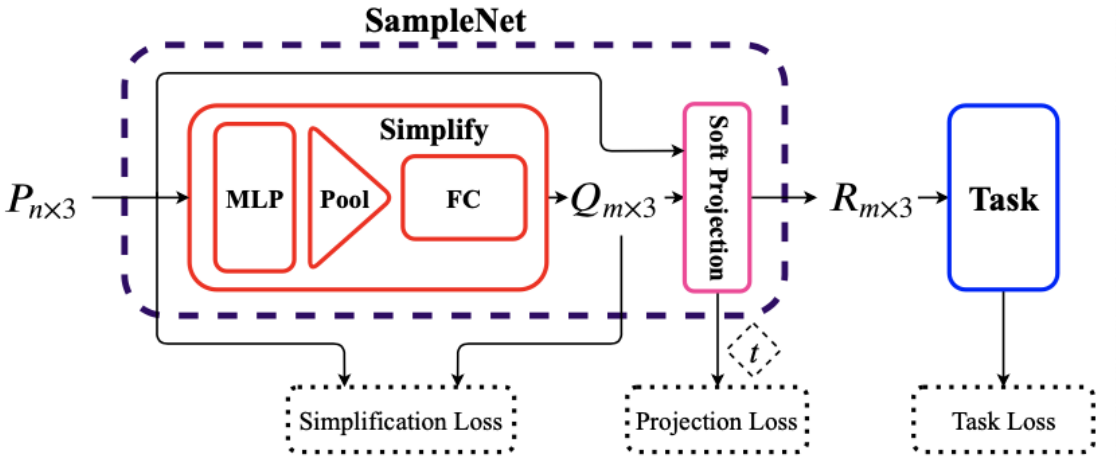

Figure 4.1. SampleNet architecture. First, the input point cloud P is *simplified* to a smaller set Q. Such simplified set is projected onto P to obtain R, the input to the task network. Fig. source: [11].

a consequence, the sampling operation cannot be trained directly. While [12] proposes a workaround through a two-step process, SampleNet [11] specifically designs a differentiable, and thus learnable, nearest neighbor approximation, the **soft-projection**. Under specific conditions, the soft-projection operation basically reduces to sampling. The sampling happens via a three phase process as depicted in Fig. 4.1:

1. the input point cloud $\mathcal{P}$ is simplified to a smaller set $\mathcal{Q}$ via a neural network;
2. $\mathcal{Q}$ is projected onto $\mathcal{P}$ by the soft-projection operation obtaining $\mathcal{R}$;
3. $\mathcal{R}$ is fed to the specific task network.

Point Cloud Simplification Even though the simplified point cloud $\mathcal{Q}$ is later projected onto $\mathcal{P}$, to encourage the closeness between the source and the simplified set a *simplification loss* term is added, based both on average nearest neighbor loss:

$$\mathcal{L}_a(X,Y) = \frac{1}{|X|}\sum_{\mathbf{x}\in X}\min_{\mathbf{y}\in Y}\|\mathbf{x}-\mathbf{y}\|_2^2 \tag{4.2}$$

and maximal nearest neighbor loss:

$$\mathcal{L}_m(X, Y) = \max_{\mathbf{x} \in X} \min_{\mathbf{y} \in Y} \|\mathbf{x} - \mathbf{y}\|_2^2 \tag{4.3}$$

The simplification loss term is built as such:

$$\mathcal{L}_{simp}(\mathcal{Q}, \mathcal{P}) = \mathcal{L}_a(\mathcal{Q}, \mathcal{P}) + \gamma \mathcal{L}_a(\mathcal{P}, \mathcal{Q}) + \beta \mathcal{L}_m(\mathcal{Q}, \mathcal{P}) \tag{4.4}$$

The first and the last term forces the generated data to stay close to the source cloud both in the average and in the worst case, while the second term ensures the full coverage of the source. The above defined loss function is an extension of the Chamfer Distance [40], achieved with $\beta = 0$ and $\gamma = 1$.

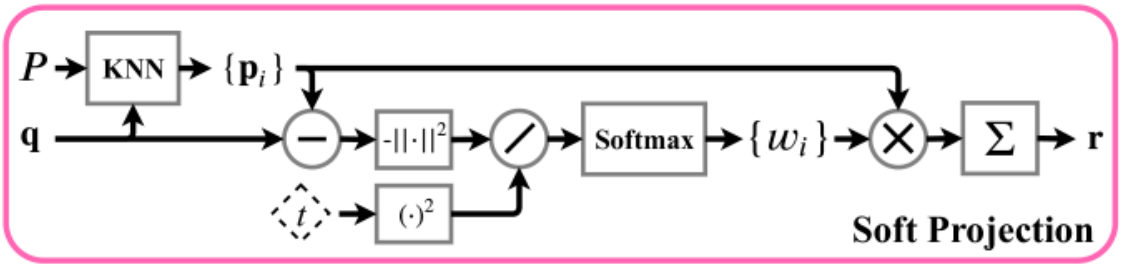

Figure 4.2. **The soft projection operation**. Each point $\mathbf{q} \in Q$ is projected onto its k nearest neighbors in P, denoted as $\{\mathbf{p}_i\}$ and weighted by $\{\mathbf{w}_i\}$ according to their distance form $\mathbf{q}$ and a learned temperature parameter t. Fig. source: [11].

