Master Thesis : Implicit neural representations for robotic grasping

Auteur : Gustin, Julien

Promoteur(s) : Louppe, Gilles

Faculté : Faculté des Sciences appliquées

Diplôme : Master en science des données, à finalité spécialisée

Année académique : 2022-2023

URI/URL : https://arxiv.org/abs/2304.08805; http://hdl.handle.net/2268.2/17624

University of Liège
School of Engineering and Computer Science

Implicit neural representations for robotic grasping

A dissertation submitted in partial fulfillment of the requirements for the degree of
Master of Science in Data Science

Author
Julien Gustin

Advisor
Pr. Gilles Louppe

Academic year 2022-2023

Abstract

Robotic grasping is a fundamental skill in many robotic applications. While most grasping methods excel in constrained tasks within structured environments. When operating in more complex and uncertain scenarios, handling uncertainty becomes essential. Bayesian frameworks provide a means to address this uncertainty but require prior knowledge about the grasping pose. However, previous research has demonstrated that using a uniform prior over the workspace is highly inefficient.

In this work, we propose a novel approach that exploits implicit neural representations to construct scene-dependent priors. This enables the application of powerful simulation-based inference algorithms to determine plausible and successful grasp poses in unstructured environments.

We demonstrate the significant improvements achieved by incorporating this informative prior. Specifically, our model achieves an impressive success rate of 97% in grasping a single object, surpassing the performance of the previous model. Additionally, we reduce acquisition time by 60% by capturing only a partial view of the scene and training a neural network to reconstruct the complete scene. Furthermore, in the more complex scenario of multi-object grasping, our model achieves a success rate of 91.37% in simulation and 95.6% in real-world scenarios, comparable to benchmark models. These results demonstrate the effectiveness of our approach and its impressive sim2real transfer capabilities.

We also provide valuable explainability by examining the predicted posterior distribution. This provides a better understanding of the uncertainty associated with the estimation of the grasping pose, enhancing the transparency of the system's decision-making process.

Acknowledgments

First of all, I would like to express my deepest gratitude to my supervisor, Professor Gilles Louppe, for his exceptional guidance and for instilling in me a passion for deep learning throughout my master's degree. I am truly grateful for everything he has done for me.

I would also like to thank Norman Marlier for the help he gave me during my thesis. He was always there for me whenever I had questions or needed assistance. His advice played a crucial role, and without him, my master's thesis wouldn't have been the same. I am also grateful to him for giving me the opportunity to publish our work and for the memorable experiences we shared in London.

I would also like to thank my friend Joachim Houyon, who was an exceptional project partner during my studies, and my friend Julien Simar, whose invaluable help and support have been with me since the start of my university journey.

Special thanks to the members of the Computer Science Students (CSS). The pleasant moments we shared made my master's years even more enriching.

Lastly, I would like to express my sincere gratefulness to my parents for providing me with the right conditions to pursue my studies, and for their support. I am also grateful to Marcelline for her constant support in the most difficult and uncertain moments of my life. To all my friends, thank you for your presence and friendship, which have been a source of strength and inspiration at every stage of my life.

Contents

1 Introduction 1
1.1 Problem statement . . . 2
1.2 Contributions . . . 3

Part I Implicit representations of 3D scenes 5

2 Neural implicit representations 6
2.1 Related work . . . 6
2.1.1 Explicit representations . . . 6
2.1.2 Implicit Neural Representations . . . 7
2.2 Occupancy-based Implicit Representations . . . 10
2.2.1 Input processing . . . 11
2.2.2 Architectures . . . 13

3 Experiments 22
3.1 Dataset . . . 22
3.2 Segmentation . . . 23
3.3 Implicit representation . . . 29
3.3.1 Evaluation protocol . . . 29
3.3.2 Quantitiative analysis . . . 30
3.3.3 Qualitative analysis . . . 32

4 Discussion 38

Part II Robotic grasping 39

5 Grasping as inference 40
5.1 Notations . . . 40
5.2 Probabilistic modeling . . . 41
5.3 Neural ratio estimation . . . 42
5.4 Scene-dependent Prior . . . 44
5.4.1 Orientation . . . 44
5.4.2 Positions . . . 44

6 Grasping pipeline 46
6.1 Density ratio estimation . . . 47
6.1.1 Architectures . . . 47

6.2 Maximum a posteriori estimation 51

7 Experiments **53**

7.1 Experiment protocol 53

7.2 Training analysis 54

7.3 Models evaluation 55

7.3.1 Simulation results 55

7.3.2 Real-world results 58

7.4 MAP estimate 59

7.5 Prior and posterior analysis 60

7.5.1 Prior over the position 60

7.5.2 Posterior 63

7.6 Ablation study 68

7.7 Discussion 69

8 Conclusion **70**

A Setup **75**

B Implicit representation priors meet Riemannian geometry for Bayesian robotic grasping **76**

C Multiresolution IsoSurface Extraction **86**

D Additional figures **87**

Chapter 1

Introduction

Robotic systems heavily rely on the ability to grasp objects. However, current methods are mainly effective for highly constrained tasks in structured environments. When attempting to operate outside these limits, dealing with uncertainty about the environment is necessary. To tackle this challenge, probabilistic models are needed. Among the available probabilistic modeling methods, Bayesian inference stands out as a promising approach to handle these uncertainties.

Bayesian inference is a method that updates initial prior beliefs about parameters θ based on new observations x using Bayes' theorem. The updated belief, known as the posterior, is given by the equation:

$$p(\theta \mid x) = \frac{p(x \mid \theta)}{p(x)} p(\theta).$$

In this equation, $p(x \mid \theta)$ represents the likelihood, $p(\theta)$ represents the prior, and $p(x)$ is a marginalization factor called evidence.

When applied to the complex task of grasping unknown objects, the parameters correspond to the hand pose, and the observed data to the success of a grasp, generally conditioned on a scene representation or additional information. Having an informative prior on potential grasping poses becomes crucial, especially for learning-based algorithms that require a certain amount of successful grasps to learn from examples, typically sampled from a prior distribution. Although a uniform prior may appear objective, it proves to be highly inefficient, particularly when objects occupy only a small portion of the workspace.

To address these issues, this research builds on recent advancements in implicit neural representations to introduce an informative, scene-dependent prior. This prior is learned through training a neural network to represent a 3D scene. By capturing valuable information about objects from a single observation, the network provides prior knowledge that indicates that the grasping point should be within an object present in the scene, rather than anywhere in the workspace. This significantly reduces the sampling area, focusing on a relevant region of interest. Integrating this learned prior into Bayesian inference aims to enhance the grasping process for robotic systems operating in uncertain and unstructured environment.

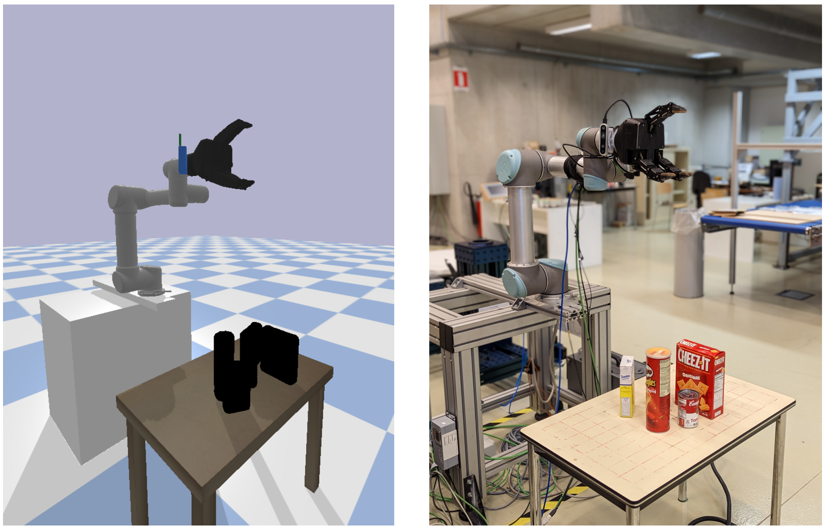


Figure 1.1. Our benchmark scene. (left) The simulated environment. (right) The real setup.

1.1 Problem statement

This work addresses the problem of grasping, which involves clearing a table of unknown objects by repeatedly grasping one object at a time using a three-finger gripper. The benchmark scene is illustrated in Figure 1.1. For more information about the setup, please refer to Appendix A.

Building on previous research on simulation-based inference for robotic grasping [1, 2], this master's thesis aims to improve the previous model by introducing an informative scene-dependent prior for grasping pose using neural implicit representations.

The previous model represented the posterior of the hand configuration as follows:

$$p(\mathbf{h} \mid S, \mathbf{V}) = \frac{p(S \mid \mathbf{h}, \mathbf{V})}{p(S \mid \mathbf{V})} p(\mathbf{x} \mid \mathbf{V}) p(\mathbf{q}),$$

where $\mathbf{h} := (\mathbf{x}, \mathbf{q})$ denotes the hand pose, composed of a position $\mathbf{x} \in \mathbb{R}^3$ and an orientation $\mathbf{q}$, $S \in \{0, 1\}$ indicates whether a grasp is successful, and $\mathbf{V} \in \mathbb{R}^{N \times N \times N}$ is a voxel grid representing the scene. In this model, the prior over the grasping position is uniform over the voxel axis-aligned bounding box of objects. While this model has produced promising results for a single object, it does not scale well to multiple objects, since the greater the number of objects placed on the table, the larger the bounding box becomes, reducing the efficiency of the approach. In addition, obtaining the voxel grid using a truncated signed distance function requires a complete scan of the scene, resulting in a time-consuming process that takes ~ 1 minute to capture depth images from various viewpoints.

To overcome these limitations, the model now assumes that the grasping point should lie within the occupied region of an object. This is achieved by conditioning the posterior on $o = 1$, where $o \in \{0, 1\}$ indicates whether a point x is inside an object. The posterior distribution is given by:

$$p(\mathbf{h} \mid S, o = 1, \mathbf{P}) = \frac{p(S \mid \mathbf{h}, o = 1, \mathbf{P})}{p(S \mid o = 1, \mathbf{P})} p(\mathbf{x} \mid o = 1, \mathbf{P}) p(\mathbf{q}),$$

with $\mathbf{P} \in \mathbb{R}^{N \times 3}$ being a partial point cloud of the scene.

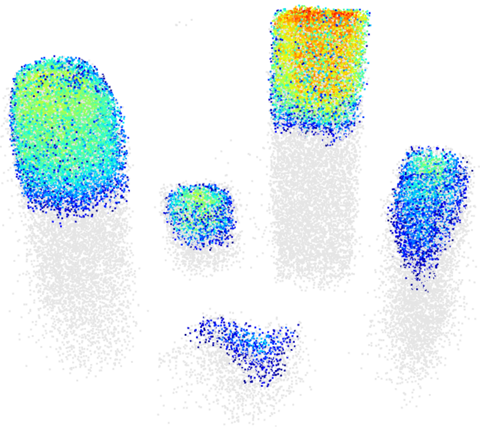


Figure 1.2. 3D visualization of the prior and posterior distributions of potential grasping points. Grasping points sampled from $p(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$ are colored with a gradient from blue to red, indicating their relative posterior values. Grey points represent grasping points sampled from the prior distribution $p(\mathbf{x} \mid o = 1, \mathbf{P})$, obtained from the same scene as depicted in Figure 1.1 in the real setup.

This assumption is natural and generally holds for most non-convex objects, as it provides a clear region of interest for the grasping rather than assuming the grasping point can be anywhere on the table. In order to obtain this prior about the position we need a way to represent any 3D scene and determines the occupancy of a point. This can be done using implicit neural representations. By taking advantage of this assumption, we can improve the quality of grasp samples when constructing the dataset of $(\mathbf{h}, S)$ pairs for a scene using this prior. This introduces a novel sampling procedure adaptable to various configurations, as long as sample generation is feasible using a simulator. Furthermore, by using a partial point cloud instead of a voxel grid, a single depth image of the scene is needed, significantly speeding up the acquisition stage and reducing its memory consumption. Figure 1.2 visually illustrates the prior and posterior distribution.

- **Part I** of the thesis extensively reviews the literature on implicit representation. Various methods are carefully compared and evaluated to identify the most effective and suitable approaches to our problem.
- **Part II** is dedicated to the integration of the selected implicit representation method into a robotic grasping pipeline. Additionally, input features with different level of knowledge about the scene are compared and assessed to determine the most relevant information to provide to the model.

1.2 Contributions

The contributions of this work can be summarized as follows:

- Comparison, evaluation, and integration of implicit representation priors.
- Speeding up computation by working with a partial view of the scene instead of a full 3D scan.
- Validation of the method through simulated and real experiments, demonstrating promising grasping performance.

Scientific contribution During my master's thesis, I had the opportunity of co-writing a paper with Norman Marlier, which was directly related to my work [3]. This paper was published as part of a workshop entitled "Geometric Representations: The Roles of Screw Theory, Lie Algebra, & Geometric Algebra," held at ICRA 2023. Our work was presented in London, where we had the chance to share our findings. The complete paper and the poster created for the workshop can be found in the appendix (see Appendix B).

Part I

Implicit representations of 3D scenes

Chapter 2

Neural implicit representations

Deep learning for 3D data is becoming increasingly important in a variety of fields, such as robotics, autonomous vehicles, and virtual reality. In these domains, achieving precise and accurate representations of the surrounding environment is crucial.

This chapter delves into the concept of 3D representations, exploring both classical explicit methods and more modern implicit methods that harness the power of deep learning. By examining these approaches, we aim to provide an overview of the diverse strategies employed in 3D representation. Furthermore, we conduct an in-depth analysis and comparison of various architectures of occupancy-based models all re-implemented in TensorFlow [4], with the aim of providing a detailed understanding of the strengths and limitations associated with each approach.

Back to our problem, we need to find a suitable representation for our scence-dependent prior $p(\mathbf{x} \mid o = 1, \mathbf{P})$. This prior can be rewritten as follows:

$$p(\mathbf{x} \mid o = 1, \mathbf{P}) = \frac{p(o = 1 \mid \mathbf{x}, \mathbf{P})p(\mathbf{x})}{p(o = 1 \mid \mathbf{P})}.$$

In the equation above, we only require to define the distributions $p(o = 1 \mid \mathbf{x}, \mathbf{P})$ and $p(\mathbf{x})$, as explained in more detail in Section 5.4.2. The last term is a uniform over the workspace, while the first term involves estimating whether a given 3D point $\mathbf{x}$ is located inside an object within the scene representation $\mathbf{P}$. It is essential for this estimation to provide not just a binary answer, but a probability value that fits within the Bayesian inference framework. Additionally, the estimation should be differentiable to facilitate optimization (refer to Section 6.2). This leads us to explore methods of representing a 3D scene in a probabilistic manner, which can be achieved using neural implicit representation, particularly occupancy networks.

2.1 Related work

2.1.1 Explicit representations

In the traditional approach to 3D reconstruction using deep learning, explicit representations such as point clouds [5–7], voxel grids [8–10], and meshes [11–13] have been widely used. These representations allow for convenient readability and manipulation, but they also have their limitations. Figure 2.1 compare these three representations visually.

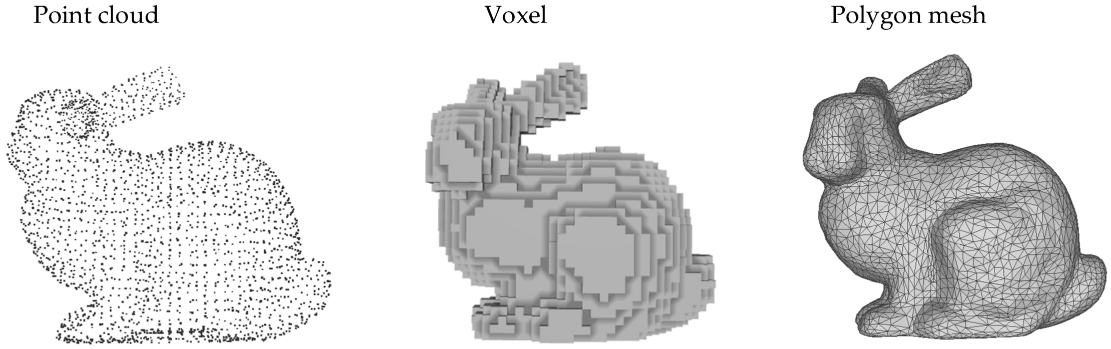


Figure 2.1. Comparison of 3D representation techniques for a rabbit model: point cloud, voxel grid, and polygon mesh. While point clouds accurately represent the shape of the model, voxel grids provide higher resolution at the cost of increased memory usage. Meshes are efficient in memory usage and can capture fine geometry, but may be challenging to create and manipulate.

Point Clouds Point clouds $\mathbf{P} \in \mathbb{R}^{3\times N}$ consist of 3D points and can accurately represent any complex shape, but they require a significant amount of points to adequately capture the scene. However, this format alone is not very useful as it does not include information about the surface or any other properties of the scene.

Voxel Grids Voxel grids $\mathbf{V} \in \mathbb{R}^{N\times N\times N}$ are another accurate 3D representation that can represent any scene with a desired level of resolution. However, the memory requirement grows quadratically with the resolution, which can be a significant limitation in deep learning-based systems.

Meshes Meshes are commonly used in graphics and CAD software and consist of a collection of vertices, edges, and faces. They can be useful in capturing complex geometry and are efficient in terms of memory usage. However, current methods struggle to accurately represent complex and water-tight topology [13].

2.1.2 Implicit Neural Representations

Implicit Neural Representations are parameterized functions that can represent signals like images, audio, and 3D shapes. Unlike conventional 3D representations that are typically discrete, a 3D implicit neural representation is a continuous function that maps coordinates to specific properties such as occupancy, opacity, or color [14–16]. These representations cannot be derived analytically or expressed as mathematical formulas due to their complexity, so state-of-the-art approaches employ neural networks to approximate them.

One notable advantage of implicit representations is their capability to achieve infinite resolution, allowing for detailed representation at any level of granularity. Additionally, some methods allow the reconstruction of complete 3D shapes from partial or noisy observations [14, 17]. This ability to deal with imperfect input data makes implicit representation methods well suited to applications where input data is noisy or incomplete, such as in real robotics [3] or object recognition tasks.

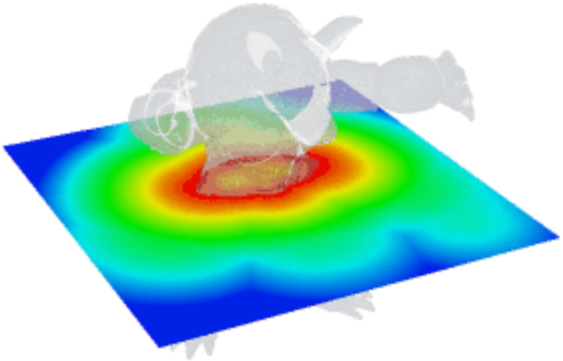


Figure 2.2. The Signed Distance Function (SDF) applied to a 3D representation of an armadillo. The SDF values are visualized using a color gradient, ranging from blue to red. The warmer colors indicate proximity to the surface, which is sliced along the z-axis. Image source: open3D.

As Vincent Sitzmann an assistant professor at the MIT noted, neural scene representation is *the way neural networks learn to represent information about our world.*[1]

Signed distance functions A Signed Distance Function (SDF) [18] of a specific 3D object determines the orthogonal distance from a given point $\mathbf{x} \in \mathbb{R}^3$ to the surface of the 3D shape, with SDF value's sign indicating whether the point $\mathbf{x}$ lies inside or outside the shape (See Figure 2.2). Traditional methods of approximating a SDF involve discretizing 3D space into a grid of voxels, where each voxel is labeled with the distance to the nearest surface. An example of such an approximation is the Truncated Signed Distance Field (TSDF) [19]. However, the use of voxels is computationally expensive and memory-intensive, limiting the scalability of this approach as well as it is tied to a fixed shape, and it is limited to representing a single shape.

Recent advances in deep learning have introduced methods that overcome these limitations by leveraging neural networks to approximate SDF, namely DeepSDF. [16]. Instead of relying on voxel grids, these methods sample points near the object's surface and train a neural network to predict the corresponding SDF values for those points. This approach offers the advantage of representing entire classes of shapes, rather than being restricted to a fixed shape. A slight improvement from Chabra et al. [20] consist in discretizing the scene into local features, which reduces the complexity of the neural network and enables faster inference. Moreover, this method solves the problem of computationally expensive memory usage associated with traditional voxel-based methods.

In conclusion, the combination of signed distance functions (SDFs) and deep learning has demonstrated immense potential in representing 3D geometry. However, it is important to note that this approach has its limitations. It currently works best for a single object and does not incorporate crucial inductive bias such as translation or rotation equivariance. This could result in difficulties in generalizing to novel object shapes and orientations, making the process of representing such objects more challenging.

Neural radiance fields Neural Radiance Field (NeRF) [15] is another way for representing a scene using a fully-connected deep neural network. In contrast to previous

[1] https://github.com/vsitzmann

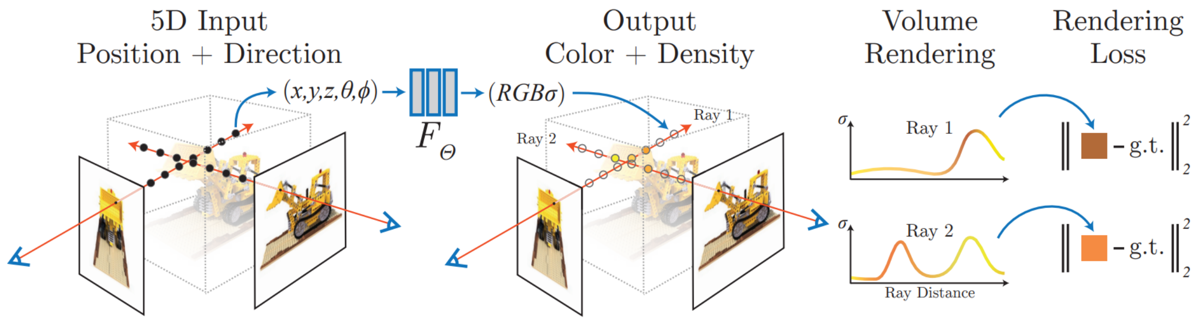


Figure 2.3. A visual summary of NeRF [15].

methods [14, 21–24], NeRF predicts the volume density/opacity $\sigma \in \mathbb{R}$ and emitted color $\mathbf{c} \in \mathbb{R}^3$ at every spacial location $\mathbf{x}$ and viewing direction $\mathbf{d}$ using a single neural network. The volume density $\sigma(x, y, z)$ can be seen as the differential probability of a ray that terminates at an infinitesimal particle at position (x, y, z). The approximation of the expected color $C(\mathbf{r})$ of camera ray $\mathbf{r}(t)$, which is computed from an origin $\mathbf{o}$ and a direction $\mathbf{d}$, within the bound $[t_n, t_f]$

$$C(\boldsymbol{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt$$

where

$$T(t) = exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

is the probability that the ray doesn't hit any other particle in a range from t_n to t, also denoted as the accumulated transmittance along the ray. This integral is estimated using quadrature, yielding an estimation $\hat{C}(\mathbf{r})$. To map the inputs to the corresponding volume density and emitted color, an MLP $F_\theta : (\boldsymbol{x}, \boldsymbol{d}) \rightarrow (\boldsymbol{c}, \sigma)$ is optimized (in practice, two networks are trained to capture different levels of detail: a "coarse" network and a "fine" network). To optimize these networks, the following loss function is used, which measures the total squared error between the rendered and true pixel colors:

$$\mathcal{L} = \sum_{r \in \mathcal{R}} \left[||\hat{C}_c(\mathbf{r}) - C(\mathbf{r})||_2^2 + ||\hat{C}_f(\mathbf{r}) - C(\mathbf{r})||_2^2 \right],$$

where $\mathcal{R}$ is the set of rays in each batch, and $C(\mathbf{r})$, $\hat{C}_c(\mathbf{r})$, and $\hat{C}_f(\mathbf{r})$ represent the ground truth, coarse volume-predicted, and fine volume-predicted RGB colors for ray $\mathbf{r}$, respectively. Once trained, to generate a synthetic image, the following operations must be performed:

1. Each pixel of that image sends a ray through the scene.
2. Query the MLP at each location, where the ray traverses the scene with the viewing direction to get a vector of colors and densities.
3. Use classical volume rendering algorithm to render these colors and the density into a pixel.

Despite its high accuracy, NeRF model has a significant limitation, it is computationally expensive and time-consuming, taking up to one or two days to optimize for a single scene using a powerful NVIDIA V100 GPU, according to the authors [15]. Moreover,

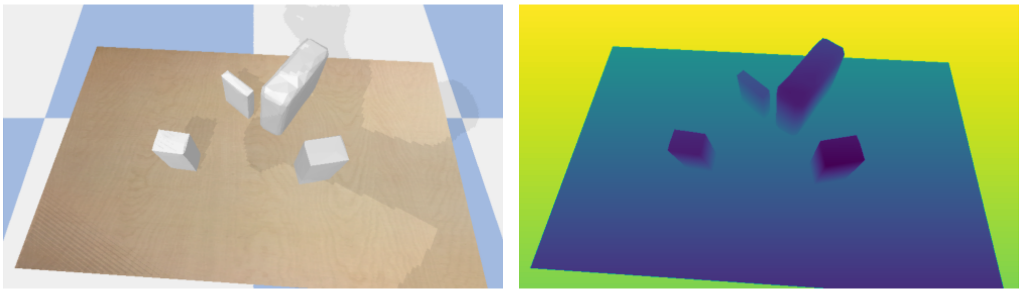


Figure 2.4. Comparison between depth images and RBG images taken in the simulated environment.

NeRF requires a large number of images captured from different perspectives to achieve high-quality results, and it does not generalize to any scene as it is trained to represent a specific scene. However, recent advancements in the field of 3D modeling have led to the development of faster alternatives, such as Instant NGP [25] by NVIDIA.

2.2 Occupancy-based Implicit Representations

Considering our specific requirements, we have decided to focus on occupancy-based implicit representation methods [14] to represent $p(o = 1 \mid \mathbf{x}, \mathbf{P})$. These methods consist in a parameterized function f_θ which takes a 3D point coordinate $\mathbf{x} \in \mathbb{R}^3$, and a representation of the scene (in our case, a partial point cloud $\mathbf{P}$) as inputs. The function then outputs a real number that represents the probability of occupancy ($o = 1$) at the given point, as determined by the scene. Mathematically, we express the occupancy network as:

$$f_\theta : \mathbb{R}^3 \times \mathcal{P} \rightarrow [0, 1].$$

One of the main advantages of using occupancy-based models is that they provide a probability that aligns well with the principles of Bayesian inference. By reframing this problem as a classification task to determine whether $\mathbf{x}$ lies within an object of a scene $\mathbf{P}$ or not, this model is trained with the objective of minimizing the binary cross-entropy (BCE) loss $\mathcal{L}$:

$$\mathcal{L}_B(\theta) = \frac{1}{|B|} \sum_{i=1}^{|B|} \sum_{j=1}^{T} \mathcal{L}(f_\theta(\mathbf{x}_{ij}, \mathbf{P}_i), o_{ij}).$$

Where B represents a batch sampled from the dataset, and o_{ij} indicates whether the j-th query point $\mathbf{x}_{ij}$ lies within an object in the point cloud $\mathbf{P}_i$.