Soft Projection The operation is depicted in Figure 4.2. Each point belonging to the generated set $\mathcal{Q}$ is projected onto its local neighbourhood, composed of its k nearest neighbors in terms of euclidean distance in the reference point cloud $\mathcal{P}$. The operation of projecting a point $\mathbf{q}$ with respect to a specific neighbourhood $\mathcal{N}_{\mathcal{P}}(\mathbf{q})$ reduces to approximate such point as a convex combination of the neighbor itself, $\mathbf{r}$.

$$\mathbf{r} = \sum_{i \in \mathcal{N}_{\mathcal{P}}(\mathbf{q})} w_i \mathbf{p}_i$$

$$w_i = \frac{e^{-d_i^2/t^2}}{\sum_{j \in \mathcal{N}_{\mathcal{P}}(\mathbf{q})} e^{-d_j^2/t^2}} \tag{4.5}$$

$$d_i = \|\mathbf{q} - \mathbf{p}_i\|_2$$

The weights $\{w_i\}$ can be intended as a probability distribution over the reference points in $\mathcal{P}$. The weights distribution is guided by the temperature parameter t. The higher the value, the more uniform the distribution. On the other hand, when the temperature approaches 0, the distribution collapses onto a delta function, located at the nearest neighbor point. In other words, as the temperature goes to zero, the soft projection turns into an **hard** projection on the nearest point. Such process is illustrated in Figure 4.3. Thus, in order to obtain a differentiable nearest neighbor approximation, a loss term is introduced to encourage a small temperature value.

$$\mathcal{L}_{proj} = t^2 \tag{4.6}$$

[11] shows how to learn a decreasing temperature profile can be beneficial to the sampling task, even though other non learned profiles would work as well.

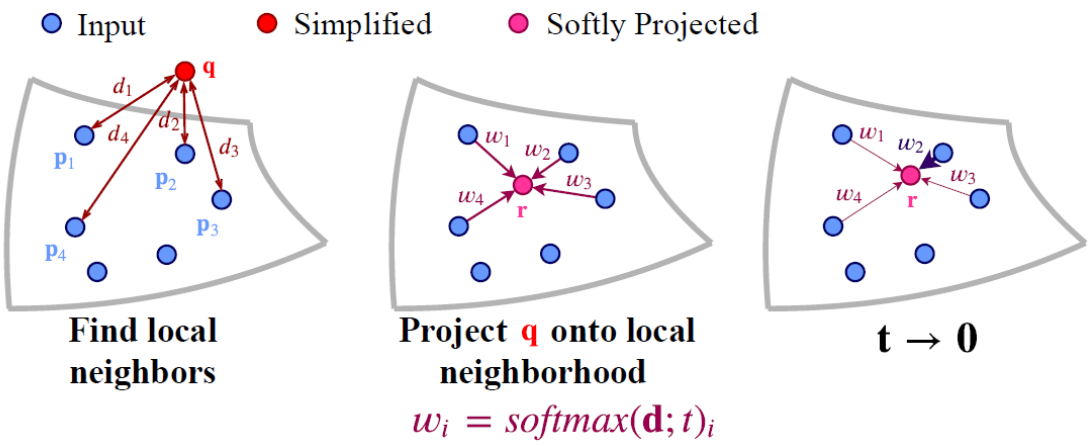

Figure 4.3. **Sampling approximation**. The projection reduces to nearest neighbor sampling as the temperature tends to zero. Fig. source: [11].

4.2 Method formalization

Lets define $\mathcal{P} \in \mathbb{R}^3$ an object *partial* point cloud and a set of grasps $\mathcal{G} = \{\mathbf{g}_i\}$, each $\mathbf{g}_i = (\mathbf{c}_1, \mathbf{c}_2, \theta, s)$, being $\mathbf{c}_1 \in \mathcal{P}$ the visible contact point and $\mathbf{c}_2 \in \mathbb{R}^3$ the contact point on the hidden surface, $\theta \in [-\pi, \pi]$ the pitch orientation and $s \in [0,1]$ a grasp score indicating its quality. Grasp with a quality score greater that a threshold t are considered *successful* and we can define the set of successful grasp with $\mathcal{G}^+$.