Point cloud representation was chosen for two main reasons. Firstly, it enables faster acquisition using only a depth image of the scene (partial view). Secondly, since our training process is based on simulated data, but we intend to deploy the model in real-world scenarios, we needed a representation that bridges the sim2real gap. To this end, we used a simulator based on PyBullet [26], which provides a powerful simulation environment despite its limited realism. Notably, depth images were chosen over RGB images as they offer a closer approximation to reality (see Figure 2.4). These depth images are then used to segment the objects and generate the corresponding point cloud representation.

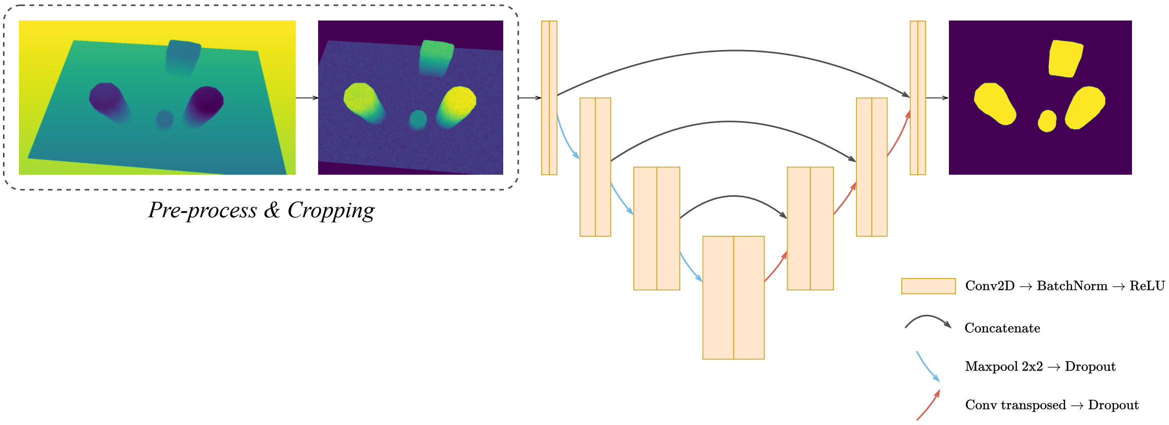


Figure 2.5. The segmentation pipeline consists of preprocessing a raw depth image obtained from a depth camera with known extrinsic and intrinsic parameters. The image is first projected into a new coordinate system, then cropped and normalized. Next, the preprocessed image is fed into a UNet model, which accurately segments the objects.

2.2.1 Input processing

Objects point cloud segmentation The segmentation of the point cloud of interest from depth images is a critical task, and to achieve this, the UNet architecture [27] has been investigated as a means to accurately segment objects from the background. Figure 2.5 show the pipeline of the segmentation task.

Before feeding the raw depth images into the UNet, we applied several preprocessing techniques to improve the accuracy and speed of the segmentation process.

- Firstly, we projected the point cloud from the camera frame to the workspace frame, ensuring that the z component was set to zero on the table. This step enabled the rendered depth image to be independent of the camera pose, thus providing robustness to the segmentation process.
- Secondly, we cropped the acquired depth image, which initially had a shape of 480×848, to a smaller size of 480×576. This was necessary due to memory limitations during processing and also to focus on the part of the image containing the workspace.
- Lastly, we standardized the data by setting the mean to zero and the standard deviation to one. This technique has been shown to enhance the convergence rate of neural networks during training [28, 29].

The process is carried out entirely through simulation, enabling direct access to the segmentation mask and eliminating the need for manual annotation of depth images. Since the same model will be used in real-world scenarios, we applied data augmentation techniques, including image rotation, zooming, and cropping, to enhance the model's ability to handle the sim2real transition. We also introduced noise to the camera position and depth image, similar to Mahler et al. [30].

We also investigated the use of the RANSAC method [31] to fit planes in the 3D point cloud. As applied to our problem, we could detect the table and set a threshold z at that height, then cutoff the point cloud beyond that threshold. However, due to the inherent noise in real depth images, this method did not perform well.

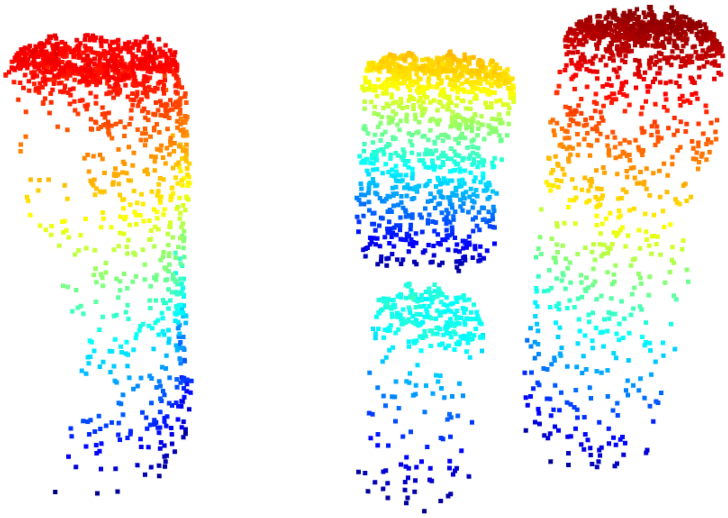


Figure 2.6. The point cloud obtained after extracting and converting the segmented objects from a depth image.

Depth image to point cloud

After obtaining an accurate segmentation of the objects, their corresponding point cloud are then extracted to generate a point cloud solely containing the objects of interest (Figure 2.6). The conversion of a depth image into a point cloud involves computing the 3D coordinates (x, y, z) for each pixel (u, v) along with its corresponding depth value d. To achieve this, we can use the following set of equations

$$\begin{aligned} x &= \frac{(u - c_x)d}{f_x} \\ y &= \frac{(v - c_y)d}{f_y} \\ z &= d \end{aligned}$$

Here, c_x and c_y represent the coordinates of the principal point, typically located close to the center of the image. f_x and f_y represent the focal lengths of the camera, and d represents the depth value at pixel (u, v). By utilizing both intrinsic and extrinsic camera parameters, we can accurately compute the 3D coordinates for each pixel and create a complete point cloud from the depth image.

Once we have computed the 3D coordinates for each pixel, we can represent the point cloud as a set of points $\mathbf{P} = \mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{N-1}$, where $\mathbf{p}_i = [x_i, y_i, z_i]$ is the 3D position of the i-th point in the cloud, and N is the total number of points.

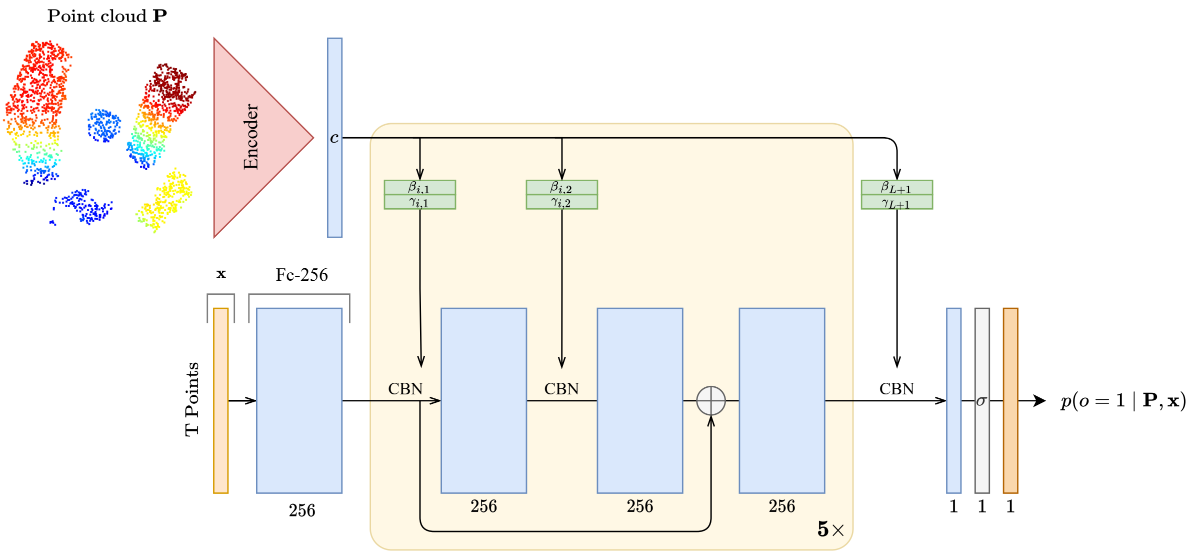

Point cloud $\mathbf{P}$
Encoder
c
$\beta_{i,1}$ $\gamma_{i,1}$
$\beta_{i,2}$ $\gamma_{i,2}$
β_{L+1} γ_{L+1}
$\mathbf{x}$
Fc-256
T Points
256
CBN
256
CBN
256
256
CBN
σ
1
1
1
$p(o = 1 \mid \mathbf{P}, \mathbf{x})$
5×

Figure 2.7. The ONet is an encoder-decoder architecture. The encoder converts a point cloud $\mathbf{P}$ to global features $\mathbf{c} \in \mathbb{R}^{C''}$, which are then fed to the decoder along with T query points. The decoder consists of 5 modified residual blocks that use Conditional Batch Normalization to condition the input by the point cloud embedding. Finally a fully connected layer with sigmoid activation represents the probability of each point $\mathbf{x}$ being inside an object given $\mathbf{P}$.

2.2.2 Architectures

The following architectures presented in this section were re-implemented entirely in TensorFlow due to the unavailability of their source code or the absence of an existing implementation using this framework.

Occupancy network (ONet)

The ONet [14] architecture consists of a PointNet-based [5] encoder and a decoder, as illustrated in Figure 2.7.

The encoder is an adapted PointNet that takes inputs of shape $N \times C$, N denotes the number of points in the point cloud $\mathbf{P}$ and C is the feature size, initially the latter corresponds to the 3 coordinates x, y, and z. The encoder begins with a fully connected layer that yields features for each point, with dimensions C'. Subsequently, these features pass through an adapted residual block five times, each consisting of a residual block [32] followed by max pooling. The output have the shape $1 \times C''$, and are then expanded to $N \times C''$ and concatenated with the output of the residual block. This procedure is repeated five times, followed by a final residual block and max pooling, and fully connected layers that output features of shape C'' that globally represent $\mathbf{P}$. A comprehensive illustration of the encoder architecture is provided in Figure 2.8.

The ONet's decoder is provided with the output features from the encoder along with a batch of T 3D coordinates $\mathbf{x}$. These query points undergo a fully connected layer to generate a 256-dimensional feature vector for each point, followed by five modified residual-blocks that uses Conditional Batch Normalization (CBN) [33, 34] to modulate the input query point using the features of the point cloud.

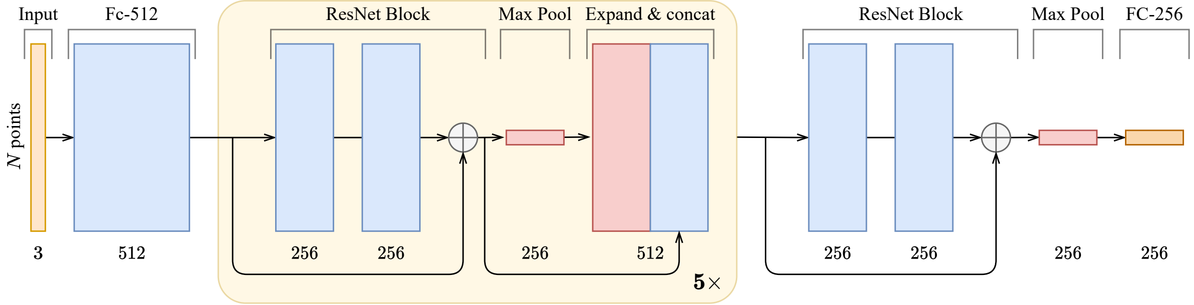


Figure 2.8. PointNet-based encoder that takes a point cloud of N-point as input. The encoder consists of a fully connected layer, a modified residual block with additional pooling and expansion layers repeated five times, a final residual block, and maxpooling. The global features of the point cloud are obtained by the last fully connected layer, which outputs 256 features.

Based on Mescheder et al. [14] we use a CBN to condition the input query points to the point cloud embedding. CBN is an adaptation of the well-known Batch Normalization [28] that normalizes an input $\mathbf{x}$ using the mean and standard deviation over the current batch, with learnable parameters γ and β, introduced to accelerate and facilitate the convergence of neural networks. However, in CBN, γ and β parameters are replaced with $\hat{\gamma} = \gamma + \Delta\gamma$ and $\hat{\beta} = \beta + \Delta\beta$, where the offsets $\Delta\gamma$ and $\Delta\beta$ are learned from conditional data, i.e., the point cloud embedding, using a multi-layer perceptron (MLP). Specifically, we obtain:

$$\Delta\beta = \mathrm{MLP}_\beta(\mathbf{c}) \qquad \text{and} \qquad \Delta\gamma = \mathrm{MLP}_\gamma(\mathbf{c}).$$

Where $\mathbf{c} = \mathrm{Encoder}(\mathbf{P})$ and the MLP used in CBN has one hidden layer with 256 neurons. The CBN operation can be expressed mathematically as:

$$y = \frac{\mathbf{x}' - \mathrm{E}[\mathbf{x}']}{\sqrt{\mathrm{Var}[\mathbf{x}'] + \epsilon}}\hat{\gamma} + \hat{\beta}$$

with $\mathbf{x}'$ being a feature representation of the query point $\mathbf{x}$.

While this method is promising, it is important to note that there are two major limitations that must be considered. Firstly, the embedding of the point cloud captures the global structure of the represented shape, making it difficult to learn correlations between the latent code and fine-grained 3D structural details of the shape. As a result, the generated shapes can be oversimplified and may struggle to represent thin details.

Secondly, in the case of robotic, objects do not have a predetermined pose and can be placed anywhere on a table with any rotation. While data augmentation can help with training, there is a lack of rotation or translation equivariance that could potentially help with faster training and be more robust to novel pose and objects.

Vector neurons occupancy network (VN-ONet)

The Vector Neurons (VN) framework, proposed by Deng et al. [35], provides a novel approach for creating rotation equivariant neural networks to process point clouds. This framework achieves rotation equivariance by replacing classical operations, such as fully

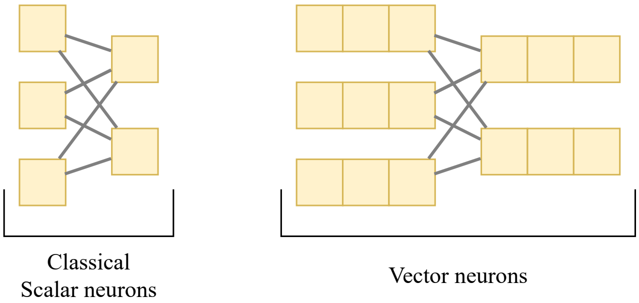


Figure 2.9. Comparison between classical scalar neurons and vector neurons.

connected layers, pooling, non-linear activation, and batch normalization, with an upgraded version that uses vector neuron representations.

In this framework, classical neuron representations are replaced with vector neurons (Figure 2.9), which are vectors $\mathbf{v} \in \mathbb{R}^3$, rather than a scalar $z \in \mathbb{R}$. A layer of neurons is represented as a matrix $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_C]^\top \in \mathbb{R}^{C\times 3}$, where C is the number of vector neurons in the layer. For a point cloud of N points, $\mathbf{x} \in \mathbb{R}^{N\times 3}$, a collection of N such vector-list features is obtained, denoted as $\mathcal{V} = \{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_N\} \in \mathbb{R}^{N\times C\times 3}$.

Similar to a neural network, the number of channels between layers can be changed via a mapping function $f(\mathcal{V}^{(d)}; \theta)$, where $\mathcal{V}^{(d)} \in \mathbb{R}^{N\times C^{(d)}\times 3}$ is the input to layer d and θ represents the learnable parameters of the layer. This mapping function produces an output tensor $\mathcal{V}^{(d+1)} \in \mathbb{R}^{N\times C^{(d+1)}\times 3}$.

In this framework, Deng et al. [35] provide several layers that replace the traditional ones including:

- **Linear layers**: A linear operation $f_{lin}(., \mathbf{W})$ is applied to the input tensor $\mathbf{V} \in \mathcal{V} \in \mathbb{R}^{N\times C\times 3}$ using a weight matrix $\mathbf{W} \in \mathbb{R}^{C'\times C}$, as follows:

$$\mathbf{V}' = f_{lin}(\mathbf{V}; \mathbf{W}) = \mathbf{W}\mathbf{V} \in \mathbb{R}^{C'\times 3} \tag{2.1}$$

  This is in contrast to the classical operation $\mathbf{x}\mathbf{W}^\top + \mathbf{b}$. It is straightforward to verify that a rotation matrix $R \in SO(3)$ commutes with this linear layer:

$$f_{lin}(\mathbf{V}R; \mathbf{W}) = \mathbf{W}\mathbf{V}R = f_{lin}(\mathbf{V}; \mathbf{W})R = \mathbf{V}'R$$

  This corresponds to the definition of rotation equivariance: $f(\mathbf{V}R) = f(\mathbf{V})R$, where $R \in SO(3)$. Note that the bias term in Eq. 2.1 had been omitted since it would interfere with equivariance.

- **Non-linear activation**: Activation functions are crucial for the success of neural networks, as they introduce non-linearity into the system and enable the input domain to be mapped differently into two half-spaces. For vector neurons, a 3D version of these non-linearities is required. However, using a fixed frame of reference, such as the standard coordinate system, would violate equivariance. To address this issue, Deng et al. [35] proposed a VN-ReLU that dynamically predicts a direction from the input vector-list feature and truncates the portion of a vector that points into the negative half-space of the learned direction. The non-linear VN-ReLU is implemented by applying a weight matrix $\mathbf{U} \in \mathbb{R}^{1\times C}$ to each input vector channel

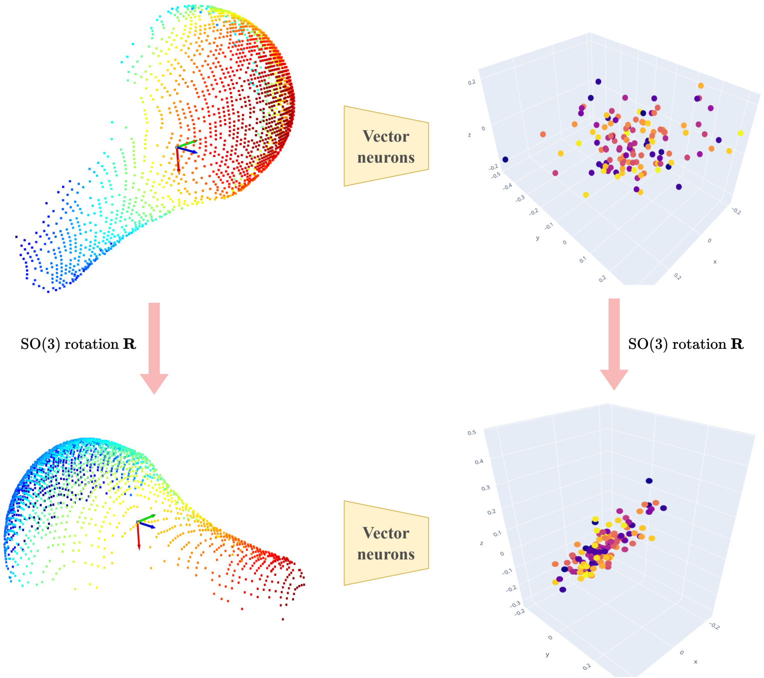


Figure 2.10. This figure illustrates the rotation equivariance properties of vector neurons. A partial point cloud of an object and the same object after being rotated by $R \in SO(3)$ are both processed by a VN-Pointnet (Figure 2.11). The resulting feature spaces are compared, revealing that the feature space of the rotated object is equivalent to the feature space obtained by rotating the original features with the same matrix R.

$\mathbf{v} \in \mathbf{V}$ to obtain a direction $\mathbf{k} \in \mathbb{R}^{1\times 3}$, where $\mathbf{k} = \mathbf{UV}$. The activation function is then defined as follows:

$$\mathbf{v}' = \begin{cases} \mathbf{v} & \text{if } \langle \mathbf{v}, \mathbf{k} \rangle \geq 0, \\ \mathbf{v} - \langle \mathbf{v}, \frac{\mathbf{k}}{||\mathbf{k}||} \rangle \frac{\mathbf{k}}{||\mathbf{k}||} & \text{otherwise.} \end{cases} \tag{2.2}$$

This approach allows for the application of a 3D version of non-linear activation functions $f_{relu}(\mathbf{V}) = [\mathbf{v}']_{c=1}^{C}$, while preserving equivariance.

- **Invariant layers**: This layer is based on the property that the product of an equivariant signal $\mathbf{V} \in \mathbb{R}^{C\times 3}$ by the transpose of another equivariant signal $\mathbf{T} \in \mathbb{R}^{C'\times 3}$ is rotation invariant. In other words, $(\mathbf{VR})(\mathbf{TR})^\top = \mathbf{V}RR^\top\mathbf{T}^\top = \mathbf{VT}^\top$. Deng et al. [35] leveraged this property to produce rotation invariant features by generating a coordinate system $\mathbf{T} \in \mathbb{R}^{C'\times 3}$ from $\mathbf{V}$ and applying it to $\mathbf{V}$. To implement this, for each $\mathbf{V}_n$ in $\mathcal{V} \in \mathbb{R}^{N\times C\times 3}$, a matrix $\mathbf{T}_n$ is produced by applying the VN-linear function to $\mathbf{V}_n$ with a target number of channels C'. Specifically, $\mathbf{T}_n$ is defined as:

$$\mathbf{T}_n := \text{VN-linear}(\mathbf{V}_n).$$

Finally, the invariant layer is defined as:

$$\text{VN-In}(\mathbf{V}_n) := \mathbf{V}_n\mathbf{T}_n^\top.$$

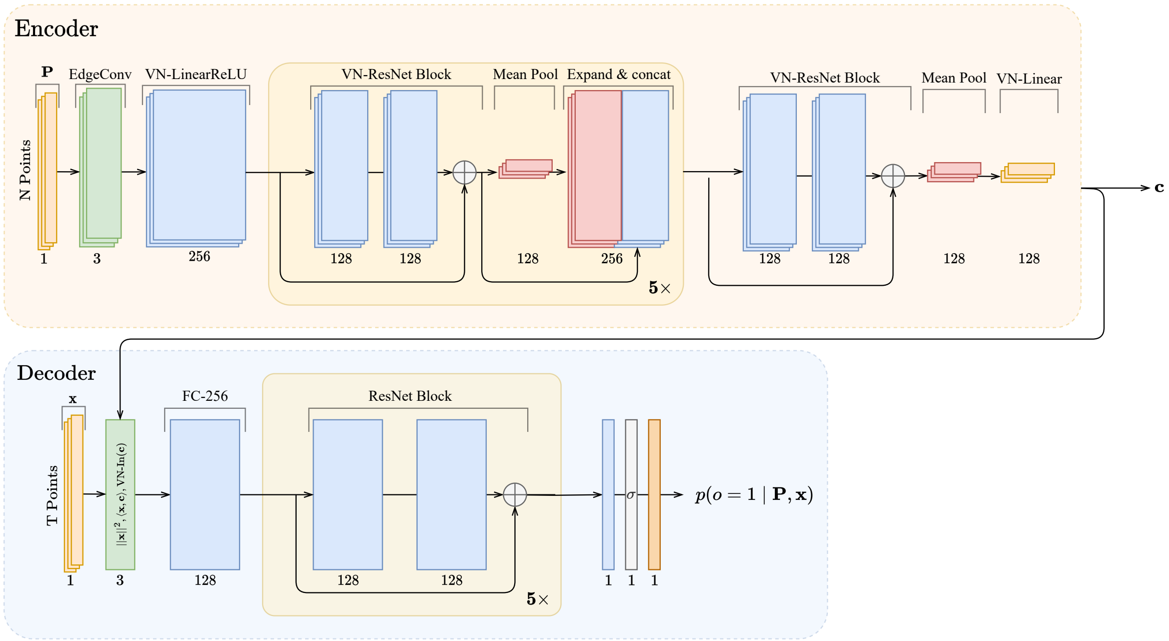


Figure 2.11. The VN-ONet architecture is shown here, which employs a VN-Pointnet encoder to convert a point cloud with N points into equivariant features $\mathbf{c}$. These features are then used to generate three invariant features by merging them with T query points $\mathbf{x}$. The resulting features are passed through a fully connected layer, followed by 5 residual blocks, another fully connected layer, and a sigmoid function, to obtain $p(o = 1 \mid \mathbf{P}, \mathbf{x})$.

Above are presented the most important layers for this work. For additional and more detailed information on these layers, please refer to the original paper [35].

Building on top of these layers we can create an upgraded version of the occupancy network as seen previously, VN-ONet (Figure 2.11). The VN-Pointnet [35] encoder is quite similar to the structure of PointNet of the ONet [14], as discussed in the previous section, albeit with updated vector neurons layers. However, a notable difference is the introduction of the edge convolution (EdgeConv) layer. This layer is included to avoid degeneracy, which occur if f_{lin} is applied to a set of point cloud coordinates $\mathbf{V}_n \in \mathbb{R}^{1\times3}$. The resulting set of $\mathbb{R}^{C\times3}$ vector-lists would have all its vector components being linearly dependent. To mitigate this problem, the EdgeConv layer selects the k nearest neighbors for each point and computes three features:

- The relative euclidean distance between a point and its k neighbors.
- The k neighbors themselves.
- The cross product between the current point and each of its k neighbors.

This yields features of shape $N \times C \times 3 \times k$, and a mean pool operation is applied to the last dimension to obtain features of shape $N \times C \times 3$.

The output of the encoder, denoted as $\mathbf{c}$, is passed to a rotation-invariant decoder together with query points $\mathbf{x}$. To achieve rotation-invariance, three invariant features are created: $||\mathbf{x}||^2$, $\langle \mathbf{x}, \mathbf{c} \rangle$, and VN-In($\mathbf{c}$). These features are then input to a (classical) 5-residual block followed by a final fully-connected layer, which outputs the probability of occupancy.

Although the rotation equivariant aspect of this model is interesting, it encounters issues concerning global feature representation and handling complex scenes containing multiple objects. Moreover, the advantages of rotation equivariance may not apply when object orientations vary randomly since it's very unlikely for a collection of objects to possess identical poses, but rotated by R, where $R \in SO(3)$. Furthermore, the model's high computational and memory usage demands may impede its performance, leading to slower processing times than alternative models.

Convolutional occupancy network (Conv-ONet)

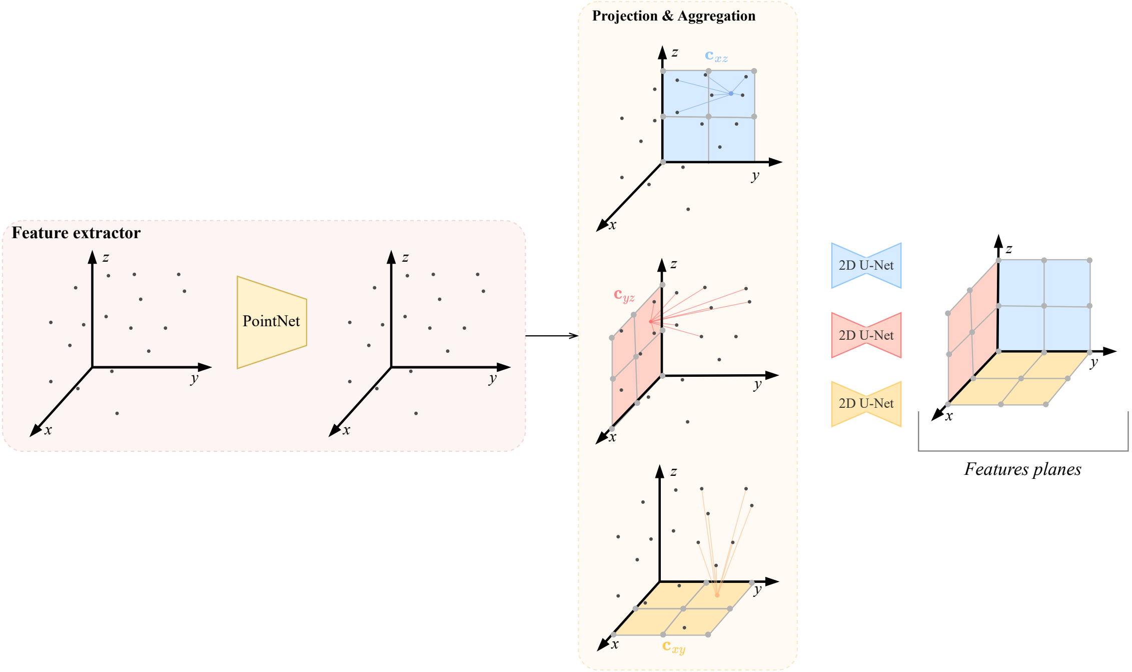


Figure 2.12. The convolutional occupancy network encoder uses a PointNet with local pooling to map 3D coordinates to the feature space. The resulting features are then projected and aggregated onto the 3 canonical planes xy, yz, and xz using a small grid (e.g., in this example, a resolution of 2×2). Finally, these feature planes are projected using a shared UNet models to aggregate local and global information. For simplicity in this figure $c_{**}(\mathbf{P})$ is written as c_{**}.

The architecture proposed by Peng et al. [17] is specifically designed to address issues of previous models such as inadequate global feature representation and lack of inductive bias, such as translation. In contrast to other neural network architectures [14, 35], this architecture makes it possible to obtain information on local features by discretizing the 3D space using three canonical planes of a given resolution onto which the features of each point are projected. Moreover it exploits the translation equivariance property of convolutional neural networks [36].

The encoder consists of a fully connected layer followed by 5 residual blocks similar to the PointNet of the occupancy network [14] (Figure 2.8). However, instead of pooling over individual points, the points falling within the same voxel of the 3D grid are aggregated. These features are then projected onto three canonical planes: $\mathbf{c}_{xy}, \mathbf{c}_{xz}$ and $\mathbf{c}_{yz}$, based on the points' original coordinates. Finally, a shared 2D U-Net is applied to each feature plane to aggregate local and global information. A schematic of the architecture is shown in Figure 2.12.

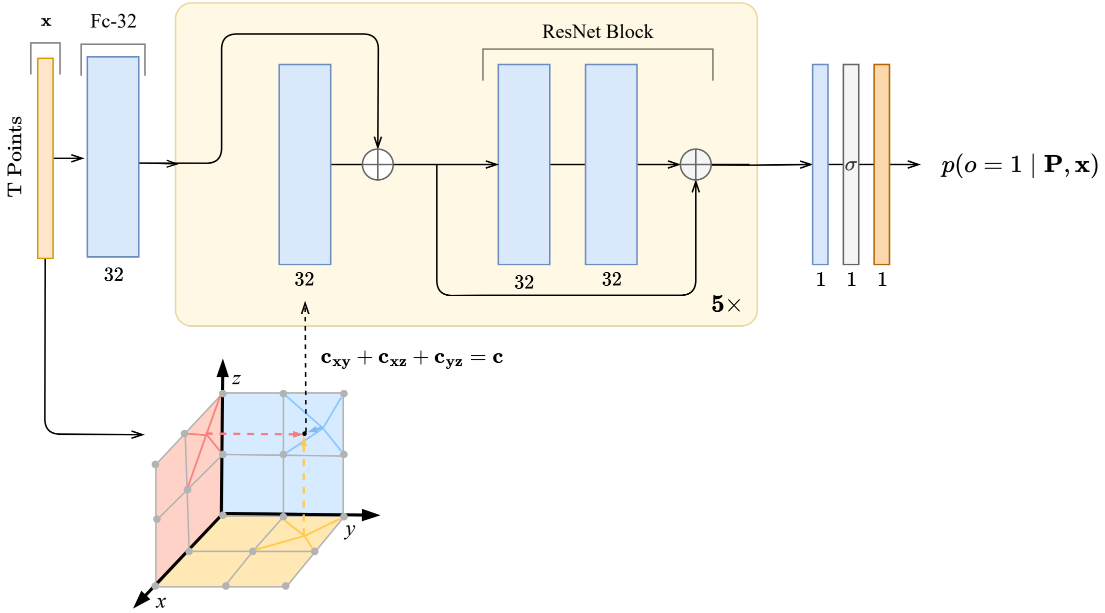

x
Fc-32
T Points
32
ResNet Block
32
32
32
5×
σ
1
1
1
$p(o = 1 \mid \mathbf{P}, \mathbf{x})$
$\mathbf{c_{xy}} + \mathbf{c_{xz}} + \mathbf{c_{yz}} = \mathbf{c}$
z
y
x

Figure 2.13. The convolutional occupancy network decoder, where $\mathbf{c}_{**}(\mathbf{P})(\mathbf{x})$ is simplified as $\mathbf{c}_{**}$ for better readability.

The occupancy probability at a given point $\mathbf{x} \in \mathbb{R}^3$ given $\mathbf{P}$ is determined by aggregating the features $\mathbf{c}_{xy}(\mathbf{P})(\mathbf{x})$, $\mathbf{c}_{xz}(\mathbf{P})(\mathbf{x})$, and $\mathbf{c}_{yz}(\mathbf{P})(\mathbf{x})$ from the corresponding locations in the three features maps, where $\mathbf{c}_{**}(\mathbf{P})(\mathbf{x})$ represents a bilinear interpolation that computes the feature at coordinate $\mathbf{x}_{**}$ using the discretized feature plane $\mathbf{c}_{**}(\mathbf{P})$. The resulting features are then summed to form the feature vector $\mathbf{c}$, as follows:

$$\mathbf{c} = \mathbf{c}_{xy}(\mathbf{P})(\mathbf{x}) + \mathbf{c}_{xz}(\mathbf{P})(\mathbf{x}) + \mathbf{c}_{yz}(\mathbf{P})(\mathbf{x}).$$

Subsequently, the point coordinates $\mathbf{x}$ pass through a fully connected layer, followed by a block that incorporates the feature vector $\mathbf{c}$ at each iteration. In each iteration, the output of the previous block is added to the output of the current block, resulting in a refined feature vector that encodes both local and global features. This process is repeated five times, and the hidden layer has a feature dimension of 32 (Figure 2.13).

Although this model incorporates interesting properties such as local and global information and translational equivariance, the major drawback comes from the slowness of operations applied to feature planes such as aggregation or interpolation.

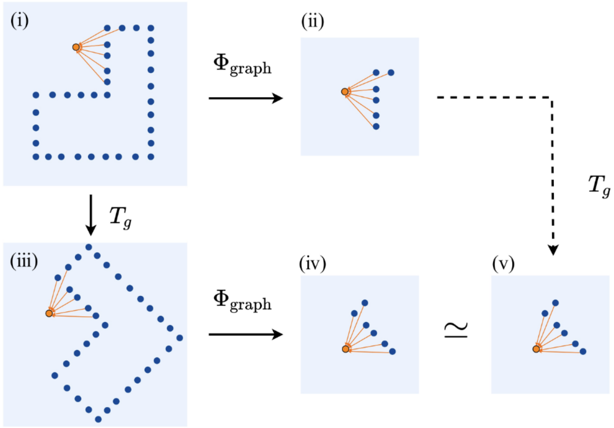


Figure 2.14. In this illustration, the concept of rotation equivariance is demonstrated by querying the k ($k = 6$) nearest neighbors (k-NN) of a query point (colored orange) from a point cloud $\mathbf{P}$. The k-NN operation generates a graph which exhibits equivariance in rotation to a transformation T_g applied to $\mathbf{P}$. This is expressed mathematically as $T_g(\Phi_{\text{graph}}(\mathbf{X})) = \Phi_{graph}(T_g(\mathbf{P}))$ (Figure from [37]).

Graph occupancy network (Graph-ONet)

Chen et al. [37] proposed a novel approach for implicit representation that leverages the 3D equivariant property of graphs, which are rotation, scaling, and translation invariant (see Figure 2.14). This property enables generalization to unseen transformations, which is important in the real world where objects are never in a fixed canonical pose.

To encode rotation equivariance through the layers, the authors adapted the vector neurons from Deng et al. [35] by incorporating rotation invariance. They introduced a hybrid feature equivariant layer (HL) (Figure 2.15), which takes scalar feature $\mathbf{s} \in \mathbb{R}^C$ and vector features $\mathbf{V} \in \mathbb{R}^{C\times 3}$ as input and outputs scalar feature $\mathbf{s}' \in \mathbb{R}^{C'}$ and vector features $\mathbf{V}' \in \mathbb{R}^{C'\times 3}$. This method is more effective and efficient than using only vector neurons since the vector $\mathbf{V}$ encodes rotation equivariance, while the scalar feature $\mathbf{s}$ is rotation invariant, resulting in more computational efficiency and transferring some of the learning responsibility to the scalar features via standard neural layers.

The encoder/decoder architecture follows a multiscale approach. In the encoder phase, features are extracted from the point cloud corresponding to the k nearest points and their Euclidean distances, forming a graph structure (refer to Figure 2.14). These features then pass through two hybrid linear layers, which extract both invariant features $\mathbf{s}_l$ and equivariant features $\mathbf{V}_l$. Subsequently, max pooling is applied along the neighbors dimension. This process maps each point of the point cloud to a latent dimension, resulting in features $\mathbf{V}_l \in \mathbb{R}^{C_l\times 3\times N}$ and $\mathbf{s}_l \in \mathbb{R}^{C_l\times N}$, where l represents the current scale. To capture both local and global information, this process is repeated two more times using subsets of points obtained through farthest point sampling (FPS) from the point cloud. This iterative approach expands the receptive fields and enables feature extraction from different scales. The initial scale consists of N points, which gradually decreases to N/4 and N/16, resulting in a total of three pairs of $\mathbf{V}_l$ and $\mathbf{s}_l$ (see Figure 2.16).

In the decoding stage of the architecture, a query point $\mathbf{x}$ is taken as input and its k nearest neighbors in the original point cloud are retrieved at different scaling factors. The retrieved features, along with the corresponding point from the point cloud and

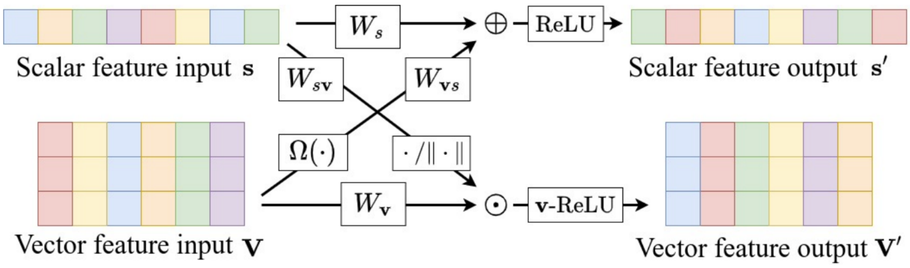

Scalar feature input $\mathbf{s}$
W_s
$W_{s\mathbf{v}}$
$W_{\mathbf{v}s}$
$\Omega(\cdot)$
$\cdot/\|\cdot\|$
$W_{\mathbf{v}}$
ReLU
v-ReLU
Scalar feature output $\mathbf{s}'$
Vector feature input $\mathbf{V}$
Vector feature output $\mathbf{V}'$

Figure 2.15. Hybrid feature equivariant layers. Visualization of how vector and scalar features share information in linear and nonlinear layers. Vector features go through an invariant layers Ω that is added to the scalar part. Scalar features are transformed with normalizing $\cdot/||\cdot||$ to scale the vector feature channels (Figure and caption from [37]).

their relative distance from the query point $\mathbf{x}$, are concatenated, similar to the encoding process. At each stage, the concatenated features go through a series of HN (Hybrid Neural Networks), which are aggregated and sent to a five-residual-block. The output from this block is then fed into a fully connected layer, followed by a sigmoid function that outputs the probability of occupancy. For more detailed information, please refer to the original paper [38] or the source code.

Although this method has strengths such as its 3D equivariant inductive bias and the integration of local and global information, it has certain limitations. It requires a complete representation of the point cloud and imposes significant memory requirements, restricting scalability in terms of point cloud size and limiting the potential size of hidden layers.

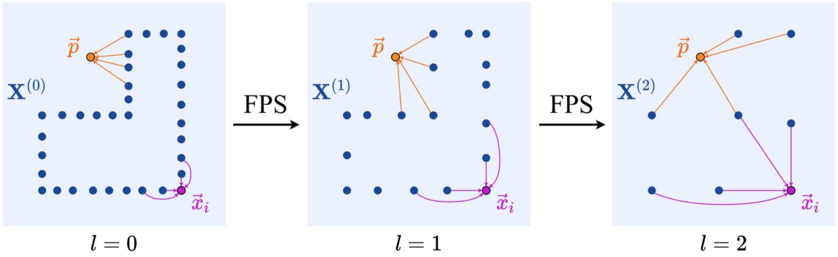


Figure 2.16. Farhest point sampling (FPS). This figure illustrates the various stages of FPS sampling used during the encoding process to extract local and global features (depicted in purple), as well as during the decoding phase with a query point p (depicted in orange).(Figure from [37])

Chapter 3

Experiments

3.1 Dataset

Figure 3.1. A subset of the YCB benchmark objects [39].

The dataset used in this study comes from Breyer et al.'s work [40] and consists of a blend of established datasets such as the YCB benchmark [39] (Figure 3.1). It is made up of a mixture of synthetic mesh objects divided into 303 training and 40 testing objects. Breyer et al. [40] defined two distinct scenarios, "pile" and "packed" (Figure 3.2). In the first scenario, objects are successively placed higher up at random locations in the workspace, resulting in the formation of a stack of objects. The second scenario consists of iteratively placing a subset of higher objects vertically at random locations in the workspace, while avoiding positions that lead to collisions with objects already placed.

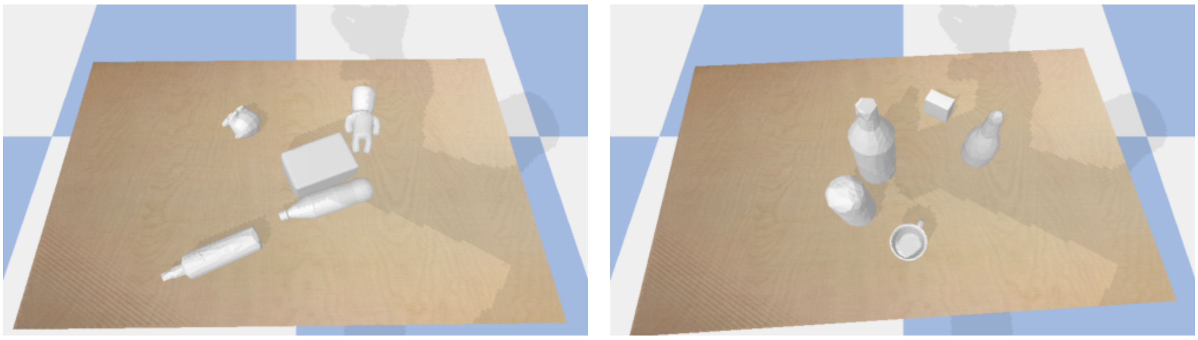


Figure 3.2. Example of a "pile" (left) and a "packed" (right) scene.

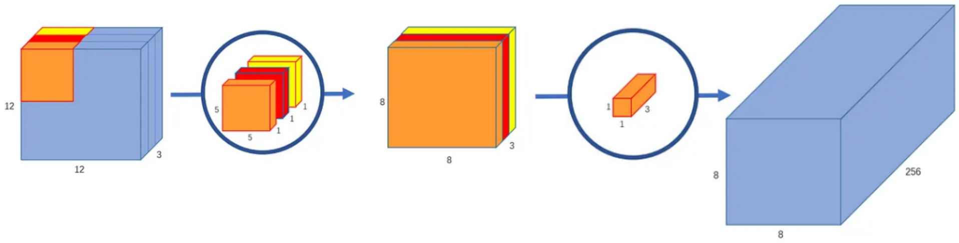


Figure 3.3. Depth-wise convolution applies a single filter to each input channel separately. This operation produces a set of feature maps, one for each input channel. Next, point-wise convolution applies a 1x1 convolution to combine the output from the depth-wise convolution. This 1x1 convolution acts as a linear combination of the features produced by the depth-wise convolution, and it is used to mix the feature maps into a single output. The advantage of using depth-wise separable convolution is that it can significantly reduce the number of parameters and computational cost compared to traditional convolution. Figure from [41].

During the data generation process, following the methodology described by the authors, m objects, where $m \sim Pois(4)$, are randomly selected, with repetition, in the respective train/test subset. Both the packed and pile scenarios are used with equal probability of 50% to ensure robustness and generalization to any situation. A depth image of the scene is captured from a fixed position of the camera, and a point cloud of the object is extracted, as explained in Section 2.2.1. Then, to maintain a fixed number of points in the point cloud ($N = 2048$), we sub-sampled or over-sampled random points of the point cloud.

3.2 Segmentation

In this section, three different architectures are compared: two UNet models (described in Section 2.2.1) and a TNet model [41]. The first UNet, called UNet 5 64, consists of five layers with progressively increasing numbers of features (64, 128, 256, 512, and 1024) in the encoding path, and symmetrically decreasing numbers of features (1024, 512, 256, 128, 64) in the decoding path. The second UNet model, named UNet 4 32, starts with 32 features and doubles the number of features at each downscaling layer for the first four layers, followed by four upscaling layers where the number of features is divided by two at each stage. The T-Net, on the other hand, is specifically designed for constrained and mobile devices, achieving accurate results with significantly fewer parameters compared to state-of-the-art models. One notable distinction between the T-Net and traditional UNet models is the use of depth-wise separable convolution [42], which separates the standard convolution operation into two separate steps: depth-wise convolution and point-wise convolution (refer to Figure 3.3).

Each model is trained on a dataset of $250,000$ depth and mask images pairs, augmented using techniques shown in Figure 3.4. The learning process aims to minimize the binary cross-entropy loss, using Adam optimizer with a learning rate of 10^{-4}. Prior to learning, images are normalized in a per image strategy. Model performance is then evaluated using two independent sets of $50,000$ samples consisting of unseen scenes and objects, without any augmentation.

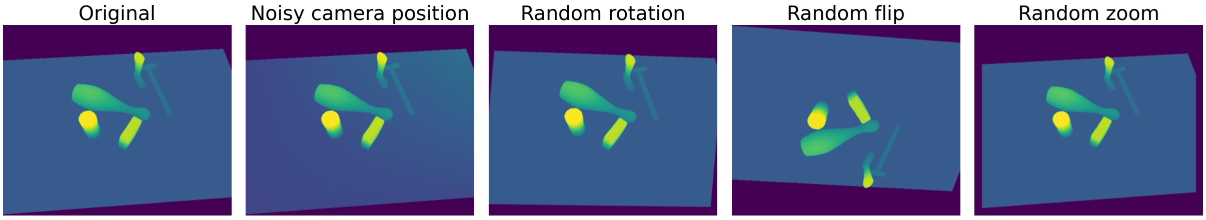


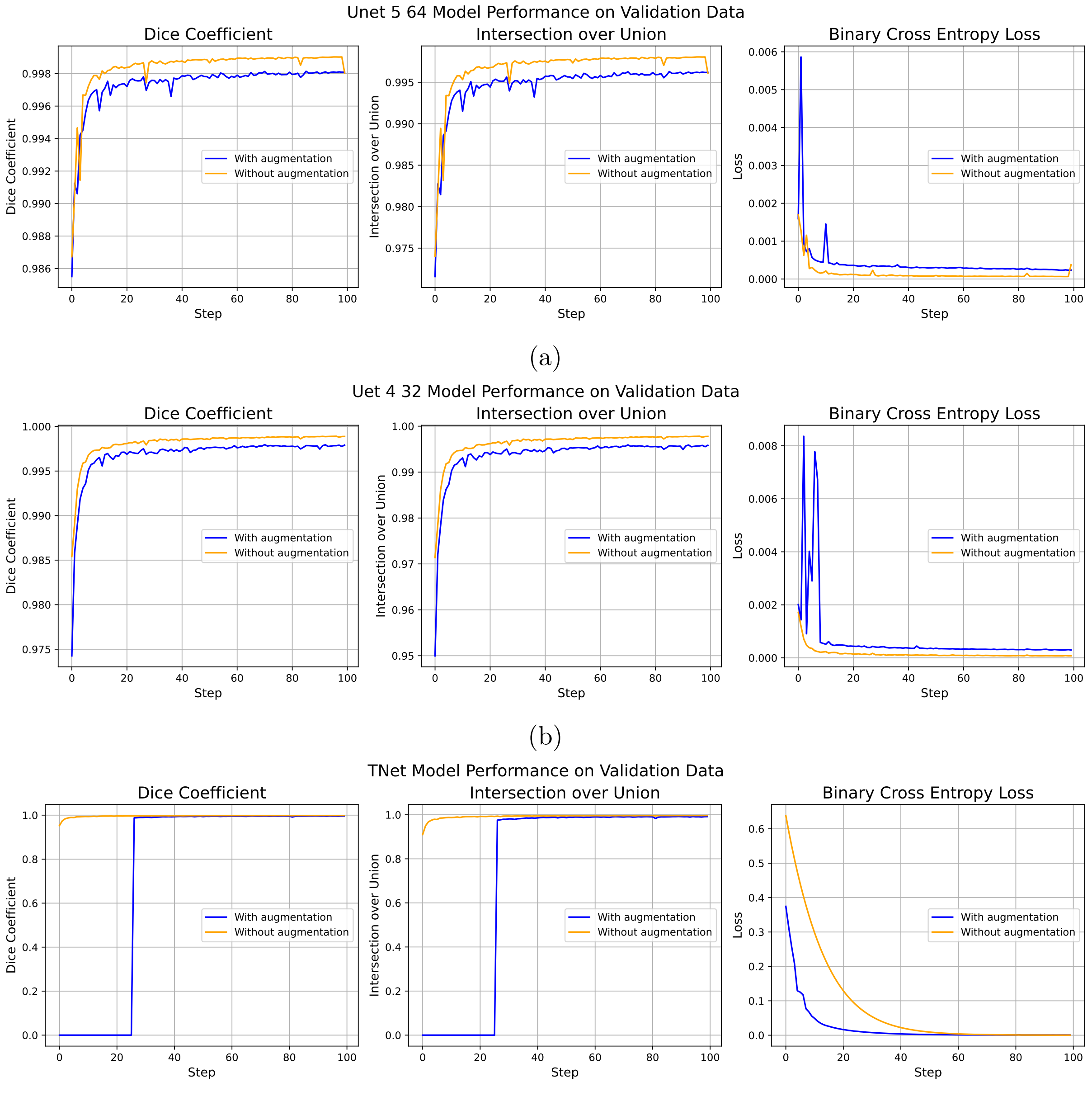
Figure 3.4. The original depth image of a scene with five objects, along with the different independent data augmentations used. The Noisy camera position augmentation adds small amounts of noise to the camera's transformation with respect to the real world, simulating real-world imprecision. The Random rotation, Random flip, and Random zoom augmentations are standard techniques used to improve model generalization and facilitate sim-to-real transfer.

Quantitative results Table 3.1 provides a comparison of different models trained with and without data augmentation and evaluated on the test set using IoU and Dice metrics. The results indicate that models with a larger number of trainable parameters exhibit better performance. However, it is important to note that all models show similar performance, with only negligible differences observed between the UNet 5 64 and the UNet 4 32, despite the significant difference in the number of parameters (100 times less for the UNet 4 32). Similarly, the TNet achieves comparable IoU and dice values with a much smaller number of parameters.

	Augmented	IoU ↑	Dice ↑	# Params.
UNet 5 64	✗	**0.9980**	**0.9989**	138, 340, 417
UNet 4 32	✗	0.9977	0.9988	8, 632, 865
TNet	✗	0.9962	0.9981	81, 694
UNet 5 64	✓	**0.9959**	**0.9979**	138, 340, 417
UNet 4 32	✓	0.9955	0.9977	8, 632, 865
TNet	✓	0.9913	0.9957	81, 694

Table 3.1. Comparison of different models with decreasing numbers of parameters using the Intersection over Union (IoU) and dice metrics. The evaluation is performed on a test set comprising 50, 000 novel simulated scenes and objects. The models under consideration are the UNet 5 64 and UNet 4 32, both of which follow the UNet architecture (see Figure 2.5). The UNet 5 64 model consists of five upsampling and downsampling layers, starting with 64 hidden dimensions. On the other hand, the UNet 5 32 model features four upsampling and downsampling layers, starting with 32 hidden dimensions. Additionally, we include the TNet model [41], which is a lightweight version of the UNet. The term "Augmented" indicates that the training was conducted using data augmentation.

Overall, all models achieve high IoU and Dice values of approximately 0.99, with slight differences in the thousandths place. It is worth mentioning that the discrepancy between models trained with augmented and non-augmented data can be attributed to the absence of data augmentation during testing. Models trained with data augmentation may have better generalization capability for real-world scenarios, as observed in sim-to-real experiments. However, they might slightly underperform on perfectly simulated data. This suggests that the models trained without data augmentation may be more specialized in handling ideal depth images from simulators. These high IoU and Dice values can be



(a)

(b)

(c)

Figure 3.5. Evolution of Dice coefficient, Intersection over Union (IoU), and loss values for the validation set during training. Both UNet models exhibit a rapid convergence to a plateau, while the TNet model demonstrates a distinct learning pattern. Notably, at around 22 epochs, the TNet model shows a significant improvement in performance. This improvement is observed specifically when using augmented data, indicating that the model does not have enough capacity to effectively learn object segmentation in the presence of noise, permutations, and other variations.