The architecture of the network we present here can be seen as a composition of four different modules stacked one after the other, each learning a different function: a *feature extractor*, a *contact sampler*, a *grasp regressor* and a *grasp classifier*. The whole architecture is presented in Figure 4.4

Feature Extractor It learns a function Φ that maps each point in $\mathcal{P}$ to an higher dimensional space. The learning of this function is influenced by the specific downstream task.

$$\Phi : \mathbb{R}^{n \times 3} \rightarrow \mathbb{R}^{n \times C} \tag{4.7}$$

Contact Sampler Inspired by [11], the sampler learns a function Γ that selects, among all the n points in $\mathcal{P}$, m points which are believed to form a good candidate set as grasp contact points. Let's name the sampled set

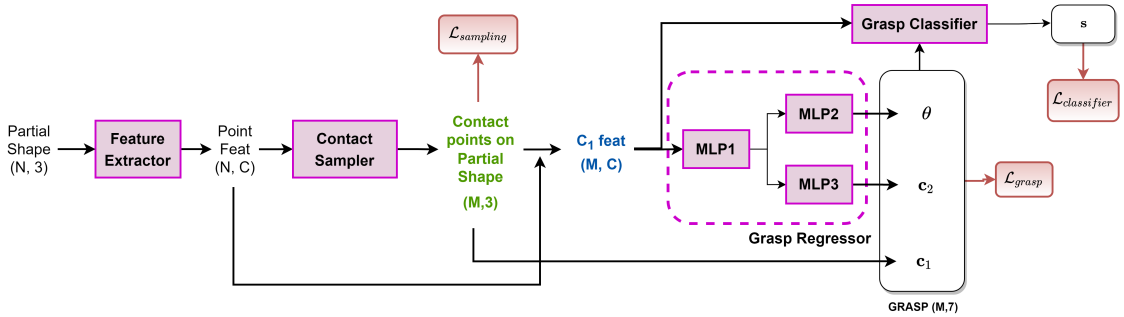

Figure 4.4. Learning2Grasp architecture. First, a set of candidate contact points is sampled from the partial point cloud. Starting from each of those points, a grasp is built regressing the second contact and the angle. A classifier finally evaluates the quality of the predicted grasps.

$\mathcal{C}^*$. Here we assume as input the point-wise features, but the sampler can be generalized considering as input the raw point cloud.

$$\Gamma : \mathbb{R}^{n \times C} \rightarrow \mathbb{R}^{m \times 3}, \quad m \leq n \tag{4.8}$$

The learning of Γ is guided by the loss term introduced in Eq. 4.4. The main difference with the original approach is that in our case the sampling task is **supervised**: an optimal subset for the downstream task is known at training time, which means some sort of *sampling truth* is provided. Defining with $\mathcal{C} \subset \mathcal{P}$ the reference set composed of the visible contact points $\mathbf{c}_1$ in the grasp set $\mathcal{G}^+$, the loss term is restricted to such subset instead of considering as target the whole point cloud:

$$\mathcal{L}_{sampling} = \alpha \, \mathcal{L}_{simp}(\mathcal{C}^*, \mathcal{C}) + \mathcal{L}_{proj} \tag{4.9}$$

Grasp Regressor The grasp regressor completes the grasp starting from the first contact point and its feature. It predicts both the second contact point $\mathbf{c}_2$ and the grasp orientation θ. It also leverages the point-wise features extracted for each of the *first* contact points. Figure 4.4 shows our specific implementation of the regressor, which predicts them in a parallel fashion using different MLPs.

$$\Omega : \mathbb{R}^{(3+C)} \rightarrow \mathbb{R}^{(3+1)} \tag{4.10}$$

The loss function for the grasp regression is designed according to the grasp distance metric introduced in Eq. 3.1. In this work, we assume that for each contact point $\mathbf{c}_1$ exists one only possible successful grasp, and thus the *grasp truth* $\mathbf{g} = (\mathbf{c}_1, \mathbf{c}_2, \theta)$ is well defined for each of the generated grasps $\mathbf{g}^* = (\mathbf{c}_1, \mathbf{c}_2^*, \theta^*)$ since they share the first contact point. Dealing with multiple truth is not very common in literature and is delegated to possible future works. Since at training time the set of sampled contact points is only *softly* projected on the point cloud (e.g. the sampled points do not actually belong to $\mathcal{P}$), the truth is computed considering the closest contact point to the sampled one. Following the center-quaternion grasp parametrization introduced in 3.1, the loss function can be formulated as such:

$$\mathcal{L}_{grasp} = \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{x}_{g_i} - \mathbf{x}_{g_i^*}\|_2 + \lambda \arccos(|\langle \mathbf{q}_{g_i}, \mathbf{q}_{g_i^*} \rangle|) \tag{4.11}$$

Grasp Classifier The grasp classifier is needed to: (1) prune out those grasps that are either unfeasible or likely to be unsuccessful and (2) generate a quality rank among grasps since most of the real world application require to execute one or few grasps and ranking them allow to chose the best one(s). It takes as input the complete grasp and predicts a quality score. The learned function is simply:

$$\Lambda : \mathbb{R}^7 \to \mathbb{R} \tag{4.12}$$

The classifier module is trained through a binary cross entropy loss which takes as input the generated scores s^* and as target the quality score s.