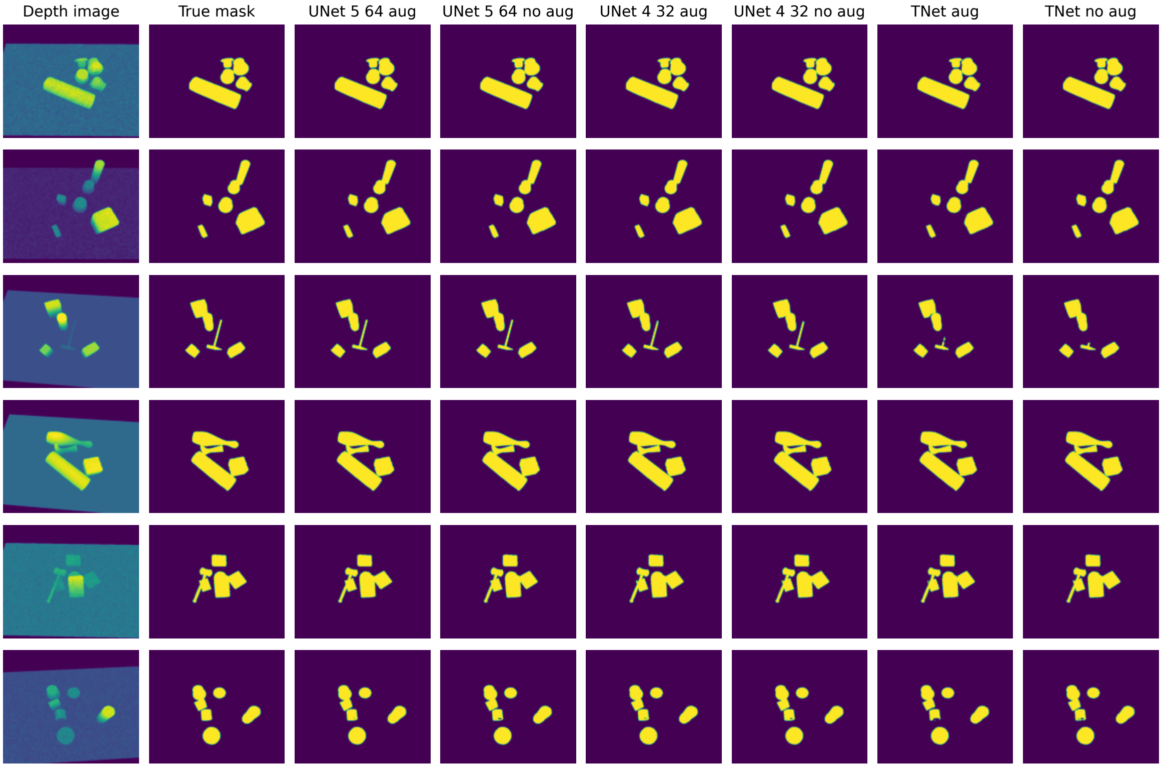


Figure 3.6. Comparison of different segmentation models trained with and without data augmentation (aug) on synthetic depth images.

explained by the fact that the task is relatively simple since it only consists of separating objects from a flat table while accounting for noise in the input data. Furthermore, by analyzing the training progress of these models (Figure 3.5), we can observed that all models reach a plateau relatively quickly during training, indicating that further training may not significantly improve their performance. This observation is supported by the train/val loss plots of the different models (Figure D.1).

Based on these results, the TNet shows the best trade-off between memory requirements and performance. However, it is important to note that as demonstrated later, this model does not possess sufficient capacity to generalize well to real depth images. Therefore, a model with a higher capacity, is required.

Qualitative results The segmentation results presented in Figure 3.6 show remarkable accuracy in various complex scenes of synthetic data for each of the models. However, on closer inspection, we can see that the TNet model has difficulty obtaining fine detail, as shown in the third and sixth figures. Although this discrepancy is noticeable, it is not considered a severe issue. For their part, the two UNet achieve outstanding results, with only a minor error observed in the sixth image. Overall, the segmentation performance of the UNet models is near perfect, demonstrating robustness and accuracy in capturing object boundaries and structures. The use or non-use of data augmentation during training does not result in significant differences in performance on these simulated data.

Per-image normalization may overlook smaller objects in scenes where both small and large objects are present. However, this is not a significant issue as segmentation is performed when an object needs to be picked up, and the workspace is cleared incrementally.

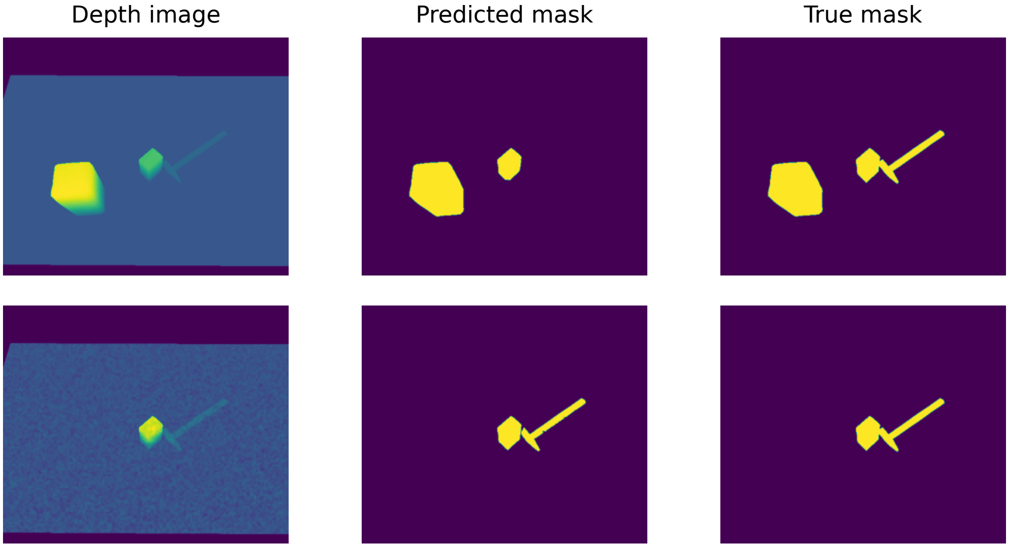


Figure 3.7. In this scene, there are three objects with varying heights. The UNet fails to segment the hammer due to the significant difference in height with the highest object (top). However, upon removing the tall object (bottom), the hammer is successfully detected. This outcome is attributed to the per-image normalization of depth images, enhancing the visibility of smaller objects when taller objects are absent, while reducing their visibility when taller objects are present.

Additionally, it is often advantageous to prioritize picking larger objects first. Furthermore, the per-image normalization ensures that smaller objects become more noticeable on the table after the larger objects are removed. Figure 3.7 provides an example illustrating this phenomenon.

Simulation to real To enhance the quality of the depth image captured by the depth camera, a median filter with a kernel size of 9 is applied. This filtering technique helps to smooth out the image and fill in any holes that may have occurred due to occlusion, reflections, or other factors. An example illustrating the effect of the median filter can be seen in Figure 3.8. Figure 3.9 provides a comparison of the segmentation results for the three models, considering whether they were trained with or without augmented data. The slight variations in depth observed on the table surface in the real depth images are due to

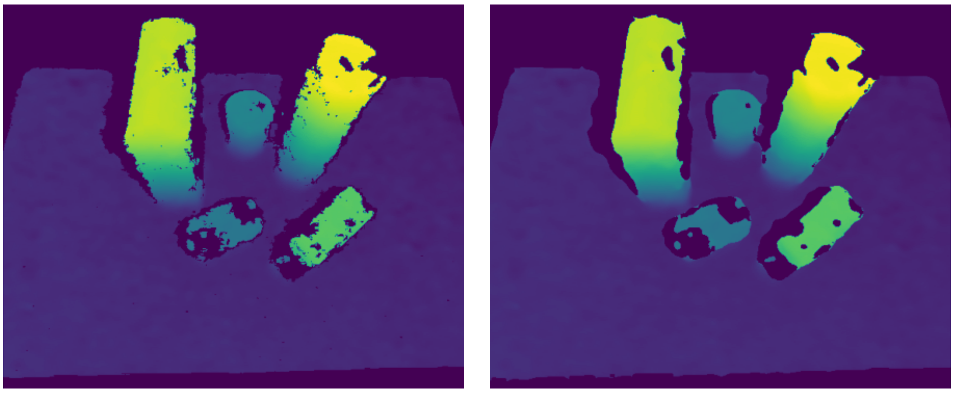


Figure 3.8. (left) The original raw depth image. (right) The same depth image after applying a median filter with a kernel size of 9 for smoothing.

a precision error in the exact position of the camera in relation to the table. Models that have not been trained with augmented data have difficulty interpreting these variations and misidentify them as objects. This is particularly evident in the second, fifth, and final scenes, where some models incorrectly detect the lower left corner of the table as objects due to its higher depth values. The TNet model, whether augmented or not, shows a lack of generalization in this scenario. In contrast, UNet models trained with augmented data show better performance in object segmentation, even for complex scenes such as the fifth and last frames. Notably, the UNet 4 32 trained with augmentation performs exceptionally well, consistently achieving near perfect object segmentation without the artifacts observed using the larger UNet. It is worth mentioning that despite being trained on scenarios involving piles and packed objects, the models perform better when applied to objects generated using the packed scenario. This is due to the more pronounced difference in depth between the table and the objects. Furthermore, although the models were not trained with "corrupted" depth images containing unknown points, they still process these images accurately.

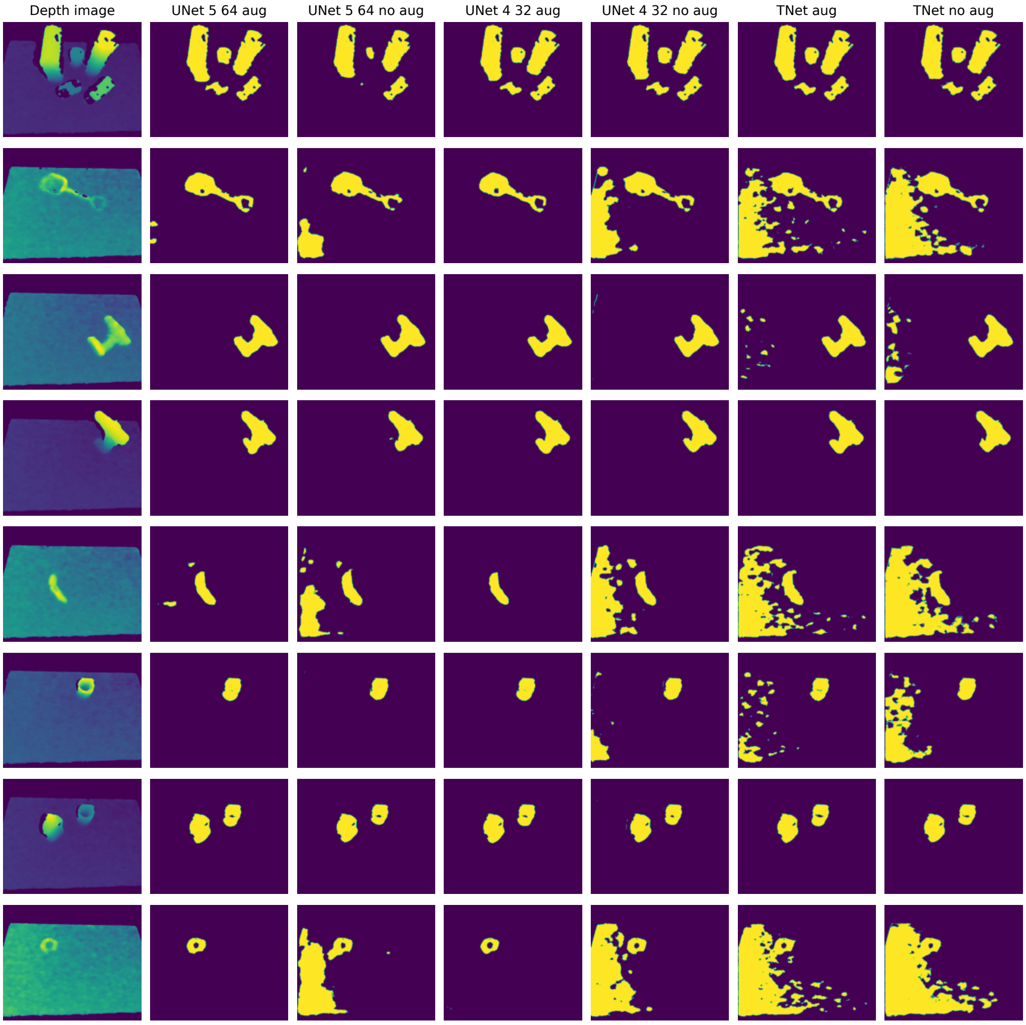


Figure 3.9. Comparison between various segmentation models trained with and without data augmentation (aug) using real depth images. It is evident that the utilization of augmented data assists the model in bridging the disparity between simulation and real scenarios.

3.3 Implicit representation

In this section, once the training procedure has been established, the models are evaluated quantitatively through a loss and metrics analysis, followed by a qualitative assessment. In addition, a comparison is made between results obtained using reconstructed and partial point clouds. Please note that the models presented in this section are the best-performing versions of each model, fine-tuned for optimal performance.

3.3.1 Evaluation protocol

Training The models are trained on a dataset of $2,048,000$ samples, then validated and evaluated on a separate set of $200,000$ samples as described in Section 3.1. Each sample consists of a point cloud with $2,048$ points, representing a scene of m objects, along with $2,048$ query points uniformly sampled within the workspace, accompanied by their respective occupancy values. The training process involved 100 iterations, with each iteration consisting of $32,000$ samples and a batch size of 32. We used the Adam optimizer with a learning rate of 0.0001, following the recommendations of the authors [14, 17, 35, 38], to minimize the BCE loss as previously defined.

Metrics The evaluation metrics used are the same as in the original paper introducing occupancy networks [14]. Since most of these metrics require a mesh representation, the MISE algorithm was used to extract the mesh by querying the occupancy network; the algorithm is described in Appendix C.

- The volumetric Intersection over Union (IoU) is computed by randomly sampling N ($100k$) points from the bounding volumes of the union of $\mathcal{M}_{GT}$ and $\mathcal{M}_{pred}$, which represent the sets of points inside the ground truth and predicted meshes, respectively. For each point, we determine if it is occupied in both the predicted mesh ($o^{pred} \in [0,1]$) and the ground truth mesh ($o^{GT} \in \{0,1\}$). Using a threshold of 0.5 for the predicted occupancy, the IoU is calculated as follows:

$$I\hat{o}U(\mathcal{M}_{pred}, \mathcal{M}_{GT}) = \frac{\sum_{i=1}^{N} o_i^{pred} * o_i^{GT}}{\sum_{i=1}^{N} o_i^{pred} + o_i^{GT}}$$

- The Chamfer-L_1 distance measures the distance between two meshes. It is defined as the average distance between each point of the predicted mesh and its nearest point on the ground truth mesh, and vice versa:

$$\text{Chamfer-}L_1(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}) = \frac{1}{2|\partial\mathcal{M}_{\text{pred}}|}\int_{\partial\mathcal{M}_{\text{pred}}} \min_{q\in\partial\mathcal{M}_{\text{GT}}} ||p-q||d_p$$
$$+\frac{1}{2|\partial\mathcal{M}_{\text{GT}}|}\int_{\partial\mathcal{M}_{GT}} \min_{q\in\partial\mathcal{M}_{\text{pred}}} ||p-q||d_q$$

  Where $\partial\mathcal{M}_{\text{pred}}$ and $\partial\mathcal{M}_{\text{GT}}$ denote the surface of the predicted and ground truth meshes respectively. We approximate this metric by sampling $100k$ points from the predicted and ground truth mesh surfaces.

- The normal consistency metric measures the alignment of surface normals between the predicted and ground truth meshes:

$$\text{Normal-Consistency}(\mathcal{M}_{\text{pred}}, \mathcal{M}_{\text{GT}}) = \frac{1}{2|\partial\mathcal{M}_{\text{pred}}|}\int_{\partial\mathcal{M}_{\text{pred}}} |\langle n(p), n(proj_2(p))\rangle| d_p$$

$$+\frac{1}{2|\partial\mathcal{M}_{\mathrm{GT}}|}\int_{\partial\mathcal{M}_{GT}}|\langle n(proj_1(q)), n(q)\rangle|d_q$$

Here, $n(p)$ and $n(q)$ are the surface normal vectors of $\partial\mathcal{M}_{\mathrm{pred}}$ and $\partial\mathcal{M}_{\mathrm{GT}}$, respectively. $\mathrm{proj}_1(q)$ and $\mathrm{proj}_2(p)$ denote the projection of q onto $\partial\mathcal{M}_{\mathrm{pred}}$ and p onto $\partial\mathcal{M}_{\mathrm{GT}}$.

To compute this metric, we find the nearest point in the ground truth mesh for each point in the predicted mesh. Then, we calculate the dot product between their surface normals and average the result over all points in the predicted mesh.

3.3.2 Quantitiative analysis

According to the loss in Figure 3.10, it appears that although the occupancy network could have been trained a little longer, the Conv-ONet achieved the lowest loss and reached the optimum very quickly, only after 18 iterations. Following closely behind are the Graph-ONet and the ONet, while the VN-ONet reached a plateau at a significantly higher loss value. This could be attributed to the Conv-ONet's inductive bias, which enables it to learn information more quickly. The Graph-ONet, which has full 3D equivariance, learned slightly more slowly than Conv-ONet, but still faster than ONet and VN-ONet. It is worth noting that both the Conv-ONet and VN-ONet have a tendency to overfit quickly. However, looking only at the loss can be misleading, and it is better to consider different metrics to evaluate the models' performance, as defined in Section 3.3.1. Table 3.2 shows that the Conv-ONet is the best performing model across all metrics, followed by the Graph-ONet. The VN-ONet and ONet yield lower results, with a considerable difference from the Conv-ONet. This disparity could be attributed to the fact that these models store global embeddings of the scene, making them less suitable for scenes containing multiple or complex objects.

	PointCloud	IoU ↑	Chamfer-$l1$ ↓	Normal consist. ↑
ONet	*partial*	0.8072 ± 0.1905	0.0015 ± 0.001	0.9418 ± 0.0622
Conv-ONet	*partial*	$\mathbf{0.8687 \pm 0.1115}$	$\mathbf{0.0010 \pm 0.0006}$	$\mathbf{0.9626 \pm 0.0333}$
VN-ONet	*partial*	0.7854 ± 0.2072	0.0018 ± 0.0013	0.8507 ± 0.1781
Graph-ONet	*partial*	0.8506 ± 0.1270	0.0013 ± 0.0006	0.9163 ± 0.0542
ONet	*reconstructed*	0.8227 ± 0.1890	0.0013 ± 0.0009	0.9475 ± 0.0572
Conv-ONet	*reconstructed*	$\mathbf{0.8667 \pm 0.1388}$	$\mathbf{0.0011 \pm 0.0007}$	$\mathbf{0.9592 \pm 0.0428}$
VN-ONet	*reconstructed*	0.7883 ± 0.2135	0.0017 ± 0.0014	0.9369 ± 0.0753
Graph-ONet	*reconstructed*	0.8603 ± 0.1335	0.0011 ± 0.0006	0.9574 ± 0.0427

Table 3.2. This table compares four models: Occupancy Network (ONet), Convolutional Occupancy Network (Conv-ONet), Vector Neurons Occupancy Network (VN-ONet), and Graph Occupancy Network (Graph-ONet). Evaluation metrics are used to assess these models on both partial and reconstructed point clouds. The results, based on a test set of 200,000 samples, show the average scores for each metric, along with their respective standard deviations. The findings indicate that Conv-ONet outperforms the other models, achieving a higher score on partial point clouds compared to reconstructed ones.

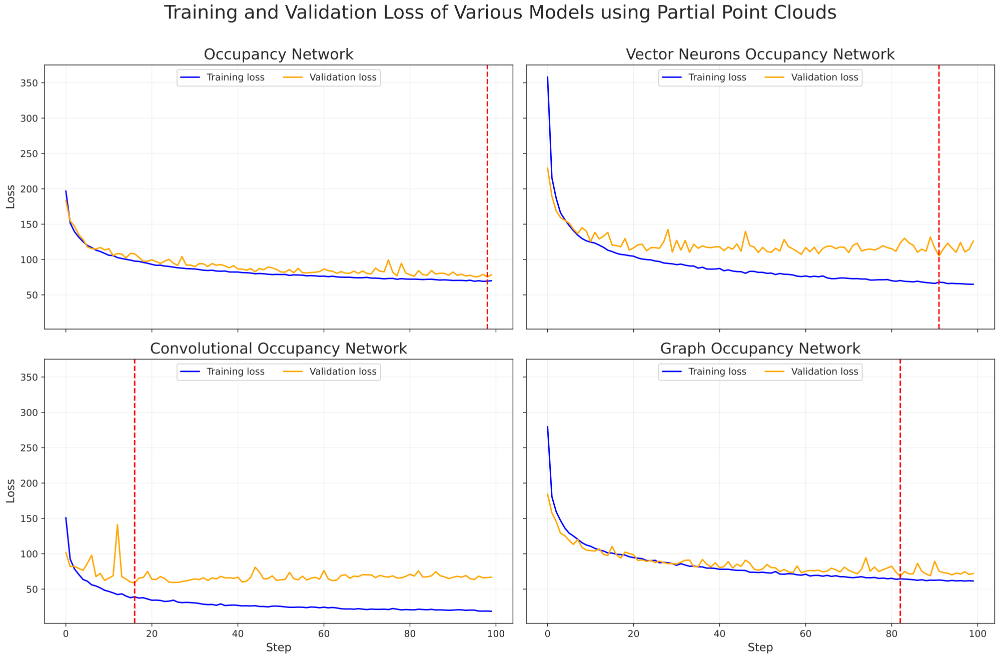


Figure 3.10. Training and validation loss over 100 iterations for each model using partial point clouds. Each iteration involves 32, 000 samples in batches of size 32, representing a total of 3.2 million different samples. The models are trained to minimize binary cross-entropy loss in order to determine whether or not a query point is inside an object. The red dotted line represents the lowest validation loss obtained by each model.

Impact of Reconstructed Point Clouds (overview)

An attempt was made to improve the performance of occupancy networks by providing them as input with reconstructed point clouds produced by PoinTr [43], a transformer-based model that reconstructs a complete point cloud from a partial one. However, this approach proved impractical due to severe GPU memory limitations, resulting in excessive overhead. Consequently, this method was quickly abandoned, which explains the brief mention of it. The model was initially pre-trained on the ShapeNet dataset [44] by the authors, followed by fine-tuning for our specific task using a dataset consisting of 100, 000 pairs of partial and complete point clouds, fo 50, 000 iterations. The authors' default hyperparameters were used for training. In the appendix, Figure D.2 presents qualitative results, illustrating the high accuracy of the reconstructed objects.

Looking at the losses of occupancy networks using reconstructed point clouds (Figure 3.11) shows that they do indeed enable us to learn a little faster, as the validation losses almost directly reach a plateau. The significant difference between training loss and validation loss may be due to the accumulation of errors in PoinTr and occupancy networks, both of which are trained on objects from the same pool, but which, during the validation phase, are confronted with unseen objects. Nevertheless, Graph-ONet achieved the lowest learning rate, which was expected since it performs best on complete point clouds. This is also reflected in the measurements (Table 3.2), as this model performed better using a reconstructed point cloud than a partial one. The other models showed similar behavior, with the exception of Conv-ONet, which obtained better results on partial point clouds, showing that this model may be limited by reconstruction errors.

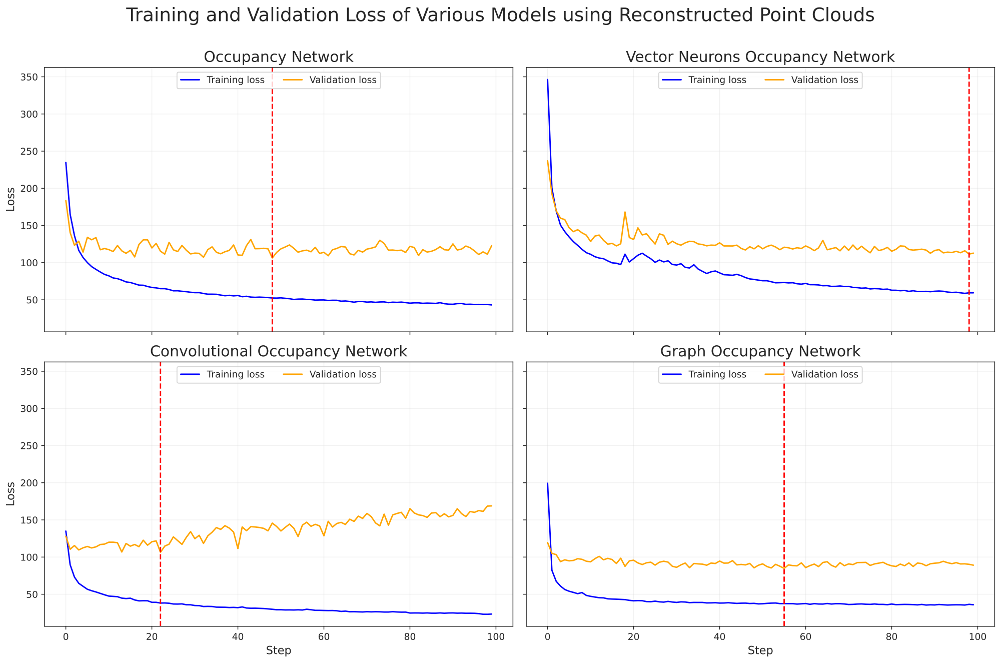


Figure 3.11. Training and validation loss over 100 iterations for each model using point clouds reconstructed by PoinTr [43]. Each iteration involves 32, 000 samples in batches of size 32, representing a total of 3.2 million different samples. The models are trained to minimize binary cross-entropy loss in order to determine whether or not a query point is inside an object. The red dotted line represents the lowest validation loss obtained by each model.

3.3.3 Qualitative analysis

In this analysis, we present a comparison of different occupancy network architectures with respect to their performance in handling objects of varying complexity levels, as illustrated in Figure 3.12. Upon analyzing the results, it is evident that the Conv-ONet outperforms the other architectures across all scenes.