$$\mathcal{L}_{classifier} = \frac{1}{g} \sum_{i=1}^{g} -s \log(s^*) - (1-s)\log(1-s^*) \tag{4.13}$$

The whole model is trained in a joint fashion through a summation of the different losses such that each module can benefit from the contribution of the downstream losses. The idea is not to let each model operates in isolation but rather specifically tune the architecture for the grasping task.

$$\mathcal{L}_{tot} = \mathcal{L}_{sampling} + \mathcal{L}_{grasp} + \mathcal{L}_{classifier} \tag{4.14}$$

4.3 Implementation details

The Learning2Grasp architecture presented in Fig. 4.4 is strongly modular and the implementation of each of its components may vary depending on the specific requirements and resources available. Also, each module is trained driven by weighted combination of loss terms. The choice of such weights allow to give different behavior to the model.

Feature Extractor As a starting point, we built our model relying on well known and reliable architectures widely used in the grasp setting: PointNet and PointNet++. Nevertheless, one of the goal of our work was also to show how the grasping task would benefit from a feature encoder pre-trained with self-supervised tasks: DeCo.

Contact Sampler Apart from the feature extraction, we completely relied on the SampleNet the architecture here. The simplification loss weight was set to $\alpha = 10$, with both $\gamma = \beta = 1$. For all the others network parameters, we sticked to the default ones. The number of sampled points is set to $m = 400$ in order to obtain a number of grasps sufficiently high to have a good coverage of the shape.

Grasp Regressor Inspired by [1], when predicting a grasp starting from the first contact point, its neighbourhood in terms of euclidean distance is also considered. More specifically, for the specific first contact point $\mathbf{c}_1$ whose feature encoding is $f_{\mathbf{c}_1}$, the regressor is fed with a weighted combination of the features of its k-sized neighbourhood $\mathcal{N}_k(\mathbf{c}_1)$.

$$f_{\mathcal{N}_k(\mathbf{c}_1)} = \sum_{j \in \mathcal{N}_k(\mathbf{c}_1)} w_j \; f_j \tag{4.15}$$

The weights $\{w_i\}$ are learned at training time. In our implementation, we chose $k = 100$ as neighbourhood size. The regression of the second contact point and the angle is performed by two different MLPs, as shown in Fig 4.4.

For what concerns the loss function, we set $\lambda = 0.1$ as contribution of the angle error such that a mistake of 1 degree in the angle rotation roughly has the same impact of an error of 1mm in the center displacement, as suggested in [10].

It is important to underline that the grasp regressor is *directly* fed with the output of the **contact sampler** such that the grasp prediction loss $\mathcal{L}_{grasp}$ can be backpropagated to the sampler as well.

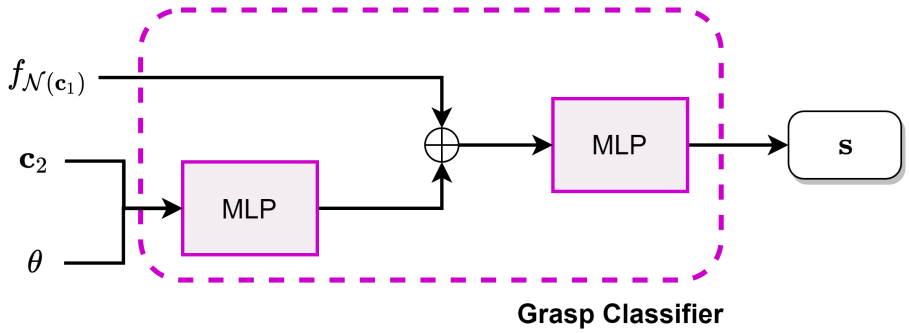

Figure 4.5. The grasp classifier implementation, inspired by [1]. A MLP expands the predicted contact point and angle to the same dimension of the first contact embedding. The second MLP takes the sum of the two embeddings and predicts the grasp score.

Grasp Classifier The grasp classifier needs to aggregate all the available information about each of the grasp components. For the first contact point, the feature extracted from the corresponding point are available. Since the second contact point is supposed to lie on the hidden surface of the shape, it cannot be linked to any of the available point embeddings. This is why, to combine the C-dimensional information of the first point embedding $f_{\mathbf{c}_1}$ to the 4-dimensional information related to the couple $(\mathbf{c}_2, \theta)$, a MLP is introduced to map such couple to a C-dimensional space. As shown in Fig. 4.5, the two embeddings are then summed up and given as input to a final MLP that reduces this information to the grasp score $\mathbf{s}$.

To properly give the classifier the ability to tell successful and unsuccessful grasp apart, it needs to be trained with both positive and negative examples in a fixed **balanced ratio**. It cannot be directly fed with the predicted grasps, since by design the module that predicts them is pushed towards regressing positive grasps. Moreover, to obtain the actual score for regressed grasps it would be needed a training-time simulation, which would end up drastically slowing down the process. For all these reasons, the classifier is fed with 1000 grasps coming directly from the ground truth, which are sampled to obtain a 3:7 positive:negative ratio. The classifier is then basically trained in parallel with respect to the regressor, still sharing the feature extractor module.

Chapter 5

Experiments

This chapter contains the detailed experimental pipeline and the results we obtained in a single-object grasp setting on the GPNet [1] dataset. Results are mainly presented through the help of tables and images that compare our method to the state-of-the-art techniques.

5.1 Dataset

At the time of the writing of this work, the state-of-the-art for end-to-end learning-based techniques on uncluttered scenes (i.e. single object scene) is GPNet. Our network architecture, discussed in chapter 4, inherits from theirs the *grasp heads* (the regressor and the classifier). To fairly compare our method with theirs, we relied on the same dataset. The dataset, generated through the physics simulator engine PyBullet, is composed of grasp sets for 225 different objects belonging to 8 categories (*bowl, bottle, mug, cylinder, cuboid, tissue box, sodacan, and toy car*) whose CAD models have been obtained from ShapeNetSem [41]. For each candidate, the following data is available:

- a set of $\sim$ 100k grasp annotations. For each grasp, it is provided
  - the grasp center $\mathbf{x}$;
  - the unit quaternion orientation $\mathbf{q}$;
  - the two contact points $\mathbf{c}_1$ and $\mathbf{c}_2$;
  - the cosine of the pitch orientation of the grasp θ;
  - the grasp score $\mathbf{s} \in \{0,1\}$, identifying whether the grasp was successful in simulation or not.

- about a thousand of different RGBD images, each one corresponding to a different view of the object;

It has to be underlined that the annotation of contact points in the dataset does not strictly follow the formalization followed so far in this work. In fact, probably due to the way data was generated in simulation, contact points of GPNet dataset do not directly belong to the grasped shape, but indeed are points in the space around the shape. A projection of such points onto the shape point cloud is then needed beforehand.

The dataset has been split in *training* set (195 objects) and *test* set (30 objects), roughly keeping the same category distribution.

YCB extension As repeatedly underlined in this work, the generalization power of the model should benefit from directly learning to sample contact points in place of complex heuristics. To validate our idea with concrete results, we created a smaller dataset with object models coming from the YCB dataset [42]. We generated data for 15 more objects, belonging both to similar (*mustard bottle*, *cracker box*) and very different (*tennis ball*, *screwdriver*) categories with respect to the original ones. We followed the grasp data generation pipeline suggested by GPNet authors.

5.2 Settings

5.2.1 Training Details

Throughout all the experiments we used PyTorch as our machine learning framework. The training is conduced on a NVIDIA 2080ti GPU. Models are trained from scratch with random initialization. Due to the nature of the dataset, which presents point cloud with different sizes, the batch size is set to 1 and all the batch normalization layers are disabled. Outliers in point clouds are pruned and each point cloud is unit-cube normalized before entering the network. The training happens thanks to an Adam optimizer with weight decay of 0.0001 and initial learning rate 0.0001, decreased by a factor of 2 every 100 epochs for 400 epochs. The threshold to consider a grasp sufficiently close to the object shape has been set to 3.5mm.

5.2.2 Evaluation Pipeline

The results reported in this chapter refer to the model at the last training epoch. For each of the test time shapes, the following pipeline has been executed:

1. the network outputs a number m of grasps in the $(\mathbf{c}_1, \mathbf{c}_2, \theta, \mathbf{s})$ parametrization;
2. the grasps with a *negative* score (i.e. $\mathbf{s} < 0.5$) are discarded;
3. the remaining grasp set is evaluated in terms of the grasp evaluation metrics introduced in section 3.1.2: *rule-based success* rate, *coverage* and *simulation success* rate. To better visualize the precision-coverage trade-off, each of those metrics is iteratively computed considering grasps in the top 10%, 30%, 50% and 100% with respect to the grasp score $\mathbf{s}$.

The low number of test shapes make single-view results quite variable. In fact, depending on the specific view, some objects have different intrinsic degree of grasping difficulty because the area with the highest density of contact points may not always be visible. To reduce such variability in the evaluation of the comparisons between different approaches, the results for a single model are averaged over 5 different views of the objects in the test set.

5.3 Results

This section lists the results obtained in the mentioned experiments. Everything is reported in comparison with GPNet. The main results, shown in Table 5.1, are related to the same setting of GPNet and are needed to give a ground for comparison between the two methods. Visualizations of test time grasps are available in Figure 5.1.

Before comparing the two methods, it should be considered that GPNet eventually reduces the huge quantity of predicted grasps through non-maximum-suppression [43], evaluating on average 10 grasps per shape. On the other hand, our method directly produces a limited quantity of grasps (about 100 per shape), thus such suppression is not needed.

Our methods outperforms GPNet in all the considered metrics. It is interesting to notice how L2G produces **simulation**-based performances more stable when extending the evaluation to grasps with lower score. Moreover, the ability to select an higher quantity of successful grasps allow L2G to largely improves with respect to the baseline.