As mentioned earlier, the VN-ONet and ONet do not perform well in scenes with multiple objects or complex structures with thin details. However, they perform reasonably well in representing simple objects, as can be observed with the second object, where the vector neurons architecture yields the best results. In contrast, Conv-ONet and Graph-Onet are highly effective in handling scenes with various levels of complexity, owing to their usage of both local and global embeddings, as well as powerful inductive bias. Although Graph-Onet yields slightly less smooth results compared to Conv-ONet, it is important to note that the former is not designed to handle partial point clouds. As Graph-ONet creates a graph using the points in the point clouds, it may not be as meaningful for incomplete point clouds. This could be a factor contributing to the slightly less smooth results obtained with Graph-Onet.

Rotation equivariance of vector neurons To ensure that the rotation equivariance properties of vector neurons hold, an experiment is conducted using a VN-Pointnet (Figure 2.11) applied to a point cloud to extract an embedding that represent the scene, and rotate it by $R \in SO(3)$. The resulting rotated embedding is compared with the embedding of the same point cloud rotated by the same R matrix. The aim is to verify whether the

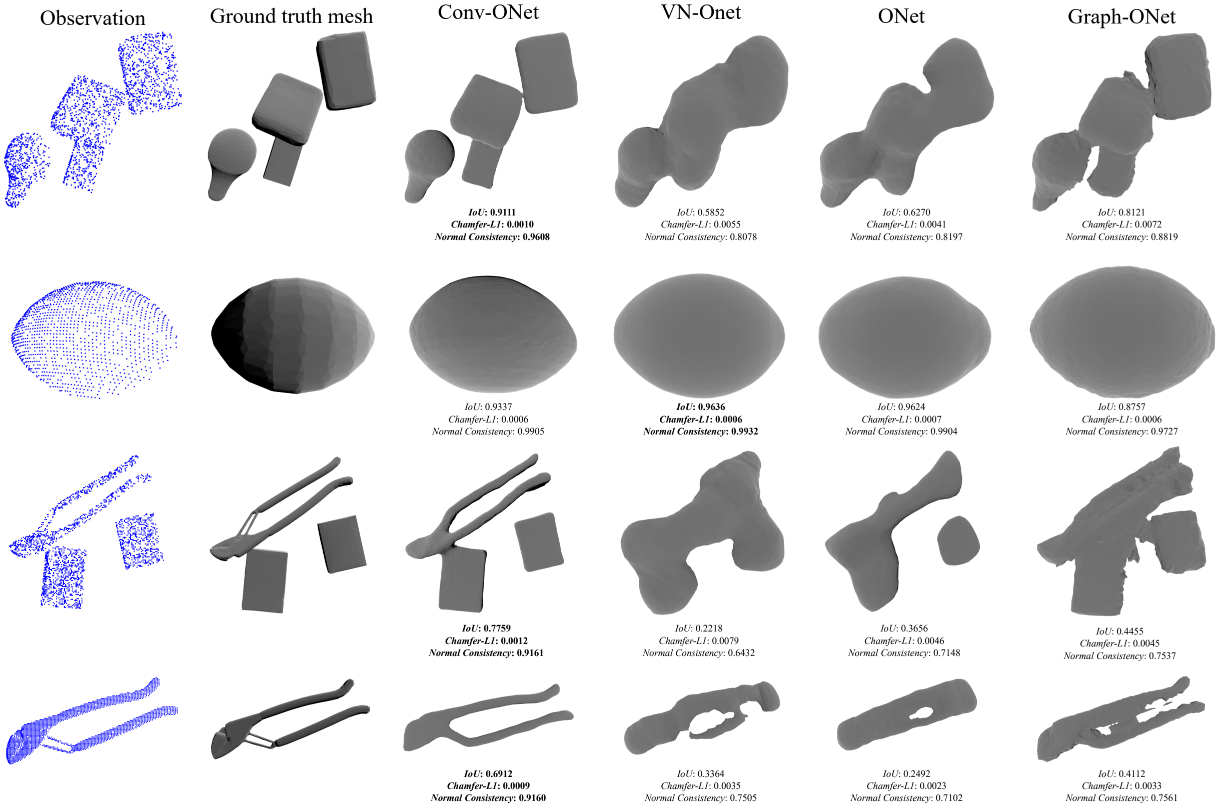


Figure 3.12. Comparison between four neural implicit representation models - Convolutional Occupancy Network (Conv-ONet) [17], Vector Neurons Occupancy Network (VN-ONet) [35], Basic Occupancy Network (ONet) [14], and Graph Occupancy Network (Graph-Onet) [37] - in their ability to reconstruct a mesh from a partial point cloud using the marching cube algorithm, as detailed in Section C. The Conv-ONet is found to be the best performing model, both quantitatively and qualitatively, when compared to the ground truth mesh.

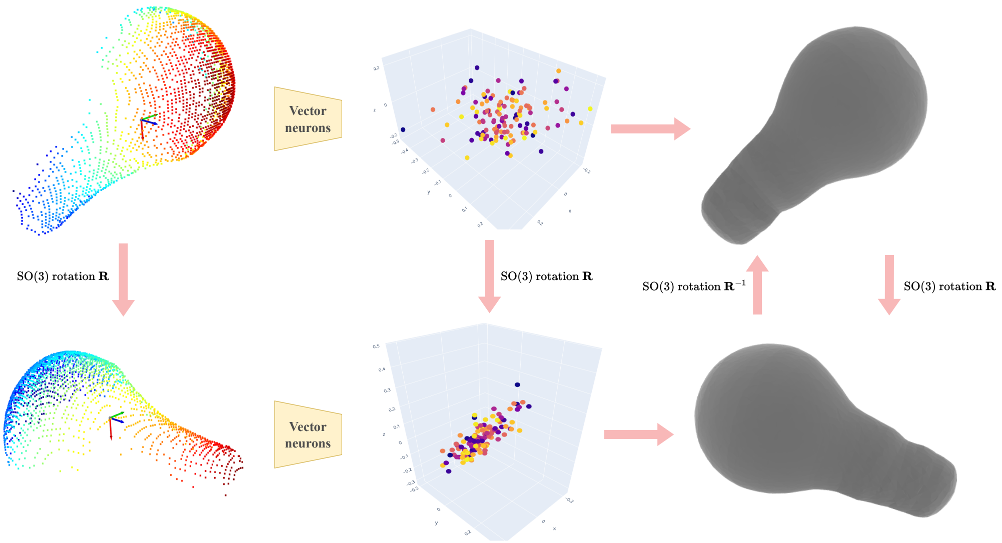

Vector neurons
SO(3) rotation $\mathbf{R}$
SO(3) rotation $\mathbf{R}$
SO(3) rotation $\mathbf{R}^{-1}$
SO(3) rotation $\mathbf{R}$
Vector neurons

Figure 3.13. This figure illustrates the rotation equivariance property of vector neurons. A partial point cloud of an object and the same object after being rotated by $R \in SO(3)$ are both processed by a VN-Pointnet (Figure 2.11). The resulting feature spaces are compared, revealing that the feature space of the rotated object is the same to the feature space obtained by rotating the original features with the same matrix R.

rotation equivariance property $f(R(x)) = R(f(x))$ applies, where f is the encoder made up of rotation equivariant layers as defined in Section 2.2.2. Visually, it seems that the rotation equivariance property has been respected, as the latent dimensions appear to be the same (Figure 3.13). However, for the sake of completeness, the norm one was computed between the two latent dimensions, resulting in a negligible value of $8 * 10^{-7}$ compared to the range of values for the latent dimensions (-0.5 to 0.5). To further validate the rotation equivariance, the meshes from the two latent spaces are compared. It is observed that the mesh has indeed been rotated, and applying the inverse rotation retrieved the original mesh, confirming the rotation equivariance property, at least empirically.

Additional analysis of convolutional occupancy network One of the advantages of the Conv-ONet is the ease with which the extracted feature space can be inspected (Figure 3.14). This enables us to diagnose any problems and ensure that the model is well-trained. Additionally, we can interpret some of the results based on the feature space analysis.

In addition to the features analysis, examining the logarithm of the predicted occupancy is crucial for this project. As explained in Section 6.2, the log of the prior probability $p(o = 1 \mid \mathbf{x}, \mathbf{P})$ obtained from the Conv-ONet is employed alongside other networks to compute the optimal pose through gradient descent. It is therefore essential to have smooth vector fields (contours) to facilitate convergence. However, as shown in Figure 3.15, we observe that while the density near the objects is high, it progressively becomes increasingly noisy as we move further away from the objects. This noise, although having low values, can introduce local minima. Additionally, due to the utilization of canonical planes in the method, some density still remains above the objects, as illustrated in the figure.

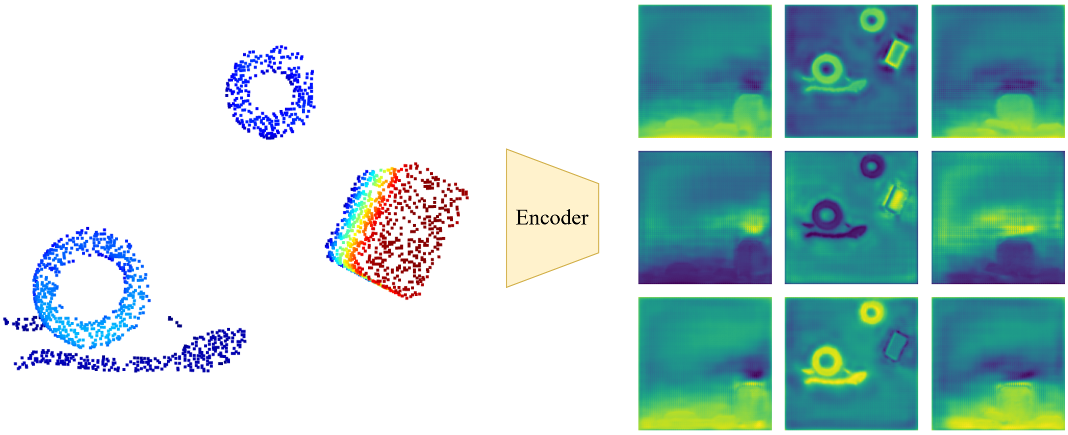


Figure 3.14. This figure depicts three out of the thirty-two sets of feature planes extracted by the feature extractor of the Conv-ONet applied to a point cloud of a scene. The feature planes correspond to the three canonical coordinate systems: xz, xy, and yz.

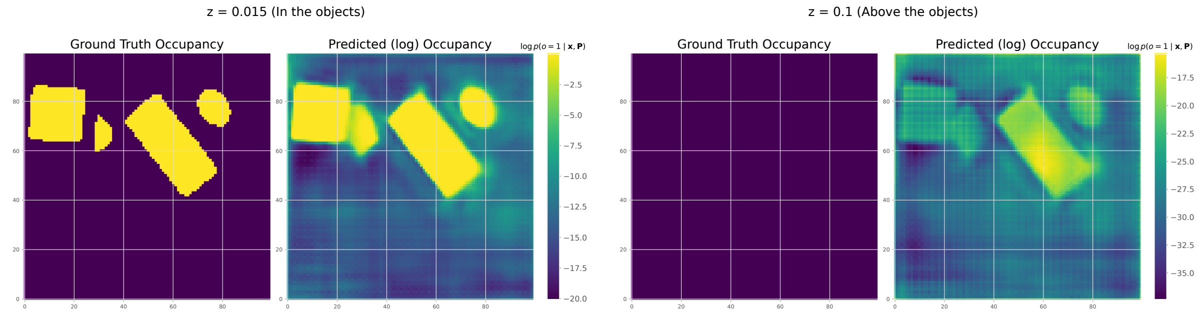


Figure 3.15. Predicted log occupancy slices extracted from a scene using a convolutional occupancy network, displayed along the z axis. The image on the left shows the slice at object level, while the image on the right shows the slice above the objects. Note that the scales of the two images are different.

Simulation to real Although the Conv-ONet shows good performance on point clouds generated from real depth images as shown in Figure 3.16, it still has some limitations. For example, in some cases, the model struggles to reconstruct objects accurately, as can be seen with the sprayer cut in two parts. This issue may be due to insufficient examples of such cases in the training data. One possible solution to this problem is to add randomly placed unknown points (represented by dark purple colors) to the synthetic depth images during training to account for real-world scenarios where certain points are not observable. However, considering that the training was performed using simulated data, the results are still quite satisfactory.

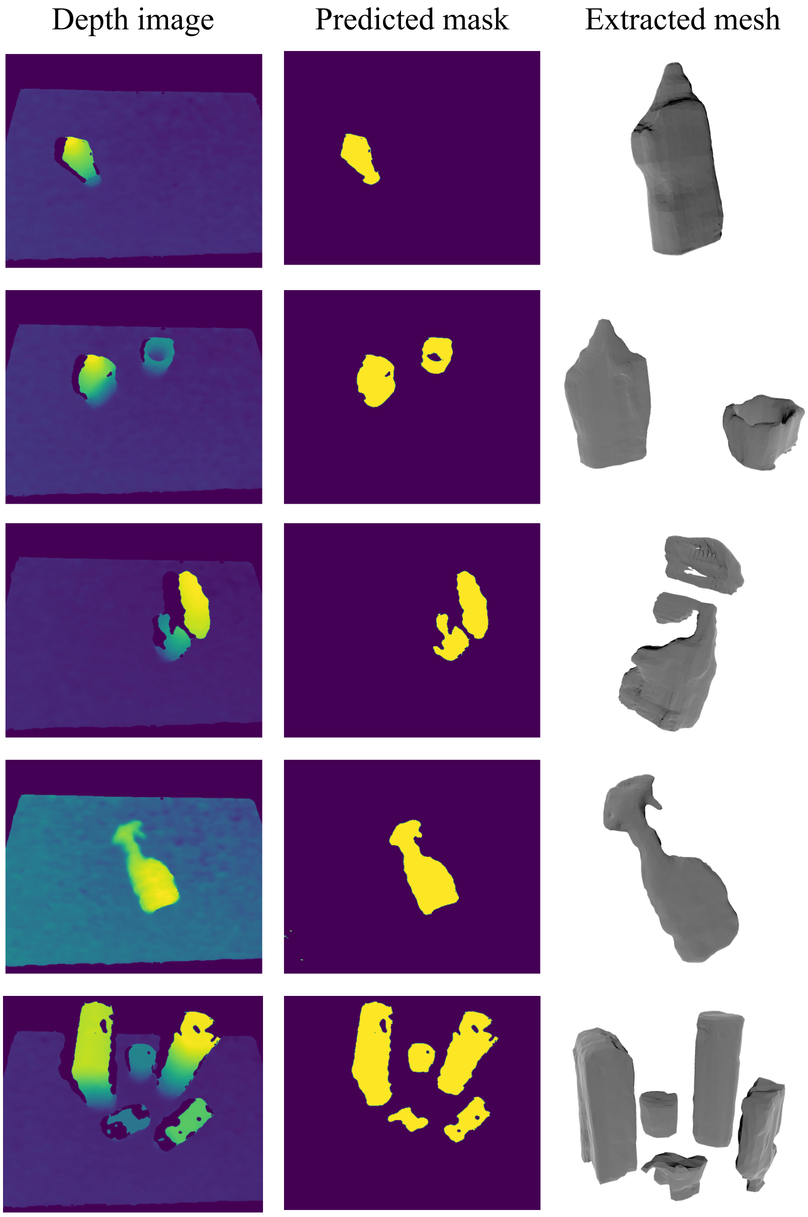


Figure 3.16. Qualitative results for real data. We applied our Conv-ONet to objects available at the lab. Despite only trained on synthetic data, our model generalizes reasonably well to real data.

Chapter 4

Discussion

Segmentation The segmentation plays a crucial role in the project as it forms the foundation for subsequent processes by extracting the point cloud to be fed into the occupancy network. Thus, a high-performing model is required, which is not only accurate but also fast and memory-efficient, as it will be used in conjunction with two other deep learning models in the final pipeline. Using a heavy model could significantly slow down the overall process and affect the performance of other models sharing the same GPU. During evaluation on simulated data, all models demonstrated comparable and satisfactory performance. In such cases, prioritizing the faster and lighter model would be beneficial. However, it is important to note that the transition from simulation to real-world data presents a significant challenge, their the TNet lacks the necessary capacity to generalize effectively. Consequently, the choice narrows down to the two UNet models. On real data both of them yield similar performance. However, the UNet 4 32 model shows better performance when trained with data augmentation. Additionally, being the lighter of the two models, it offers advantages in terms of memory savings and faster inference. Therefore, the UNet 4 32 model trained with data augmentation will be selected for the next part of the project.

Occupancy networks In this project, having a accurate representation of the object is essential since the main goal is to provide prior knowledge for the robot's object grasping tasks, and having a poor representation will limit the grasping possibilities. However, it is worth noting that the likelihood-to-evidence ratio (Chapter 5) updates this knowledge and can mitigate any errors that may occur. Furthermore, considering the size of the gripper, capturing fine details may not be critical, thereby reducing the impact of imperfect object representations. Based on the quantitative and qualitative results, the convolutional occupancy network (Conv-ONet) appears to be the most suitable model for our purpose. It offers efficiency, speed, and demonstrates strong generalization capabilities in both novel scenes and real-world scenarios. The outputs of the Conv-ONet provide accurate object representations, serving as a reliable prior for our model. For simplicity, we will refer to it as the "occupancy network" in Part II. In contrast, the simple occupancy network and vector neurons models lack the ability to efficiently handle complex scenes, despite the valuable rotation invariant property possessed by the vector neurons. The graph occupancy network, while an option, produces less smooth results and consumes excessive memory, making it an unfavorable choice.

Part II

Robotic grasping

Chapter 5

Grasping as inference

Building upon the concept of implicit representation discussed in Part I, this chapter focuses on exploring the different components of the posterior and their estimation, with a particular emphasis on how this approximated posterior can be used to extract an optimal grasping pose. As a reminder, the posterior probability, denoted as $p(\mathbf{h} \mid S, o = 1, \mathbf{P})$, represents the updated prior belief given observations. It can be expressed as:

$$p(\mathbf{h} \mid S, o = 1, \mathbf{P}) = \frac{p(S \mid \mathbf{h}, o = 1, \mathbf{P})}{p(S \mid o = 1, \mathbf{P})} p(\mathbf{h} \mid o = 1, \mathbf{P}). \tag{5.1}$$

5.1 Notations

The following notations are taken from [3] with slight modifications.

Frames: We use several reference frames in our work. The world frame $\underrightarrow{\mathcal{F}}_{\text{W}}$ and the workspace frame $\underrightarrow{\mathcal{F}}_{\text{S}}$ can be chosen arbitrarily and are not tied to a physical location. The world frame is used for the robot and the sensor, while the workspace frame is employed for our inference system. $\underrightarrow{\mathcal{F}}_{\text{C}}$ and $\underrightarrow{\mathcal{F}}_{\text{E}}$ correspond to the camera and the tool center point, respectively.

Hand Configuration: The hand configuration $\mathbf{h} \in \mathbb{R}^3 \times \mathbb{S}^1$ is defined as the pose $(\mathbf{x}, \mathbf{q}) \in \mathbb{R}^3 \times \mathbb{S}^1$ of the hand. In this context, $\mathbf{x}$ represents the vector $\vec{SE}$ and $\mathbf{q}$ represents the planar rotation using complex numbers $(\cos\beta, \sin\beta)$. Here, β corresponds to the rotation of the gripper. Both $\mathbf{x}$ and $\mathbf{q}$ are expressed in the $\underrightarrow{\mathcal{F}}_{\text{S}}$ coordinate system.

Success: A binary variable $S \in \{0, 1\}$ indicates whether the grasp fails ($S = 0$) or succeeds ($S = 1$), i.e., whether an object is successfully grasped and held above the table for at least 3 seconds.

Observation: Given the depth image I of a scene, along with its corresponding camera-to-world transformation $\mathbf{T}_{\text{WC}}$ and camera intrinsic matrix K, we construct a point cloud $\mathbf{P} \in \mathbb{R}^{2048\times3}$ expressed in $\underrightarrow{\mathcal{F}}_{\text{S}}$.

Occupancy: A binary variable $o \in \{0, 1\}$ indicates whether a point $\mathbf{p} \in \mathbb{R}^3$ is occupied by an object in the scene.

Latent Variables: The unobserved variables $\mathbf{z} \in \mathcal{Z}$ capture uncertainties about the nonsmooth dynamics of contact, the sensor noise, as well as the number of objects and their geometry.

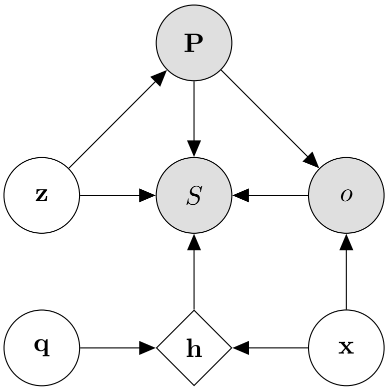


Figure 5.1. Probabilistic graphical model of the environment. Gray nodes correspond to observed variables and white nodes to unobserved variables. Unobserved variables **z** capture uncertainties about the dynamics of the system, the sensor noise, as well as the geometry of the object. On the other hand, the variable **h** is deterministic given **x** and **q**.

5.2 Probabilistic modeling

The posterior given by Equation 5.1 can be further decomposed. Specifically, considering that **q** is independent of the point cloud and occupancy, we have:

$$p(\mathbf{h} \mid o = 1, \mathbf{P}) = p(\mathbf{x} \mid o = 1, \mathbf{P})p(\mathbf{q}).$$

As discussed in Part I, by applying the Bayes rule to the prior over the position, we obtain $p(\mathbf{x} \mid o = 1, \mathbf{P}) = \frac{p(o=1|\mathbf{x},\mathbf{P})}{p(o=1|\mathbf{P})}p(\mathbf{x})$. This decomposition leads to the following expression:

$$p(\mathbf{h} \mid S, o = 1, \mathbf{P}) = \frac{p(S \mid \mathbf{h}, o = 1, \mathbf{P})}{p(S \mid o = 1, \mathbf{P})} \frac{p(o = 1 \mid \mathbf{x}, \mathbf{P})}{p(o = 1 \mid \mathbf{P})} p(\mathbf{x})p(\mathbf{q}).$$

Based on this complete posterior, we can construct the Bayesian network illustrated in Figure 5.1. The variables $S, \mathbf{P}, o, \mathbf{x}$, and $\mathbf{q}$ are modeled as random variables to account for noise and incorporate prior beliefs. The network structure is motivated by the dependencies among the variables, including S depending on $\mathbf{h}, o, \mathbf{P}$, z and o depending on $\mathbf{P}$ and $\mathbf{x}$, and $\mathbf{P}$ depending on z.

Grasping problem Given our probabilistic graphical model (Figure 5.1), we approach the grasping problem as a Bayesian inference task to determine the hand configuration $\mathbf{h}^*$. This configuration represents the most probable a posteriori, given a successful grasp within an occupied region of the scene. In other words, we are seeking the maximum a posteriori (MAP) estimate of the hand configuration:

$$\mathbf{h}^* = \arg\max_{\mathbf{h}} \; p(\mathbf{h} \mid S = 1, o = 1, \mathbf{P}).$$

The posterior can be expressed as the product of the likelihood-to-evidence ratio r and a scene-dependent prior:

$$p(\mathbf{h} \mid S, o = 1, \mathbf{P}) = r(S \mid \mathbf{h}, o = 1, \mathbf{P})p(\mathbf{h} \mid o = 1, \mathbf{P}),$$

With $r(S \mid \mathbf{h}, o = 1, \mathbf{P}) = \frac{p(S|\mathbf{h},o=1,\mathbf{P})}{p(S|o=1,\mathbf{P})}$. For brevity, let's assume that the Boolean variables S and o are set to 1 by default from now on.

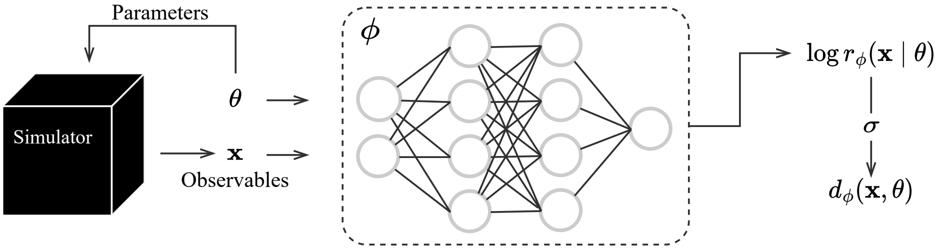

Parameters
Simulator
θ
$\mathbf{x}$
Observables
ϕ
$\log r_\phi(\mathbf{x} \mid \theta)$
σ
$d_\phi(\mathbf{x}, \theta)$

Figure 5.2. Graphical summary of neural ratio estimation for posterior approximation.

Both the likelihood function $p(S \mid \mathbf{h}, o, \mathbf{P})$ and the evidence $p(S \mid o, \mathbf{P})$ are intractable. This intractability comes from the fact that the likelihood is incomplete. Specifically, it can be expressed as $p(S \mid \mathbf{h}, o, \mathbf{P}) = \int_{\mathcal{Z}} p(S, z \mid \mathbf{h}, o, \mathbf{P}) dz$ where the integral involves summing over the latent space $\mathcal{Z}$ that has large cardinality. Similarly, the evidence integrating over all possible hand positions can be computationally expensive and also requires the evaluation of the intractable likelihood function: $p(S \mid o, \mathbf{P}) = \int_{\mathcal{H}} p(S \mid \mathbf{h}, o, \mathbf{P}) p(\mathbf{h} \mid o, \mathbf{P}) d\mathbf{h}$. However, it is still feasible to generate samples from physical simulators, enabling the use of Bayesian likelihood-free inference algorithms [45].