Method	Feat Extractor	Rule-based success rate @k%			
		10	30	50	100
GPNet	PointNet++	93.30%	93.20%	82.00%	79.80%
	DeCo	87.33%	85.77%	82.91%	62.33%
L2G(ours)	DeCo	**94.89%**	**95.42%**	**95.25%**	**94.85%**

Method	Feat Extractor	Coverage rate @k%			
		10	30	50	100
GPNet	PointNet++	6.80%	14.40%	19.90%	30.70%
	DeCo	6.82%	14.33%	23.61%	32.21%
L2G(ours)	DeCo	**16.87%**	**24.74%**	**30.02%**	**36.82%**

Method	Feat Extractor	Simulation-based success rate @k%			
		10	30	50	100
GPNet	PointNet++	90.00%	76.10%	72.30%	58.80%
	DeCo	88.67%	**89.59%**	81.81%	57.97%
L2G(ours)	DeCo	**90.95%**	88.61%	**86.49%**	**79.36%**

Table 5.1. Grasp quality metrics comparison between L2G and GPNet. **N.B**: the PointNet++ results are reported from the original paper [1] since we never managed to reproduce their results.

Generalization evaluation on YCB extension Results on the YCB extension dataset are reported in Table 5.2. Our method outperforms the original one in most of the metrics. Metric values are generally lower than the ones obtained on GPNet dataset due to the domain shift.

Test execution time GPNet approach is based on sampling a huge quantity of contact candidates and prune most of them either trough custom geometric rules or modules specifically trained for this purpose. For each shape, around tens of thousands of candidates are generated. Moreover, to reduce the number of evaluated grasps at test time, generated grasps are reduced through *non maximum suppression*.

These operations are the major bottleneck of their architecture in terms of computational time. Our method, removing all the operations related to either pruning contact candidates or reducing the grasp proposals, is dozens of times faster, as shown in Table 5.3.

Method	Rule-based success rate @k%			
	10	30	50	100
GPNet	**82.22**%	74.34%	66.13%	53.55%
L2G(ours)	79.28%	**77.16**%	**77.30**%	**75.87**%

Method	Coverage rate @k%			
	10	30	50	100
GPNet	5.17%	15.11%	20.08%	25.52%
L2G(ours)	**13.17**%	**18.89**%	**23.95**%	**32.25**%

Method	Simulation-based success rate @k%			
	10	30	50	100
GPNet	**50.44**%	43.06%	36.85%	27.30%
L2G(ours)	44.64%	**44.26**%	**43.91**%	**43.56**%

Table 5.2. Grasp quality metrics comparison between L2G and GPNet on the YCB extension dataset.

Method	Execution time		
	min	max	mean
GPNet	1.18	44.04	10.24
L2G(ours)	**0.02**	**0.27**	**0.08**

Table 5.3. Grasp pose estimation test execution time (reported in seconds, per object) comparison between L2G and GPNet.

5.3.1 Variation Study

Among the different possible variants to the method, we present here the one which we think is worth exploring and will probably be part of future studies since it tries to overcome some of the limitations of the presented method. The purpose of this section is not to present a concrete and stable solution to such issue, but rather to highlight the improvable points of the method and provide a draft of possible direction.

Grasp classifier variation As shown in Figure 4.5, to combine the point features of the visible contact point $\mathbf{c}_1$ and the information coming from the second contact point $\mathbf{c}_2$ plus the angle θ a module which expands the

Variant	Rule-based success rate @k%			
	10	30	50	100
L2G - base	94.89%	95.42%	95.25%	94.85%
L2G - clf. var.	96.13%	96.06%	95.51%	94.52%

Variant	Coverage rate @k%			
	10	30	50	100
L2G - base	16.87%	24.74%	30.02%	36.82%
L2G - clf. var.	18.18 %	25.63%	30.36%	36.33%

Variant	Simulation-based success rate @k%			
	10	30	50	100
L2G - base	90.95%	88.61%	86.49%	79.36%
L2G - clf. var.	92.31%	91.47%	89.63%	83.33%

Table 5.4. L2G variant comparison with the baseline on the GPNet dataset.

4-dimensional information to a C-dimensional space is needed. The solution presented in our method relies on the implementation of the classifier presented in [1]. Nevertheless, it is not clear what such MLP introduced in the classifier is trying to learn, since the only input he sees is just a part of the grasp, which is not meaningful without the rest of the **context**. We provide a simple patch to this issue, giving the whole grasp $\mathbf{g} = (\mathbf{c}_1, \mathbf{c}_2, \theta)$ as input to the MLP. In this way, this small module needs to learn a mapping from the grasp $SE(3)$ to an higher dimensional feature space. Results of this experiment are reported in Table 5.4, showing an improvement that confirms that the classifier module has room for improvement.