5.3 Neural ratio estimation

To approximate the ratio $r(S \mid \mathbf{h}, o, \mathbf{P})$, we can use an amortized neural ratio estimation (NRE) approach [45, 46]. This involves using a specific neural network r_ϕ designed for ratio estimation. Let's consider a simple scenario where our objective is to estimate the posterior distribution of parameters $p(\theta \mid x_{obs})$, which explains the observed data x_{obs} of a complex phenomenon. For this purpose, we have a simulator that generates data x based on parameters θ, formally denoted as $x \sim p(x \mid \theta)$. By applying Bayes' rule, we can express the posterior $p(\theta \mid x)$ as follows:

$$p(\theta \mid x) = \frac{p(x \mid \theta)}{p(x)} p(\theta).$$

However, evaluating the likelihood function $p(x \mid \theta)$ and the evidence $p(x)$ may be computationally or temporally impractical, or there might be no closed-form expression available. Introducing the likelihood-to-evidence ratio, $r(x \mid \theta) = \frac{p(x|\theta)}{p(x)}$, enables us to rewrite posterior in the form of

$$p(\theta \mid x) = r(x \mid \theta) p(\theta).$$

The likelihood-to-evidence ratio cannot be directly evaluated, but it can be approximated using the likelihood ratio trick [47]. This trick involves training a classifier $d_\phi(x, \theta)$ to distinguish between positive tuples (labeled $y = 1$) sampled from the joint distribution $(x, \theta) \sim p(x, \theta)$ and negative tuples (labeled $y = 0$) sampled from the product of marginals $(x, \theta) \sim p(x)p(\theta)$. The Bayes optimal classifier that minimizes the binary cross-entropy loss can be formulated as

$$d^*(x, \theta) = \frac{p(x, \theta)}{p(x, \theta) + p(x)p(\theta)}.$$

Applying the logit function to d^* allows us to recover the log-likelihood-to-evidence ratio:

$$\operatorname{logit}(d^*(x, \theta)) = \log \frac{d^*(x, \theta)}{1 - d^*(x, \theta)} = \log \frac{p(x, \theta)}{p(x)p(\theta)} = \log \frac{p(x \mid \theta)}{p(x)} = \log r(x \mid \theta).$$

In practice, we approximate d^* using a neural network that outputs $\log r_\phi(x \mid \theta)$. We then apply the sigmoid function to obtain an approximate classifier d_ϕ, as illustrated in Figure 5.2. This classifier estimates the probability of the positive class as:

$$\sigma(\log r_\phi(x \mid \theta)) = \frac{1}{1 + \exp(-\log r_\phi(x \mid \theta))} = d_\phi(x, \theta).$$

Finally, by training d_ϕ, we obtain a more accurate estimate r_ϕ of r. This allows us to approximate the posterior distribution as:

$$p(\theta \mid x) \approx \hat{p}(\theta \mid x) = r_\phi(x \mid \theta)p(\theta).$$

NRE for grasping Back to our grasping problem we can frame it as a NRE by sampling from two distributions, labelled as $y = 0$ and $y = 1$, respectively:

- $S, \mathbf{h} \sim p(S, \mathbf{h} \mid o = 1, \mathbf{P}) = p(S \mid \mathbf{h}, o = 1, \mathbf{P})p(\mathbf{h} \mid o = 1, \mathbf{P})$,
- $S, \mathbf{h} \sim p(S \mid o = 1, \mathbf{P})p(\mathbf{h} \mid o = 1, \mathbf{P})$.

For which the optimal classifier is:

$$d^*(S, \mathbf{h}, o, \mathbf{P}) = \frac{p(S, \mathbf{h} \mid o = 1, \mathbf{P})}{p(S, \mathbf{h} \mid o, \mathbf{P}) + p(S \mid o, \mathbf{P})p(\mathbf{h} \mid o, \mathbf{P})}.$$

Applying the logit function to the optimal classifer allow to retrieve the log-ratio:

$$\begin{aligned}
\operatorname{logit}(d^*(S, \mathbf{h}, o, \mathbf{P})) &= \log \frac{d^*(S, \mathbf{h}, o, \mathbf{P})}{1 - d^*(S, \mathbf{h}, o, \mathbf{P})} \\
&= \log \frac{\frac{p(S,\mathbf{h}|o,\mathbf{P})}{p(S,\mathbf{h}|o,\mathbf{P})+p(S|o,\mathbf{P})p(\mathbf{h}|o,\mathbf{P})}}{1 - \frac{p(S,\mathbf{h}|o,\mathbf{P})}{p(S,\mathbf{h}|o,\mathbf{P})+p(S|o,\mathbf{P})p(\mathbf{h}|o,\mathbf{P})}} \\
&= \log \frac{\frac{p(S,\mathbf{h}|o,\mathbf{P})}{\cancel{p(S,\mathbf{h}|o,\mathbf{P})+p(S|o,\mathbf{P})p(\mathbf{h}|o,\mathbf{P})}}}{\frac{p(S,\mathbf{h}|o,\mathbf{P})+p(S|o,\mathbf{P})p(\mathbf{h}|o,\mathbf{P})-p(S,\mathbf{h}|o,\mathbf{P})}{\cancel{p(S,\mathbf{h}|o,\mathbf{P})+p(S|o,\mathbf{P})p(\mathbf{h}|o,\mathbf{P})}}} \\
&= \log \frac{p(S, \mathbf{h} \mid o, \mathbf{P})}{\cancel{p(S, \mathbf{h} \mid o, \mathbf{P})} + p(S \mid o, \mathbf{P})p(\mathbf{h} \mid o, \mathbf{P})\cancel{-p(S, \mathbf{h} \mid o, \mathbf{P})}} \\
&= \log \frac{p(S, \mathbf{h} \mid o, \mathbf{P})}{p(S \mid o, \mathbf{P})p(\mathbf{h} \mid o, \mathbf{P})} \\
&= \log \frac{p(S \mid \mathbf{h}, o, \mathbf{P})\cancel{p(\mathbf{h} \mid o, \mathbf{P})}}{p(S \mid o, \mathbf{P})\cancel{p(\mathbf{h} \mid o, \mathbf{P})}} \\
&= \log \frac{p(S \mid \mathbf{h}, o, \mathbf{P})}{p(S \mid o, \mathbf{P})} \\
&= \log r(S \mid \mathbf{h}, o, \mathbf{P}).
\end{aligned}$$

Then, as explained earlier by training a classifier to approximate d^* we can retrieve an estimation of the ratio, named r_ϕ.

5.4 Scene-dependent Prior

The last component of the model is the prior $p(\mathbf{h} \mid o, \mathbf{P})$, which can be decomposed into two parts: the position $p(\mathbf{x} \mid o, \mathbf{P})$ and the orientation $p(\mathbf{q})$, represented as $p(\mathbf{h} \mid o, \mathbf{P}) = p(\mathbf{x} \mid o, \mathbf{P})p(\mathbf{q})$. This prior is scene-dependent as it depends on a point cloud $\mathbf{P}$ extracted from a depth image of the current scene.

Treating grasping as a learning problem necessitates a well-balanced and exhaustive dataset for training robots to effectively grasp objects, covering as much as possible all the grasping poses. By approaching the problem as described in Section 5.3, we can take advantage of an informed prior to sample promising grasping poses, enabling the network to learn more efficiently.

5.4.1 Orientation

From Norman et al. paper [3] the prior of the orientation is defined as "$\mathbf{q}$ is a uniform distribution over the unit circle $\mathbb{S}^1$. This prior is *invariant* to any rotation $\mathbf{R} \in \mathrm{SO}(2)$ applied to $\mathbf{q}$, satisfying $p(\mathbf{q}) = p(\mathbf{Rq})$. This property enables free selection of the reference frame on the table. Additionally, the prior can be extended to SO(3) by using quaternions on $\mathbb{S}^3$."

5.4.2 Positions

The prior distribution over the position, denoted as $p(\mathbf{x} \mid o = 1, \mathbf{P})$, can be decomposed as follows:

$$p(\mathbf{x} \mid o = 1, \mathbf{P}) = \frac{p(o = 1 \mid \mathbf{x}, \mathbf{P})}{p(o = 1 \mid \mathbf{P})} p(\mathbf{x}) \propto p(o = 1 \mid \mathbf{x}, \mathbf{P})p(\mathbf{x}),$$

where $p(o = 1 \mid \mathbf{x}, \mathbf{P})$ refers to the Convolutional Occupancy Network discussed in Section 2.2.2, and $p(\mathbf{x})$ is uniform over the workspace. Sampling from $p(\mathbf{x} \mid o = 1, \mathbf{P})$ is necessary for training the ratio r_ϕ as described in Section 5.3. However, an analytical form of this distribution is not available. To overcome this challenge, we use Markov chain Monte Carlo methods [48], specifically the Hamiltonian Monte Carlo (HMC) [49]. HMC requires a differentiable likelihood, which is satisfied by the differentiable properties of the neural network representing the likelihood in our case. Figure 5.3 illustrates an example of the posterior obtained by applying HMC to a specific scene.

Since objects generally occupy a small portion of the total workspace, the use of an uninformed prior such as a uniform distribution over the object bounding box, as in the previous model [2], does not scale well with an increasing number of objects in the scene. It would require an impractically large number of samples to achieve a sufficient number of successful grasps to allow r_ϕ to learn how to grasp objects. In contrast, our approach allows direct sampling of grasping positions within the object volume using only a depth image of the scene. However, our prior assumes that the grasping point lies inside the object volume, which limits pose coverage, particularly for convex shapes.

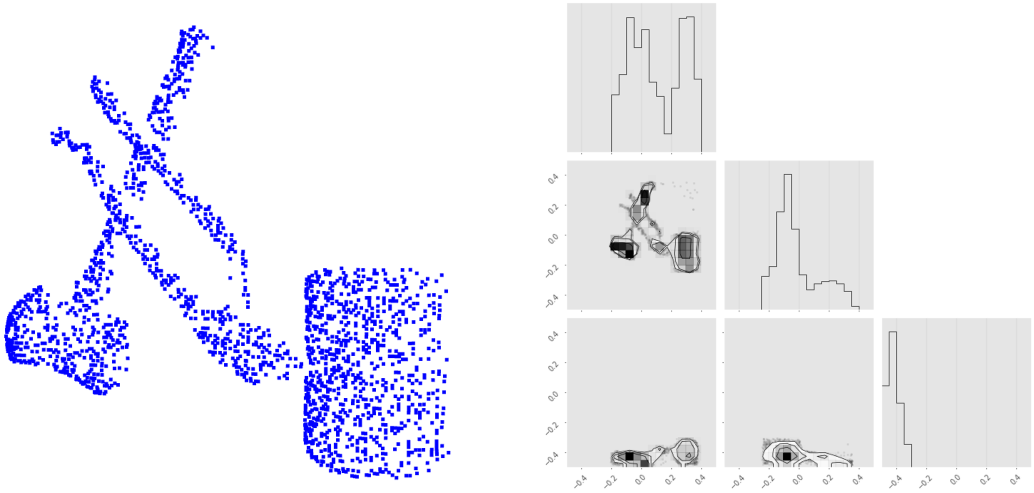


Figure 5.3. Corner plot representing the samples obtained from the distribution $p(\mathbf{x} \mid o = 1, \mathbf{P})$ using Hamiltonian Monte Carlo (HMC) for the given partial point cloud displayed on the left.

Chapter 6

Grasping pipeline

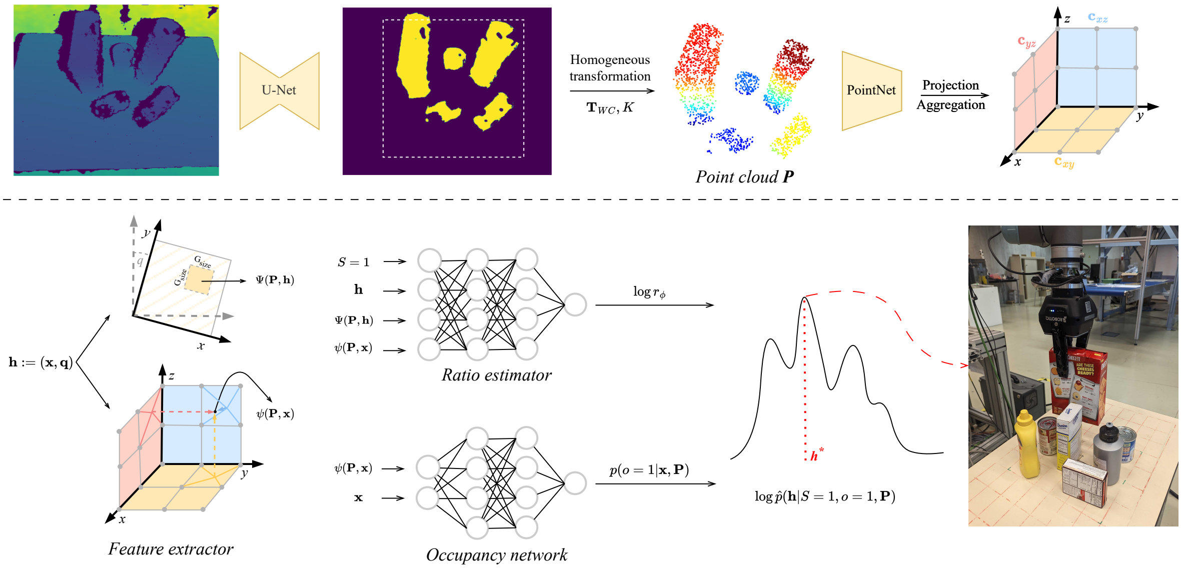

U-Net
Homogeneous transformation
$\mathbf{T}_{WC}, K$
Point cloud $\boldsymbol{P}$
PointNet
Projection
Aggregation
z
$\mathbf{c}_{xz}$
$\mathbf{c}_{yz}$
y
x
$\mathbf{c}_{xy}$
y
q
$\mathrm{G}_{\mathrm{size}}$
$\mathrm{G}_{\mathrm{size}}$
$\Psi(\mathbf{P}, \mathbf{h})$
x
$\mathbf{h} := (\mathbf{x}, \mathbf{q})$
z
$\psi(\mathbf{P}, \mathbf{x})$
y
x
Feature extractor
$S = 1$
$\mathbf{h}$
$\Psi(\mathbf{P}, \mathbf{h})$
$\psi(\mathbf{P}, \mathbf{x})$
Ratio estimator
$\log r_\phi$
$\psi(\mathbf{P}, \mathbf{x})$
$\mathbf{x}$
Occupancy network
$p(o = 1|\mathbf{x}, \mathbf{P})$
h^*
$\log \hat{p}(\mathbf{h}|S = 1, o = 1, \mathbf{P})$

Figure 6.1. Our grasp inference pipeline. It begins with a noisy depth image of the scene, from which we first separate the objects from the background using a U-Net [27]. We then generate three canonical feature planes following the approach in [17]. To evaluate a given $\mathbf{h}$, we extract point-wise $\psi(\mathbf{P}, \mathbf{x})$ and local $\Psi(\mathbf{P}, \mathbf{h})$ features and feed them to the ratio and occupancy networks. Using the resulting differentiable posterior and Riemannian optimization, we finally identify the most plausible hand configuration $\mathbf{h}^*$.

In this chapter, we present the pipeline components for inferring a grasping position from a depth image, taking advantage of a scene-dependent prior that incorporates object occupancy. Here is an overview of the pipeline:

1. **Input processing**: The background is removed from a depth image, and only object pixels are extracted using a segmentation model based on a U-Net architecture The segmented depth image is then converted into a point cloud $\mathbf{P}$. To ensure consistency, $\mathbf{P}$ is preprocessed by uniformly downsampling and upsampling through point repetition, resulting in 2048 points (refer to Section 2.2.1).

2. **Posterior Approximation**: The encoder part of the prior $p(\mathbf{x} \mid o = 1, \mathbf{P})$ also serves as a feature extractor, generating three canonical feature planes from $\mathbf{P}$. From

these planes, point-wise features $\psi(\mathbf{P},\mathbf{x})$ and local features $\Psi(\mathbf{P},\mathbf{h})$ are extracted. These features play a crucial role in evaluating both the occupancy and ratio networks, enabling us to approximate the posterior distribution $p(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$.

3. **Maximum A Posteriori (MAP)**: The final step involves computing the MAP estimate by maximizing the log posterior. Riemannian gradient ascent is used to preserve the non-linearity of $\mathbb{S}^1$ during this estimation process.

For a visual representation of this pipeline, refer to Figure 6.1.

6.1 Density ratio estimation

In order to improve the reliability and conservativeness of our posterior approximations, we follow the approach demonstrated by Hermans et al. [50] and use an ensemble of neural ratio estimators. We employ five individual models and compute their average to approximate the density ratio. Mathematically, this is expressed as:

$$\log \hat{r} = \log \left(\frac{1}{5} \sum_{i=1}^{5} \exp(\log \hat{r}_i) \right)$$

To convert the logarithmic ratios obtained from the models into regular ratios, we apply the exponential function. These five models are trained independently on the same dataset and share the same architecture.

6.1.1 Architectures

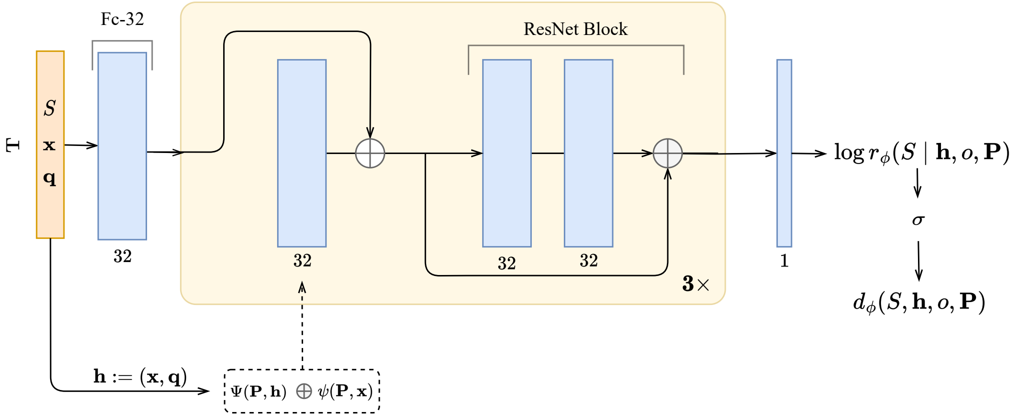

Fc-32
ResNet Block
T
S
$\mathbf{x}$
$\mathbf{q}$
32
32
32
32
1
$3\times$
$\log r_\phi(S \mid \mathbf{h}, o, \mathbf{P})$
σ
$d_\phi(S, \mathbf{h}, o, \mathbf{P})$
$\mathbf{h} := (\mathbf{x}, \mathbf{q})$
$\Psi(\mathbf{P},\mathbf{h}) \oplus \psi(\mathbf{P},\mathbf{x})$

Figure 6.2. The architecture of the ratio estimator r_ϕ involves taking the success S, hand pose $\mathbf{h}$, and feature representation of the point cloud $\mathbf{P}$ (consisting of point-wise ψ and local Ψ features) as inputs. r_ϕ then processes these inputs to produce an estimation of the logarithmic ratio r_ϕ.

In our work, we experimented with several ratio estimator architecture, denoted by $r_\phi(S \mid \mathbf{h}, o, \mathbf{P})$, which incorporate different levels of scene knowledge. The base architecture is similar to the one of the Conv-ONet decoder (Figure 6.2), but the extracted features are different. The ratio estimator takes three inputs: the success S (either 0 or 1), the hand pose $\mathbf{h}$, and the features representing the point cloud $\mathbf{P}$. The features of $\mathbf{P}$ are obtained using the encoder of the Conv-ONet, which uses the same weights as the prior

$p(o = 1 \mid \mathbf{x}, \mathbf{P})$ and outputs three canonical feature planes: $\mathbf{c}_{xy}(\mathbf{P}), \mathbf{c}_{xz}(\mathbf{P})$, and $\mathbf{c}_{yz}(\mathbf{P})$. These global features do not enable the ratio to reason about the hand's position and orientation or estimate the potential success of a grasp, as they represent the entire scene in different canonical planes. Instead, we extracted both point-wise $\psi(\mathbf{P}, \mathbf{x})$ and local $\Psi(\mathbf{P}, \mathbf{h})$ information, which provide additional details about the scene while avoiding unnecessary noise. It is worth noting that although the figures do not explicitly display it, most of the fully connected layers are followed by a ReLU activation to account for non-linearity. For further details, please refer to the source code.

Here is a list of models we considered, each with varying levels of information about the scene extracted from the features planes:

Point-wise features

As a baseline, we used the simplest model that extracts features only at the 3D point level (Figure 6.3). For a given query coordinate $\mathbf{x}$, we extracted the local features as

$$\psi(\mathbf{P}, \mathbf{x}) = \mathbf{c}_{xy}(\mathbf{P})(\mathbf{x}_{xy}) + \mathbf{c}_{xz}(\mathbf{P})(\mathbf{x}_{xz}) + \mathbf{c}_{yz}(\mathbf{P})(\mathbf{x}_{yz}),$$

where $\mathbf{c}{*}{*}(\mathbf{P})(\mathbf{x}_{**})$ represents the bilinear interpolation at the precise 2D coordinate of the feature plane as described in Section 2.2.2.

However, this model's simplicity prevent its ability to reason about the scene as it does not have local feature representation $\Psi(\mathbf{P}, \mathbf{h})$. With only a limited view of the scene, it is difficult to understand the implication of the orientation and position of the gripper for grasping an object. In addition, it cannot take into account the shape of the object or anticipate potential collisions with other objects, since it has no explicit information about them.

For the subsequent models, the point-wise feature ψ remains unchanged, while only the local features Ψ vary

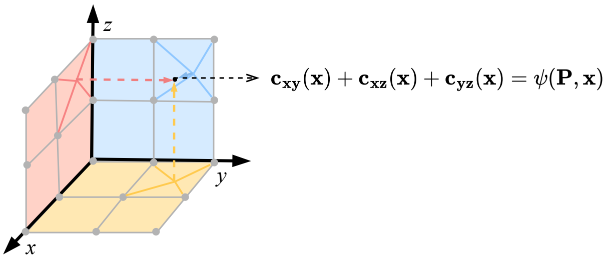

z
y
x
$\mathbf{c_{xy}}(\mathbf{x}) + \mathbf{c_{xz}}(\mathbf{x}) + \mathbf{c_{yz}}(\mathbf{x}) = \psi(\mathbf{P}, \mathbf{x})$

Figure 6.3. Point-wise features for a given point cloud. It is the sum of the bilinear interpolations from the canonical feature planes at a coordinate $\mathbf{x}$.

Hands features

To enhance the ratio's understanding of the scene, incorporating features from the fingertips and palm of the hand at its grasping pose could prevent collisions with other objects in the scene. However, this approach still does not enable reasoning about the scene as a whole. Specifically, we denote

$$\Psi(\mathbf{P}, \mathbf{x}) = [\psi(\mathbf{P}, \mathbf{x}_{f1}), \psi(\mathbf{P}, \mathbf{x}_{f2}), \psi(\mathbf{P}, \mathbf{x}_{p})],$$

where $\mathbf{x}_{f1}$, $\mathbf{x}_{f2}$, and $\mathbf{x}_p$ are the coordinates of the fingertips and palm when reaching the pose $\mathbf{h}$. Figure 6.4 illustrates the locations where the features are extracted, with green and blue representing the fingertips and red representing the palm. However, this approach is limited to extremely localized information and does not allow a more global understanding of the scene, indicating only whether the hand collides with an object or not, leaving room for potential failures.

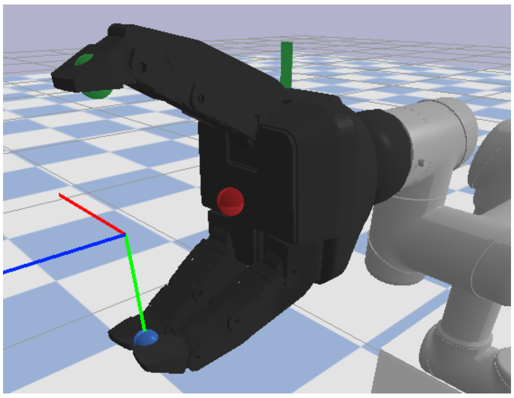


Figure 6.4. The two fingertips of the hand (blue, green) and the palm (red) of the gripper.

Graph features

This model, inspired by the Graph Occupancy Network [37], it uses the features extracted from the PointNet before the projection (See Figure 2.12). For a query point $\mathbf{x}$, the k-nearest neighbors' features are taken, along with the relative distance to $\mathbf{x}$ and concatenated, then passed through two fully connected layers as we can see in Figure 6.5 to produce the *local* features $\Psi(\mathbf{P}, \mathbf{x})$. However, due to the incompleteness of the point cloud, these features may lack sufficient information, as demonstrated in Section 3.3.3 regarding the implicit representation of a partial view. Moreover, this model is quite time and resource-consuming.

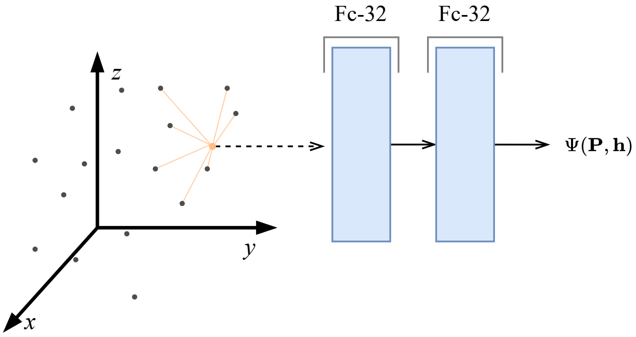


Figure 6.5. To obtain the local features $\Psi(\mathbf{P}, \mathbf{h})$, we use the features of the k nearest neighbor points, which are processed by a PointNet. These features are combined with the relative distance from the query point $\mathbf{x}$ and then fed through two fully connected layers. The size of the receptive field primarily relies on the value of k.

Cropped features

As the robotic arm operates in a 4 degrees of freedom (DoF) configuration, objects are grasped using a top-down approach. Therefore, in order to estimate the potential success of a grasp, the model needs to understand the effect of a specific hand position. Specifically, it requires an understanding of the object's shape and potential collisions between the hand and objects. To achieve this, the feature plane $\mathbf{c}xy$ is cropped to match the size of the gripper and centered on $\mathbf{x}xy$ (Figure 6.6). This cropping approach enables the model to gain a better understanding of the scene by providing features of the scene that are focused around the expected hand position. To obtain $\Psi(\mathbf{P}, \mathbf{h})$, the cropped feature plane undergoes two convolutional neural networks (CNNs) with 16 hidden layers and a kernel size of 3, followed by two fully connected layers with a hidden size of 32.

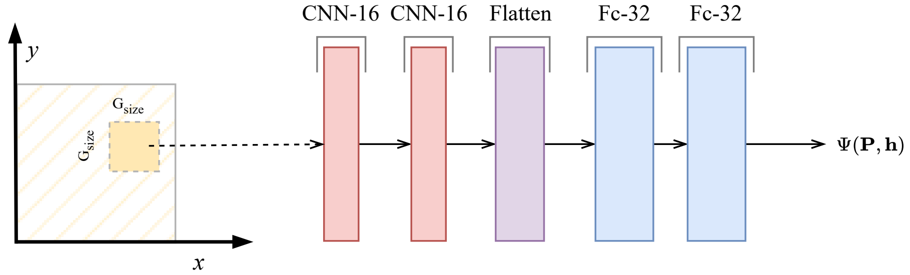


Figure 6.6. To extract the local features $\Psi(\mathbf{P}, \mathbf{h})$ at the hand level, the canonical planes $\mathbf{c}_{xy}$ are first cropped to the size of the gripper (G_{size}), which roughly represents 20% of the workspace surface, centered at $\mathbf{x}_{xy}$.

Cropped & Rotated features

This model shares a similar architecture with the previous one but introduces a new component to account for the rotation of the gripper. The rotation is incorporated by applying a rotation transformation to the cropped feature planes. The rotation angle, denoted as β, represents the gripper's rotation on the unit circle embedded as the complex number $\mathbf{q} = (\cos(\beta), \sin(\beta))$. The angle β is calculated using the formula $\beta = \text{atan2}(\cos(\beta), \cos(\beta))$. If any points fall outside the input boundaries, they are filled with 0. By explicitly incorporating rotation into the features, rather than requiring the model to learn how to rotate them based on $\mathbf{q}$, the model benefits from an inductive bias that facilitates faster learning and improves performance.