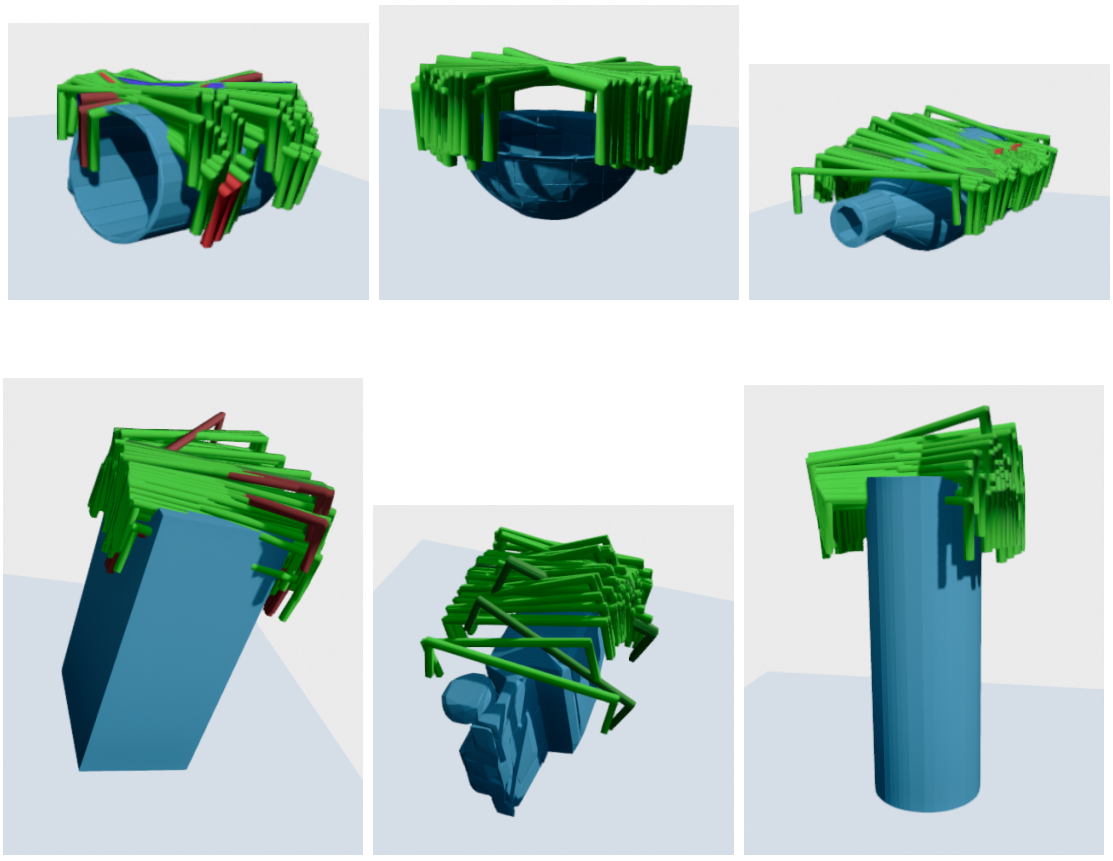

Figure 5.1. Visualization of grasps generated with L2G on test shapes. Green grasps are successfully executed, while the red ones are not.

Chapter 6

Conclusions

Throughout all this work, it has been underlined the gaining importance of robotics. In particular, we focused our attention on one specific task with which we, as humans, are challenged every day: grasping objects.

The growing availability of *Geometric Data* [44], along with increasing interest in creating robots able to embed human capabilities and the presence of simulators able to carefully produce grasp annotated data [33, 31], has opened the doors to designing learning pipeline for 6dof grasp pose estimation in the 3d world [38, 1, 9].

Even though excellent performances have already been obtained in terms of real world precision of generated grasps, the existing solutions are not free of custom constraints and heuristics specifically designed for the task at hand, which make it hard to extend such approaches to different settings, nor they are lightweight solutions. In fact, they typically need to generate and evaluate a lot of unsuccessful proposal to produce a meaningful number of successful ones. We saw a place for great improvement here, since most of the objects show clear and determined **pattern** in the distribution of their contact points, and this would make possible to learn where an object should be grasped from and directly focus on this area to generate poses.

Inspired by [11, 12], we built a novel architecture that leverages the power of differentiable sampling to select contact point candidates on the object shape and later regress grasp poses. The method has been tested against state-of-the-art solutions, showing improvements both in the observed grasp quality metrics and in execution times.

This work also poses the attention to some of the limitations of the current proposed solution, leaving open spots for further research that will be carried out in the nearest future. More specifically, the directions we are currently

exploring are:

- **multi modal sampling**: some objects show more than a single mode in the distribution of contact points (ex: a cylinder can be grabbed both from the top and from the side). Since the discontinuity of such distribution is not addressed by the sampling approach used in this work, introducing it would definitely help into having an higher grasp coverage;
- **1:N grasp truth mapping**: in our work, we assume that given a contact point $\mathbf{c}_1$ on the visible surface, it only exists one possible grasp related to it. This may not be always the truth.
- **real world testing**: this work lacks of real world execution of the proposed grasps, even though previous work have shown this translation is usually not hard to achieve;
- **grasp ranker**: in most of the grasp datasets, including the GPNet one, grasps are either labelled as successful or not. This is limiting, since there are grey areas in the between and having a more precise measure of the goodness/robustness of a grasp would actually train the classifier as a ranker rather than a binary classifier.
- **cluttered scene extension**: objects are rarely isolated form others or in a clean environment. Nevertheless, one of the main strengths of the method is its by design extensibility to different settings.

Bibliography

[1] C. Wu, J. Chen, Q. Cao, J. Zhang, Y. Tai, L. Sun, and K. Jia, "Grasp proposal networks: An end-to-end solution for visual learning of robotic grasps," 2020.

[2] E. Ahmed, A. Saint, A. E. R. Shabayek, K. Cherenkova, R. Das, G. Gusev, D. Aouada, and B. Ottersten, "A survey on deep learning advances on different 3d data representations," 2019.

[3] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2017.

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," 2017.

[5] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," 2019.

[6] F. Pistilli, G. Fracastoro, D. Valsesia, and E. Magli, "Learning graph-convolutional representations for point cloud denoising," 2020.

[7] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 3734–3743, PMLR, 09–15 Jun 2019.

[8] A. Alliegro, D. Valsesia, G. Fracastoro, E. Magli, and T. Tommasi, "Denoise and contrast for category agnostic shape completion," 2021.

[9] A. Mousavian, C. Eppner, and D. Fox, "6-dof graspnet: Variational grasp generation for object manipulation," 2019.

[10] C. Eppner, A. Mousavian, and D. Fox, "A billion ways to grasp: An evaluation of grasp sampling schemes on a dense, physics-based grasp data set," 2019.

[11] I. Lang, A. Manor, and S. Avidan, "Samplenet: Differentiable point cloud sampling," *CoRR*, vol. abs/1912.03663, 2019.

[12] O. Dovrat, I. Lang, and S. Avidan, "Learning to sample," *CoRR*, vol. abs/1812.01659, 2018.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[14] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[15] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," 2014.

[16] C. M. Jiang, D. Wang, J. Huang, P. Marcus, and M. Nießner, "Convolutional neural networks on non-uniform geometrical signals using euclidean spectral transformation," 2019.

[17] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, p. 18–42, Jul 2017.

[18] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," 2018.

[19] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," 2020.

[20] S. Altmann, *Rotations, Quaternions, and Double Groups*. Dover books on mathematics, Dover Publications, 2005.

[21] I.-M. Chen and J. Burdick, "Finding antipodal point grasps on irregularly shaped objects," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 4, pp. 507–512, 1993.

[22] V.-D. Nguyen, "Constructing force-closure grasps," in *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1368–1373, 1986.

[23] C. Ferrari and J. Canny, "Planning optimal grasps," in *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pp. 2290–2295 vol.3, 1992.

[24] J. Mahler, B. Hou, S. Niyaz, F. T. Pokorny, R. Chandra, and K. Goldberg, "Privacy-preserving grasp planning in the cloud," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, p. 468–475, IEEE Press, 2016.

[25] K. Kleeberger, R. Bormann, W. Kraus, and M. Huber, "A survey on learning-based robotic grasping," *Current Robotics Reports*, vol. 1, p. 239–249, 12 2020.

[26] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps

with synthetic point clouds and analytic grasp metrics," 2017.

[27] A. Depierre, E. Dellandréa, and L. Chen, "Jacquard: A Large Scale Dataset for Robotic Grasp Detection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Madrid, Spain), Oct. 2018.

[28] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," *CoRR*, vol. abs/1509.06825, 2015.

[29] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *CoRR*, vol. abs/1603.02199, 2016.

[30] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *CoRR*, vol. abs/1806.10293, 2018.

[31] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning." `http://pybullet.org`, 2016–2020.

[32] B. O. Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[33] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified particle physics for real-time applications," *ACM Trans. Graph.*, vol. 33, jul 2014.

[34] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 1180–1189, JMLR.org, 2015.

[35] S. Jenni and P. Favaro, "Self-supervised feature learning by learning to spot artifacts," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[36] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," *CoRR*, vol. abs/1505.05192, 2015.

[37] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *J. Mach. Learn. Res.*, vol. 17, p. 2096–2030, jan 2016.

[38] A. ten Pas, M. Gualtieri, K. Saenko, and R. P. Jr., "Grasp pose detection in point clouds," *CoRR*, vol. abs/1706.09911, 2017.

[39] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi, "The farthest point strategy for progressive image sampling," *IEEE Transactions on Image Processing*, vol. 6, no. 9, pp. 1305–1315, 1997.

[40] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3d

object reconstruction from a single image," *CoRR*, vol. abs/1612.00603, 2016.

[41] M. Savva, A. Chang, and P. Hanrahan, "Semantically-enriched 3d models for common-sense knowledge," pp. 24–31, 06 2015.

[42] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[43] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 3, pp. 850–855, 2006.

[44] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.