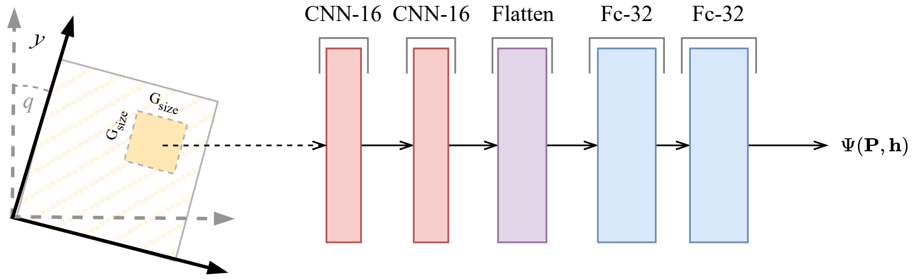
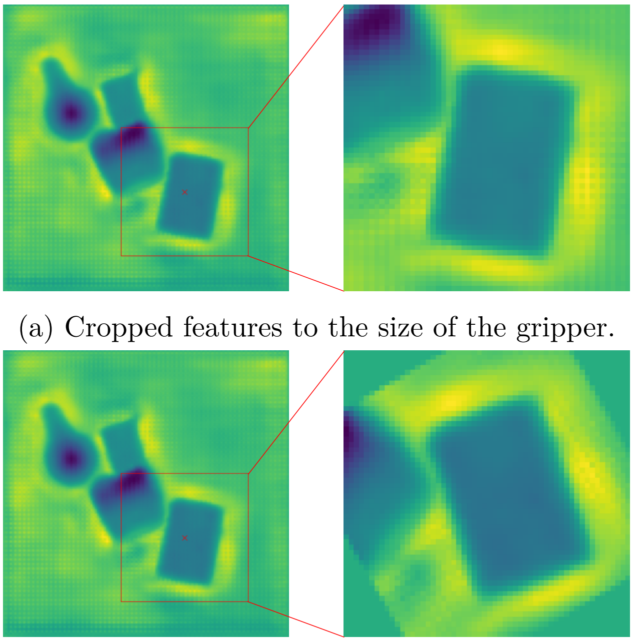


Figure 6.7. In order to extract local features Ψ at the hand level while considering the orientation, the canonical planes $\mathbf{c}_{xy}$ are first cropped to match the size of the gripper and centered at $\mathbf{x}_{xy}$. After the cropping process, these cropped planes are rotated according to the gripper's rotation $\mathbf{q}$ and subsequently passed through two CNN. The resulting feature maps are then flattened and further processed by two fully connected layers.



(a) Cropped features to the size of the gripper.

(b) Cropped & Rotated features, using a rotation β set to 30°.

Figure 6.8. Comparison between "Cropped features" and "Cropped & Rotated features" obtained from a specific scene's xy feature planes.

For a better understanding, Figure 6.8 show an xy feature plane extracted from a scene. The first plot (Figure 6.8a) illustrates the feature plane after it has been cropped based on the dimensions of the gripper. The second plot (Figure 6.8b) demonstrates the same feature plane after applying a rotation angle β to simulate the orientation of the gripper.

6.2 Maximum a posteriori estimation

Having defined the prior $p(\mathbf{h} \mid o, \mathbf{P})$ and estimated the likelihood-to-evidence ratio $r_\phi(S \mid \mathbf{h}, o, \mathbf{P})$, we can now compute the posterior distribution as the product of these two:

$$\hat{p}(\mathbf{h} \mid S, o, \mathbf{P}) = r_\phi(S \mid \mathbf{h}, o, \mathbf{P}) p(\mathbf{h} \mid o, \mathbf{P}).$$

To achieve a successful grasp, we aim to find the hand pose $\mathbf{h}$ that maximizes the posterior probability given a successful grasp, occupancy o, and point cloud $\mathbf{P}$. In other words, we seek the maximum a posteriori (MAP) estimate:

$$\mathbf{h}^* = \arg\max_{\mathbf{h}} \; \hat{p}(\mathbf{h} | S = 1, o = 1, \mathbf{P}). \tag{6.1}$$

To ensure numerical stability, we minimize the negative logarithm of the posterior probability:

$$\begin{aligned}
\mathbf{h}^* &= \arg\max_{\mathbf{h}} \; \log r_\phi(S = 1 \mid \mathbf{h}, o = 1, \mathbf{P}) + \log p(\mathbf{h} \mid o = 1, \mathbf{P}) \\
\mathbf{h}^* &= \arg\max_{\mathbf{h}} \; \log r_\phi(S = 1 \mid \mathbf{h}, o = 1, \mathbf{P}) + \log \frac{p(o = 1 \mid \mathbf{x}, \mathbf{P}) p(\mathbf{x})}{p(o = 1 \mid \mathbf{P})} + \log p(\mathbf{q}) \\
\mathbf{h}^* &= \arg\min_{\mathbf{h}} \; -\log r_\phi(S = 1 \mid \mathbf{h}, o = 1, \mathbf{P}) - \log p(o = 1 \mid \mathbf{x}, \mathbf{P}) - \log p(\mathbf{x}) - \log p(\mathbf{q})
\end{aligned}$$

Since an analytical form of this estimated posterior is not available, we rely on approximation methods to find the maximum. Gradient-based optimization methods are applicable since the model is composed of neural networks. Hence, we use Adam optimizer with a learning rate of 0.005 for $\mathbf{x}$ and 0.05 for $\mathbf{q}$. However, as shown in Figure 3.15, the log density of the prior is not smooth and contains many local minima, which will certainly complicate the optimization. Although the likelihood-to-evidence ratio may smooth the posterior to some extent, it is safer to use multiple starting points for optimization. Thus, we sample K position and orientation pair from their respective prior $\mathbf{x}, \mathbf{q} \sim p(\mathbf{x})p(\mathbf{q})$ and select the top-k points using two different strategies: the *sampling*-based and the *optimized*-based strategy.

- In the **Sampling-based strategy**, the posterior is evaluated for each of the K position and orientation pairs ($\mathbf{x}$, $\mathbf{q}$), and the top-k points with the highest values are selected as starting points.
- In the **Optimized-based strategy**, the positions of the K samples are optimized for 100 iterations to maximize $p(o \mid \mathbf{x}, \mathbf{P})$. The optimized points are then evaluated on the conditioned posterior, and the top-k position and orientation pairs ($\mathbf{x}$, $\mathbf{q}$) are selected. This strategy allows starting with points that are already on the object, potentially close to the optimal position. Optimizing the prior helps speed up the reasoning process compared to optimizing all K points on the posterior, which can be computationally intensive.

Finally, the selected k points are optimized on the posterior for 300 iterations, and the hand pose $\mathbf{h}$ that minimizes the negative log posterior is chosen as the final grasping point. In practice $K = 1000$ and $k = 10$.

When optimizing the hand pose, the orientation $\mathbf{q}$ is represented as a point on the unit circle $\mathbb{S}^1$. This representation poses a challenge as it is not defined in Euclidean space, and using traditional gradient descent methods would violate geometric assumptions. To address this issue, the approach of using Riemannian optimization, as proposed by Norman et al. [2, 3], is adopted. For more detailed information on the implementation of Riemannian optimization in this context, please refer to the original paper by Marlier et al. as Riemannian optimization is not within my contributions. For completeness Figure 6.9 schematically summarise this optimization process.

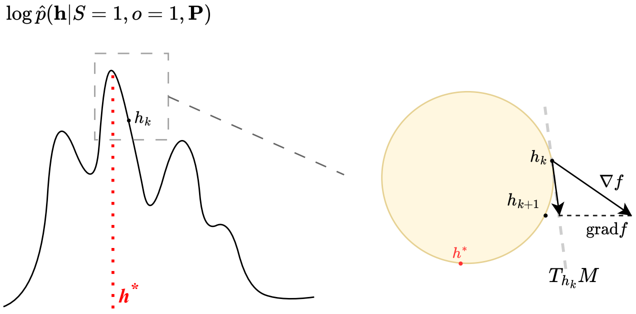


Figure 6.9. The figure illustrates Riemannian optimization, where $\mathbf{h}_k$ represents the previous optimization point and $\mathbf{h}_{k+1}$ denotes the next point. The Euclidean gradients $\nabla f(\mathbf{h}_k)$ are transformed into their Riemannian counterparts grad $f(\mathbf{h})$, which are computed and projected onto the tangent space $T_{h_k}M$. The update rule uses an exponential mapping given by $\mathbf{h}_{k+1} = \exp \mathbf{h}_k(-\alpha_k \mathrm{grad}\ f(\mathbf{h}_k))$, where $\exp_x(v) : \mathcal{T}_x\mathcal{M} \rightarrow \mathcal{M}$.

Chapter 7

Experiments

In this chapter, we carry out experiments using the different ratio architectures previously defined. These experiments are conducted both in simulated environments and in real-life scenarios, allowing us to compare our approach with state-of-the-art methods. In addition, the posterior and prior distributions are examined in depth to provide insight into our method of grasping and explaining how the model apprehends an object. Finally, we provide a complete ablation studies to validate our design choices. As a reminder the robot operates in a 4-degree-of-freedom (DoF) setting, which limits the grasping approach to a top-down orientation.

7.1 Experiment protocol

The prior $p(\mathbf{x} \mid o = 1, \mathbf{P})$ uses the same Conv-ONet, as described in Part I, with a resolution of 128 and a hidden layer feature dimension of 32.

Data generation process The structure defined in Figure 5.1 provides a direct and straightforward method for generating data by conditioning on $o = 1$. Formally the data generation process is:

$$\begin{aligned}
\mathbf{z} &\sim p(\mathbf{z}) \\
I &\sim p(I \mid \mathbf{T}_{WC}, \mathbf{z}) \\
\mathbf{P} &= f(I, \mathbf{T}_{WC}, K) \\
\{\mathbf{x} &\sim p(\mathbf{x} \mid o = 1, \mathbf{P})\} \\
\{\mathbf{q} &\sim p(\mathbf{q})\} \\
\{\mathbf{h} &= (\mathbf{x}, \mathbf{q})\} \\
\{\tau_{1:l} &\sim \Lambda(\tau_0, \mathbf{IK}(\mathbf{h}), \mathbf{P})\} \\
\{S &\sim p(S \mid \tau_{1:l}, \mathbf{z})\}
\end{aligned}$$

Grasping is successful when the robot holds an object above the table for a minimum of 3 seconds. This grasp is performed at pose $\mathbf{h}$ and is executed using trajectory $\tau_{1:l}$ obtained through a path planner Λ. The path planner takes into account $\mathbf{P}$ to avoid collisions and uses an inverse kinematics solver $\mathbf{IK}$ to determine the robot's motion required to reach $\mathbf{h}$. For a given $\mathbf{P}$ a set of pairs $\mathcal{D}(\mathbf{P}) = \{(\mathbf{h}^n, S^n) : n = 1 : N\}$ are sampled.

The data generation process takes place entirely in a simulated environment built with PyBullet. The latent variable $\mathbf{z}$ follows the description described in [2].

Training The samples $\mathcal{D}(\mathbf{P})$ are drawn from the distribution $p(S \mid \mathbf{h}, o = 1, \mathbf{P})p(\mathbf{h} \mid o = 1, \mathbf{P})$, which are considered positive samples based on Section 5.3. However, to train our model, we also require negative samples from $p(S \mid o = 1, \mathbf{P})p(\mathbf{h} \mid o = 1, \mathbf{P})$. To obtain these negative samples, we randomly shuffle the hand poses $\mathbf{h}^{1,\ldots,N}$ from $\mathcal{D}(\mathbf{P})$. This shuffling process ensures that the success labels and hand poses are independent given $\mathbf{P}$, enabling us to train our discriminator. We use the following loss function for training the discriminator:

$$\begin{aligned}\mathcal{L} =& \frac{1}{|B| \times 2N} \sum_{i=1}^{|B|} \Bigg(\sum_{(S,\mathbf{h}) \in \mathcal{D}(\mathbf{P}_i)} \mathcal{L}_{BCE}(d_\phi(\mathbf{P}_i, \mathbf{h}, S), 1) \\ &+ \sum_{(S,\mathbf{h}') \in \text{shuffle}(\mathcal{D}(\mathbf{P_i}))} \mathcal{L}_{BCE}(d_\phi(\mathbf{P}_i, \mathbf{h}', S), 0) \Bigg)\end{aligned} \tag{7.1}$$

The occupancy value o is not explicitly provided as an input during training, as the model is conditioned on samples drawn from $o = 1$, which treats it as a constant in this context. However, $\mathbf{P}$ varies according to the scene.

The classifier is trained for 150 epochs using $10,000$ different scenes composed of two sets of $4,000$ $(\mathbf{h}, S)$ pairs for a given $\mathbf{P}$. The optimizer used is Adam with default parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and the initial learning rate is 10^{-4}, which is halved every 50 epochs but remains bounded to 10^{-5}. Batch size vary according to the model used to accommodate memory constraints: 32 for the "point-wise" and "hands" features, 1 for the others. Each batch consists of a $\mathbf{P}$ and $\mathcal{D}(\mathbf{P})$. The objects used are the same as those defined in section 3.1, but using only the packed scenario.

Metrics To evaluate and compare our models, we use metrics inspired by the work of Breyer et al. [40] defined as follows:

- **Success rate (SR)**: The success rate measures the percentage of successful grasps. Since we are evaluating the grasping performance and not the path planner, we allow for up to 3 failed attempts from the path planner before considering the grasp as a failure.
- **Percent clear (% clear)**: This metric assesses the model's ability to clear objects from a table. For a scene with m objects, we evaluate how well the model can grasp and remove these objects. If the model successfully grasps k consecutive objects without failure but fails to grasp the next one, the % clear for that particular scene is calculated as m/k, where $m \leq k$. The % clear for all scenes are then aggregated to provide an estimate of the model's performance in clearing objects from the table.

7.2 Training analysis

Figure 7.1 shows the mean loss and its standard deviation for the different models described in Section 6.1.1. This plot indicates that the *Graph features*, *Cropped features*, and *Cropped & Rotated features* models converge rapidly to their optimum, indicating

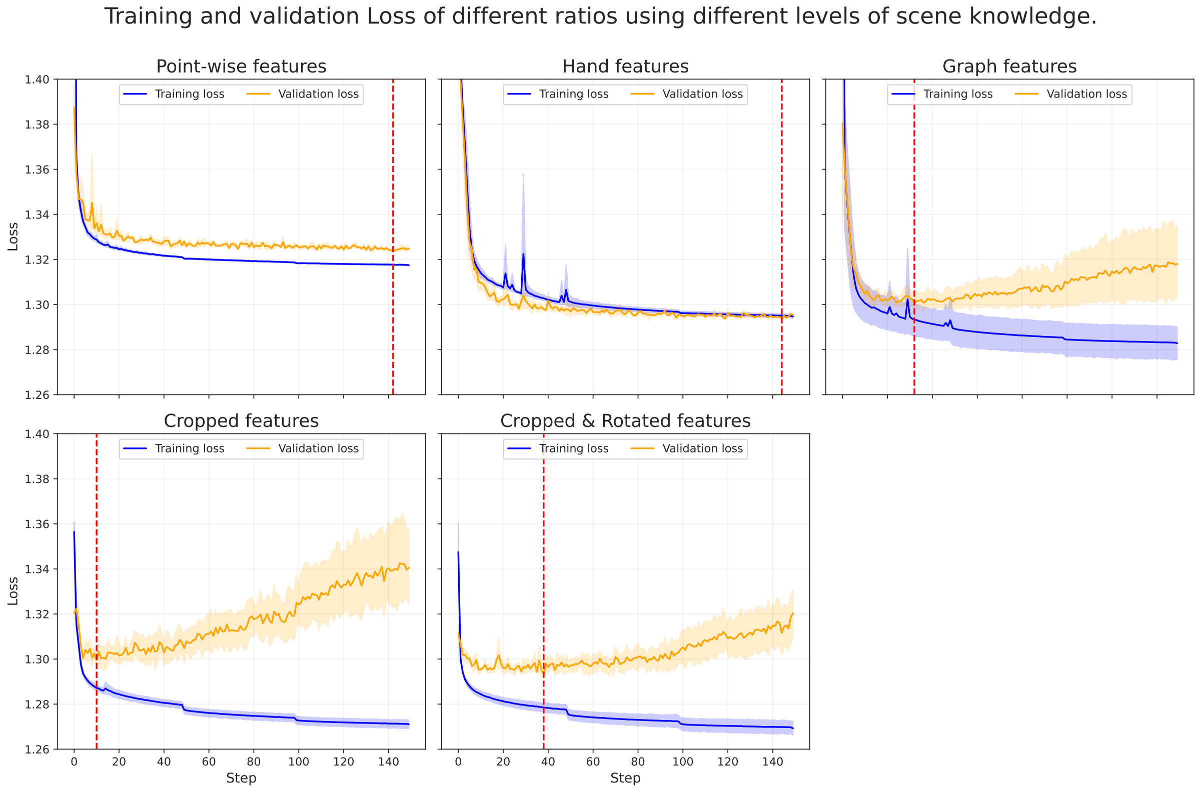


Figure 7.1. Training and validation loss of different models with varying levels of scene knowledge. Each model is trained five times, and the results presented here display the average loss along with the standard deviation. The dashed red line indicates the step for which the average validation loss is the lowest for each of the models. The training process started with a learning rate of 10^{-4} and is halved every 50 steps.

that providing more information about the scene enables faster learning. Curiously, the *Cropped features* model reaches its optimum faster than the *Cropped & Rotated features*, whereas the latter has more meaningful features since the feature planes have been rotated according to the gripper. Although the *Cropped & Rotated features* model achieves a lower loss and exhibits slightly less overfitting. But overall, it seems that the increase in inductive bias in the models leads to a greater tendency to overfitting and faster convergence. However, it is important to note that the loss alone may not be a reliable metric for model performance. For instance, the *Hand features* model achieves the lowest loss but performs relatively poorly compared to the *Cropped & Rotated features* model, as shown in Table 7.1. Finally, it is worth mentioning that the training of the *Graph features* model seems to be more unstable and has a higher variance than the others. This high variance can be explained by the fact that the point cloud is incomplete, which makes it more difficult to learn patterns on the data.

7.3 Models evaluation

7.3.1 Simulation results

The quantitative results obtained from 200 rounds of experiments are shown in Table 7.1. Each round involves a table with 5 objects randomly placed according to the packed

	Initial points strategy	SR ↑	% cleared ↑
Point-wise features	Optimized-based	71.26	43.15
Hands features	Optimized-based	79.23	55.06
Graph features	Optimized-based	85.58	63.82
Cropped features	Optimized-based	88.16	67.45
Cropped & Rotated features	Optimized-based	**91.37**	**75.25**
Point-wise features	Sampling-based	70.62	40.95
Hands features	Sampling-based	77.49	49.77
Graph features	Sampling-based	85.01	61.45
Cropped features	Sampling-based	89.12	68.03
Cropped & Rotated features	Sampling-based	**89.78**	**71.07**

Table 7.1. We compared different models based on their success rates (%) and percent cleared for picking experiments in the packed scenario. These experiments were conducted in a simulated environment, involving 5 objects over 200 rounds.

scenario (Section 3.1). The end of a round occurs when there are no more objects left on the table or in the case of an unsuccessful grasp.

Success rate Considering first the *Optimized-based strategy* for the choice of the initial points, the *Point-wise features* model show the poorest performance, which is understandable since it only provide information about the grasping point without considering the surrounding context. Consequently, the success rate (SR) is relatively low (71.26%). However, incorporating information on fingertip and palm positions resulted in an 8% improvement. This improvement may come from a reduction in gripper collision errors, as the model has more information about the hand in space. The *Graph features* significantly enhanced the performance, achieving a success rate of 85.58%. This indicates that having more comprehensive global information about the scene and object structure is beneficial. This observation is supported by the last two models, which provide a cropped and centered view of the current scene based on hand position and size. Notably, the introduction of rotation in feature space as a function of clamp orientation yields a success rate of 91.37%, confirming its positive impact on ratio performance. In terms of the choice between the *Optimized-based* strategy and the *Sampling-based* strategy, the performance difference is negligible. However, the *Optimized-based* strategy slightly improves the results..

Percent cleared The percent cleared achieved by each model is relatively low, but it does show improvement with a higher success rate. These low values suggest that the models struggle when multiple objects are present in the scene. When a failure occurs and there are still many objects remaining, the percent cleared is penalized more severely. This observation is supported by the data presented in Figure 7.2.

Failure rates at different stages of each round reveal interesting trends. The *Cropped features* and *Cropped & rotated features* models show a decreasing failure rate as the round progresses, suggesting limitations associated with the size of the gripper. On the other hand, the remaining models exhibit an increasing failure rate as the number of objects on the table decreases. This suggests that gripper size may not be the only factor

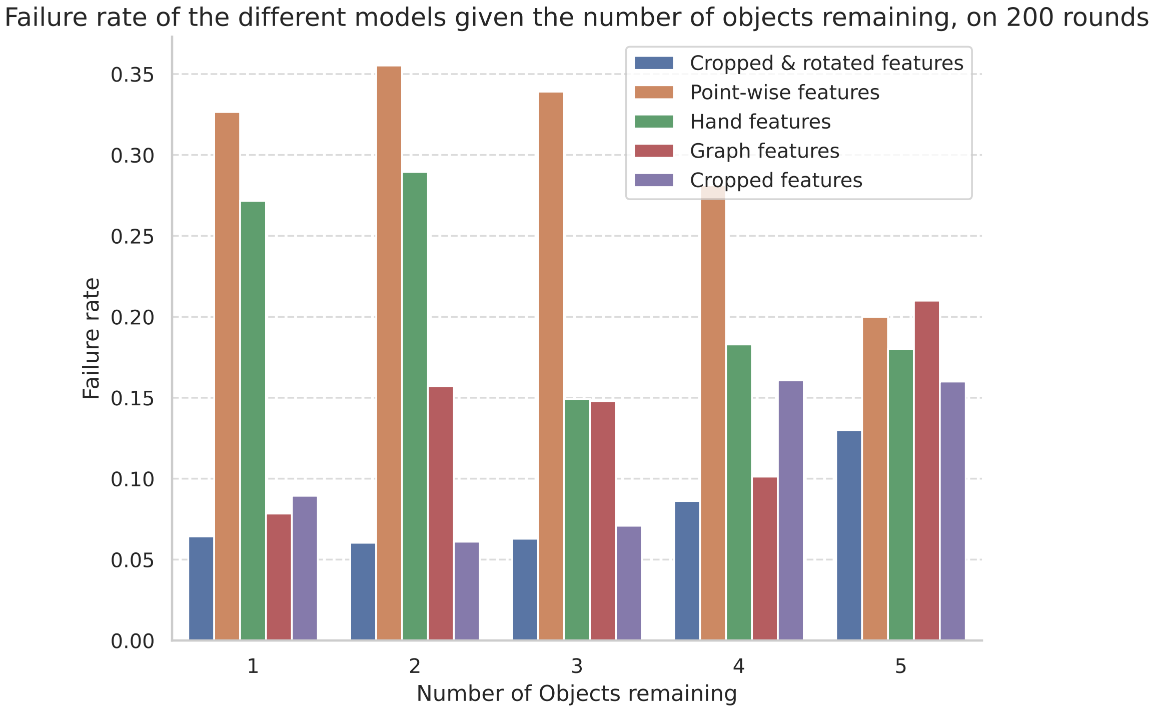


Figure 7.2. Failure rates of the different models based on the number of objects remaining, referred to as stages. The analysis was performed in a simulated environment, encompassing 200 rounds of experiments.

contributing to failure in these cases. Other factors, such as incorrect gripper position and orientation, may also play a role. Also models may give priority to the easiest objects and struggle with the more complex ones, resulting in a higher failure rate when fewer objects remain. Ideally, the failure rate should remain constant regardless of the number of objects remaining, assuming there is sufficient space between objects, as observed in the "packed" scenario. However, closer examination of the *Cropped & rotated features* and *Cropped features* models reveals that their failure rates stabilize when three or fewer objects remain, but increase for four or five objects. This suggests that the size of the gripper significantly affects these failures. To improve the percent cleared and success rate, t might be useful to use a smaller gripper capable of not colliding with other objects during grasping.

Comparison with Benchmarks Models To evaluate the effectiveness of our approach, we compared our best model with two existing methods: Grasp Pose Detection (GPD) [51] and Volumetric Grasping Network (VGN) [40] in terms of success rate and percentage cleared. The evaluation was carried out using the same dataset and a similar scenario, and the results are summarized in Table 7.2.

Our model achieved a high success rate of **91.37**%, which is very close to the best success rate achieved by VGN (91.5%). However, it should be noted that our model operates in a more constrained environment, with only 4 degrees of freedom (DoF) compared with 6 DoF for the other models. This limited range of motion for the gripper increases the likelihood of collisions and significantly reduces the range of all possible poses. As a result, our model has a slightly lower percent cleared than VGN.

Furthermore, it is important to highlight that the gripper used in our approach is primarily designed for large objects, which makes it more difficult to grasp smaller objects. This design choice contributes for some of the limitations in our model's performance.

Comparison with the Previous Model We achieved a success rate of **97**% on a single object, outperforming the previous model [2], which achieved a success rate of 91%. In addition, the samples drawn for the prior distribution achieved a success rate of 17%, whereas the previous model was below 1% [52].

7.3.2 Real-world results

Method	Success rate (%)	% cleared
Simulation results		
GPD	73.7	72.8
VGN ($\varepsilon = 0.95$)	91.5	79
VGN ($\varepsilon = 0.9$)	87.6	80.4
VGN ($\varepsilon = 0.85$)	80.4	79.9
Ours	91.37	75.25
Real-world results		
Ours	95.6	88

Table 7.2. Success rates (%) and percent cleared for picking experiments for the packed scenario with 5 objects over 200 rounds. (Since the publication of our paper [3], our model has been trained longer and with more data, which explains the increase in success rate).

Moving on to real-world results, we conducted 20 rounds of experiments with 5 objects selected from a set of 13 unknown objects (see Fig. 7.3). We achieved an outstanding success rate of **95.6%** and a percent cleared of **88%**. These results demonstrate the strong adaptability and performance of our approach in real-world scenarios. In both simulation and real-world settings, the majority of failure cases are due to insufficient friction forces, causing the objects to slip.

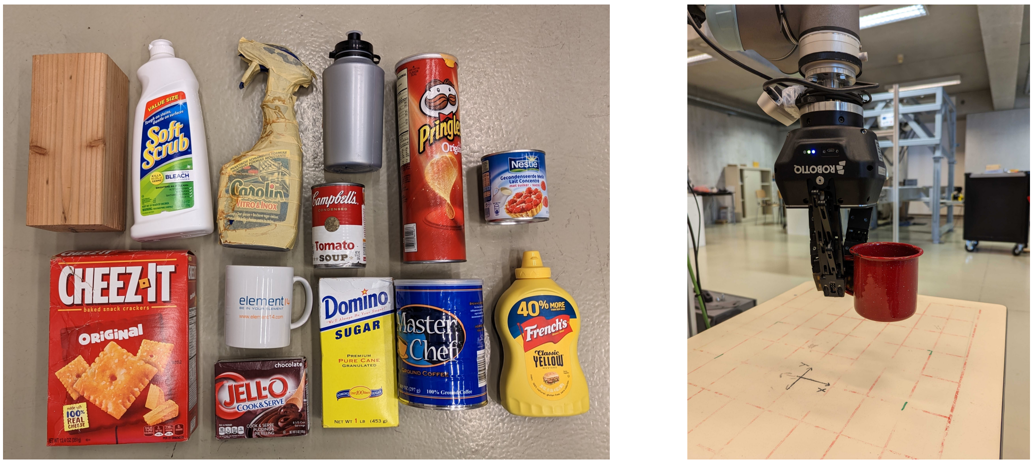


Figure 7.3. (left) Object assets used in the real setup. (right) Successful grasp of a mug.

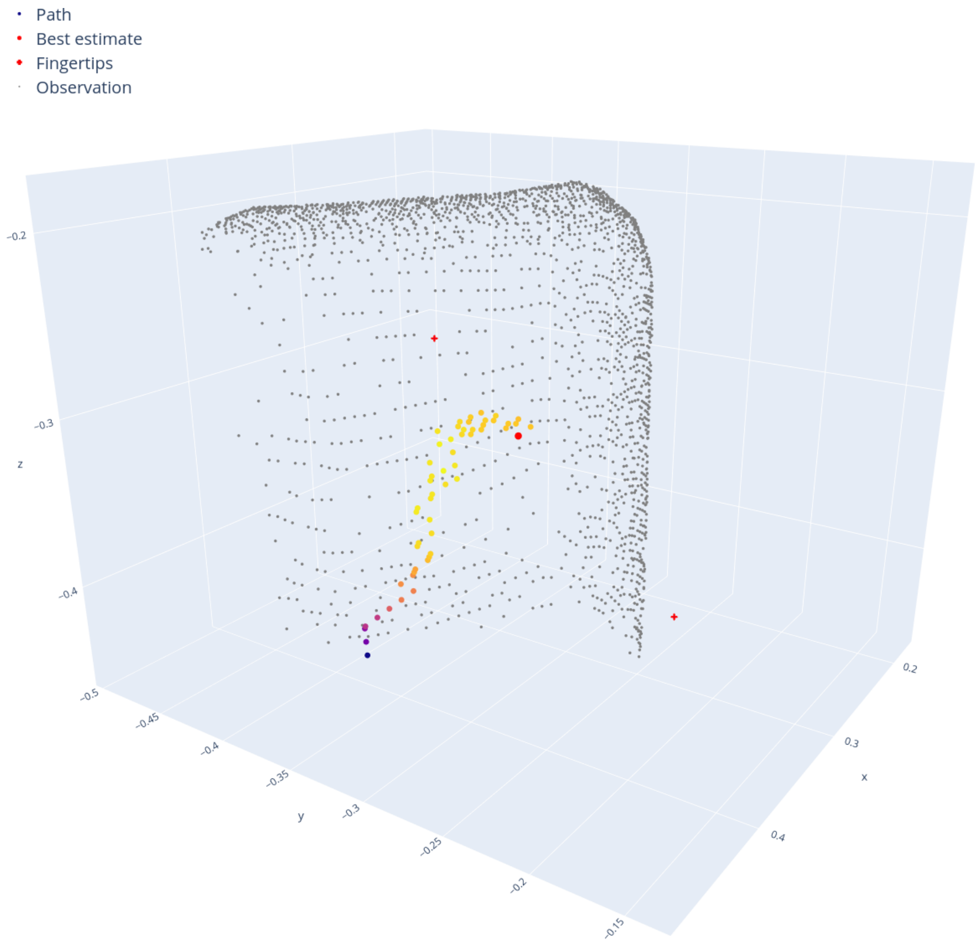
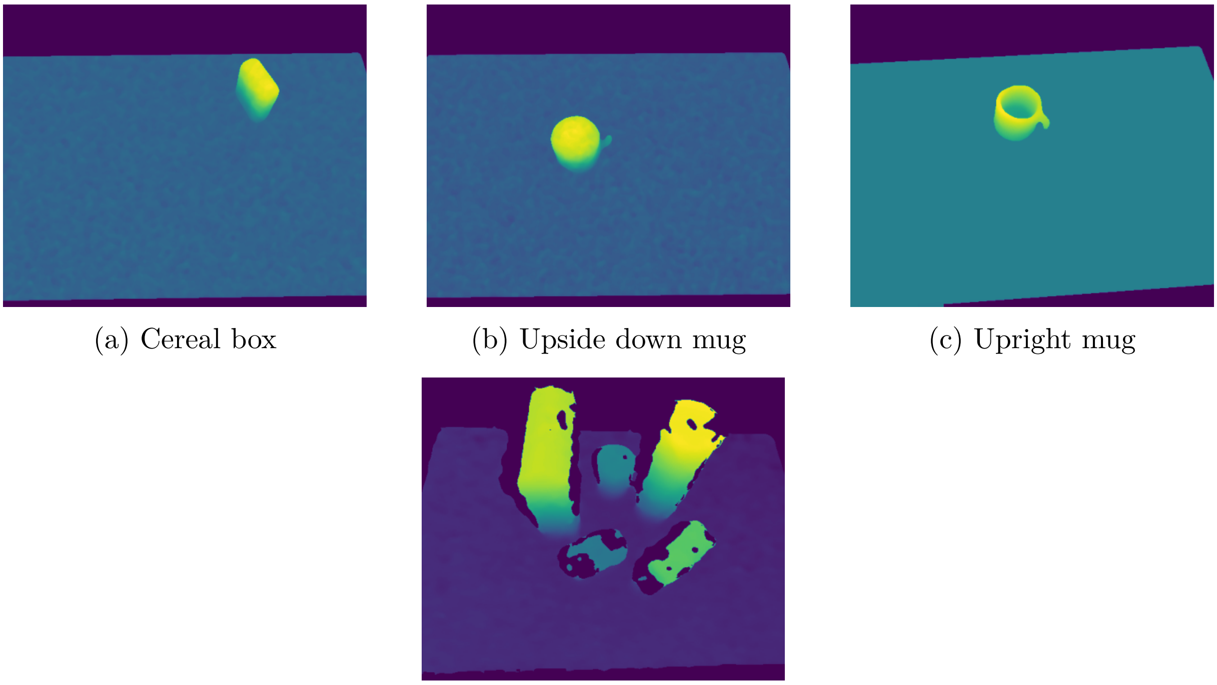


Figure 7.4. Gradient-based optimization is performed for 300 iterations to estimate the maximum a posteriori $\mathbf{h}^* = \arg\max_{\mathbf{h}} \ \hat{p}(\mathbf{h}|S = 1, o = 1, \mathbf{P})$. Starting from a random location, the path gradually converges towards the center of the object. The best estimate is marked by a red dot, and the crosses represent the tips of the gripper's finger. The color of the path ranges from purple to yellow, indicating lower to higher values of the posterior value.

7.4 MAP estimate

Figure 7.4 shows an example of posterior optimization for 300 iterations using *Random & Cropped features* model to estimate $\mathbf{h}^* = \arg\max_{\mathbf{h}} \ \hat{p}(\mathbf{h}|S = 1, o = 1, \mathbf{P})$ as explained in Section 6.2. The optimization process starts from a randomly chosen starting point in the object's bounding box. Throughout the iterations, the grasp position gradually moves towards the center of the object, eventually reaching a convenient position. Additionally, the orientation of the gripper fingers is aligned parallel to the object, to ensure successful grasping.



(a) Cereal box

(b) Upside down mug

(c) Upright mug

(d) Multi-object

Figure 7.5. Baseline Scenarios: Images (a), (b), and (c) are simulated scenarios, while image (d) is a real depth image.

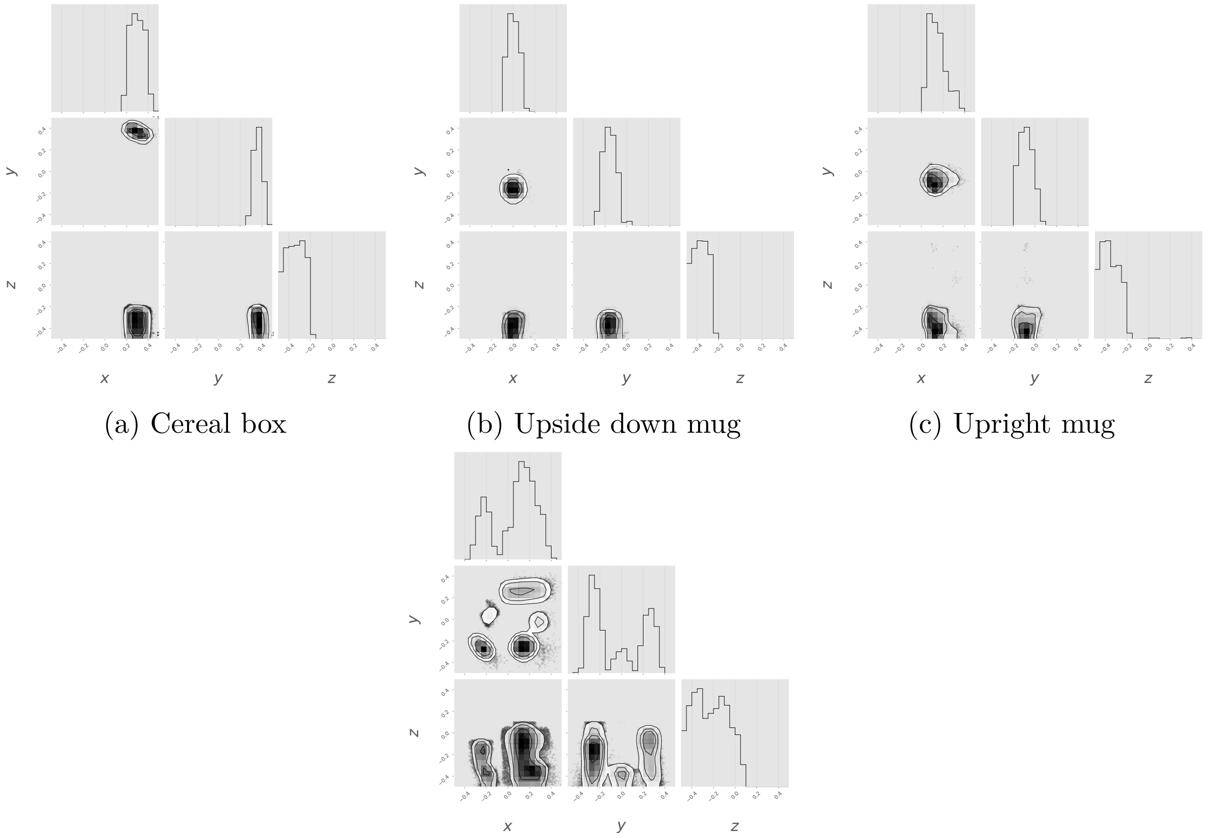
7.5 Prior and posterior analysis

Previous section provided a quantitative analysis of our model's performance. However, to gain a deeper understanding of how the model makes decisions based on its input, further investigation is required. This section aims to compare the posterior and prior distributions of four distinct scenes (refer to Figure 7.5) to provide insight into the model's decision-making process. Each scene represents a unique scenario, a rectangular box, an upright and inverted convex shape (mug), and a multi-object scene. The first three scenes are simulated, while the fourth scene is based on a real depth image.

7.5.1 Prior over the position

To sample $\mathbf{x}$ from $p(\mathbf{x} \mid o = 1, \mathbf{P})$, as explained in Section 5.4.2, we use HMC with specific hyperparameters. These include 200 chains with $25,000$ transitions each, and a burn-in of $1,000$. The integration parameters used are $\epsilon = 0.01$ and $L = 20$. The initial points for each chain are uniformly sampled within the bounding box of the objects.

Figure 7.6 illustrates the resulting distributions with a corner plot for the different scenarios, and Figure 7.7 shows these samples in 3D. These distributions accurately reflect the locations, shapes, and potential grasping points of objects in the four scenarios. Notably, in the multi-object scene (Figures 7.6d, 7.7d), the distribution accurately represents all five objects. This indicates that the prior distribution is informative and robust, even when dealing with a real scene. However, for convex objects, it does not assign density to the interior of the mug, as it is not an occupied area, showing the limitation of this method. But when the convex area is hidden, as in Figure 7.7b, it does not recognize that it is a convex shape and assigns a density to the interior of the convex region.



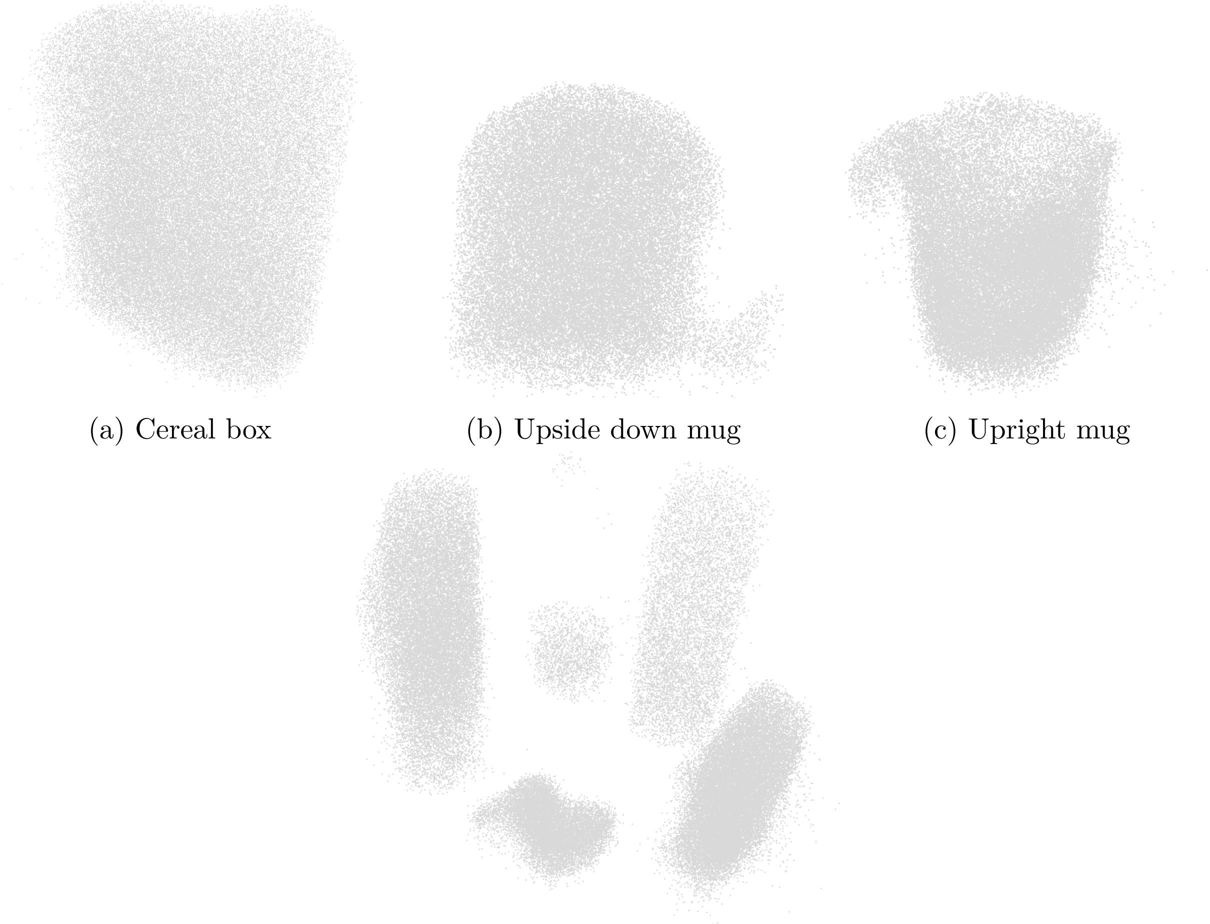
(a) Cereal box

(b) Upside down mug

(c) Upright mug

(d) Multi-object

Figure 7.6. Corner plot illustrating hand poses sampled from the **Prior:** $\mathbf{x} \sim p(\mathbf{x} \mid o = 1, \mathbf{P})$, obtained using HMC for the four baseline scenarios depicted in Figure 7.5.



(a) Cereal box

(b) Upside down mug

(c) Upright mug

(d) Multi-object

Figure 7.7. 3D visualization illustrating the grasping points sampled from the **Prior:** $\mathbf{x} \sim p(\mathbf{x} \mid o = 1, \mathbf{P})$, obtained using HMC for the four baseline scenarios depicted in Figure 7.5.

7.5.2 Posterior

Similarly to the prior distribution, we can sample from the posterior, conditioned on a successful grasp and an occupied region. Specifically, we sample $\mathbf{h}$ from the distribution $\hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P}) = r_\phi(S = 1 \mid \mathbf{h}, o = 1, \mathbf{P}) p(\mathbf{h} \mid o = 1, \mathbf{P})$. However, since the likelihood is not available, we use a likelihood-free version of HMC extended with a geodesic integrator, following the approach proposed by Norman et al. [3], with the following hyper-parameters: 250 chains with $100,000$ steps, a burn-in period of $5,000$ steps, and a thinning factor of 150. The integration parameters for HMC are set to $T = 20$ and $\epsilon = 0.01$. To gain insight into the importance of the features used in r_ϕ, we compare the resulting posterior samples obtained from the *Point-wise features* and *Cropped & Rotated features* models.

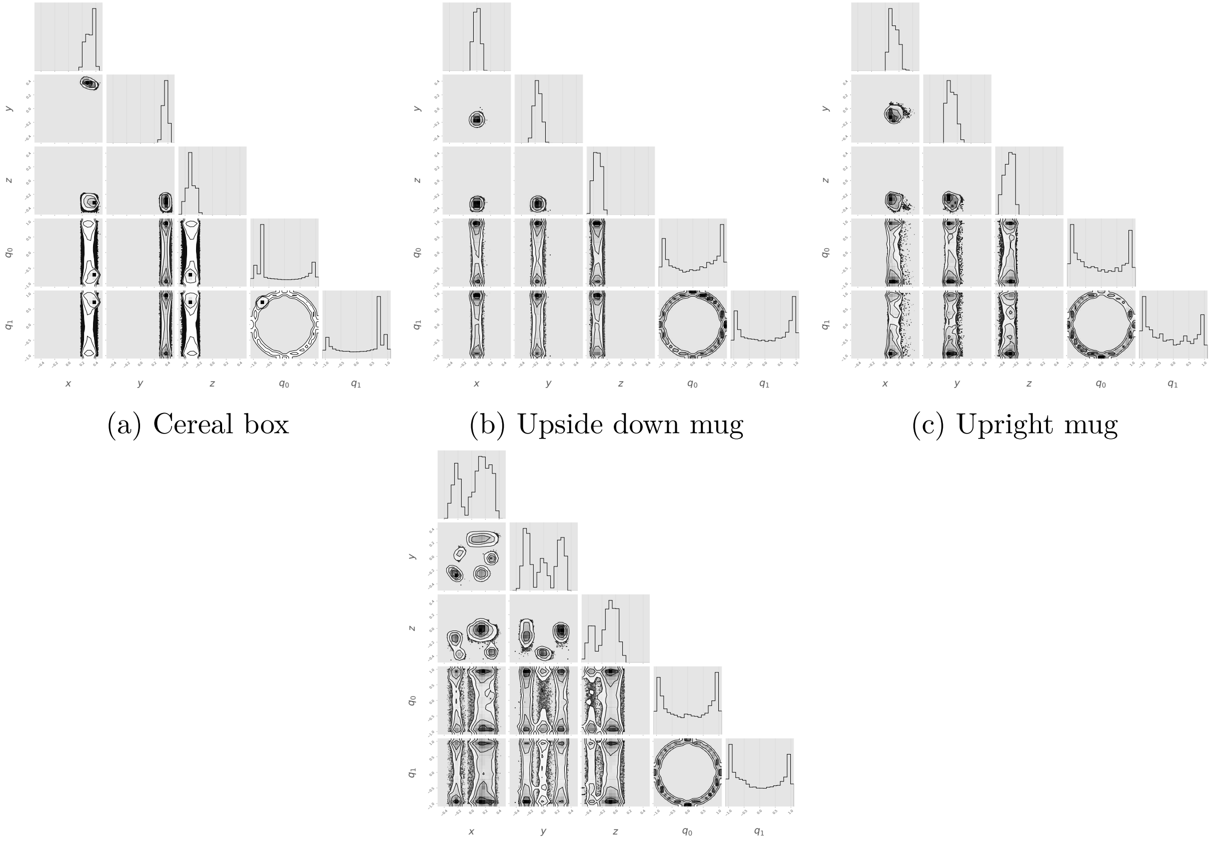
Corner plots and 3D visualizations are available for both models:

- *Point-wise features*: Figures 7.8, 7.9,
- *Cropped & Rotated features*: Figures 7.11, 7.12.

Position For both models, the estimated posterior tends to prioritize grasping objects from the top rather than the bottom. However, the *Point-wise features* model does not seem to take into account the presence of other objects in the surrounding, since it assigns density to all objects, even when they cannot be grasped due to collision (Figure 7.9d). In contrast, the *Cropped & Rotated features* model assigns a higher density to grasps points located on the top of the highest objects, as shown in Figure 7.12d. This approach minimizes the risk of collisions with other objects and ensures that the gripper's palm does not touch the object.

As expected, the limitations of the model become evident when examining the upright cup scene (Figures 7.9c and 7.12c). The posterior distribution is conditioned to grasp the object within an occupied region, and thus primarily considers grasping points located within the object. However, in the case of the upright cup, the best way to grasp it would be to put the grasping point at the center of the mug, which is not occupied. Instead, the model puts densities on the walls of the cup. On the other hand, when the mug is turned upside down (Figures 7.9b and 7.12b), the model perceives it as a filled object and allows grasping points even in areas where there are empty space in the real world.

Orientation Additionally, Figure 7.10 and Figure 7.13 provides insights into the hand orientation for the different grasping points in the multi-object scene, comparing the *Point-wise features* and *Cropped & Rotated features* models. It shows that for the *Point-wise features* model, the orientation is mainly uniform, even when objects are on the path. This uniform distribution can also be observed in the corner plot (Figure 7.8d). In the case of the *Cropped & Rotated features* model, the orientation is not uniform. For example, for the large rectangular box, it assigns higher densities to grasp it by the thinner side, because it knows that the other side of the box is larger than the size of the hand. Moreover, in the right figure, where we see the scene from the side, for the small object with low densities, it only allows the hand to rotate when there are no collisions with other objects. This is not the case with the *Point-wise features* model, which remains uniform even if the gripper collides with other objects.



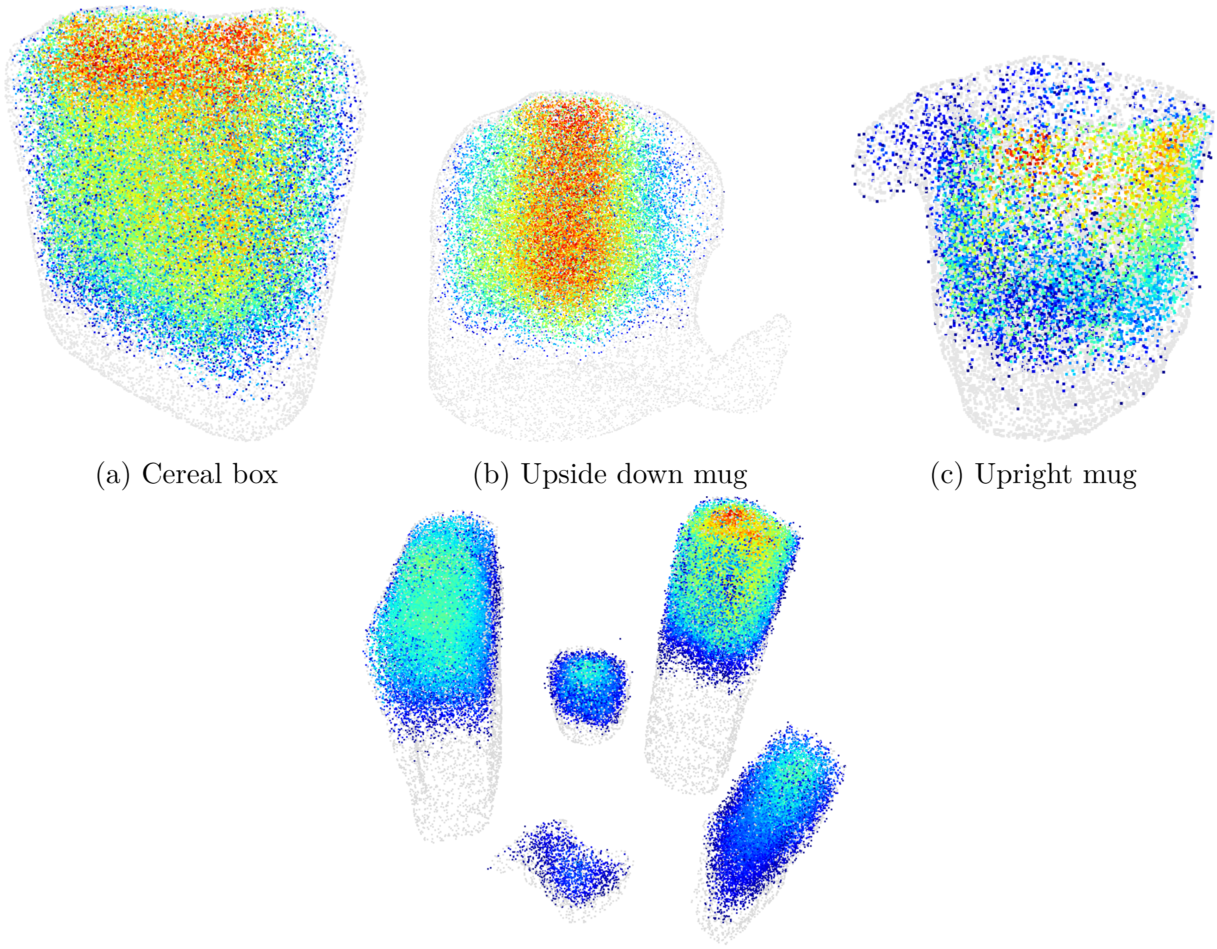
(a) Cereal box

(b) Upside down mug

(c) Upright mug

(d) Multi-object

Figure 7.8. Corner plot illustrating hand poses sampled from the **Posterior:** $\mathbf{h} \sim \hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$, obtained using geodesic likelihood-free HMC for the four baseline scenarios depicted in Figure 7.5. Using an ensemble of five **Point-wise features** ratio to estimate the posterior.



(a) Cereal box

(b) Upside down mug

(c) Upright mug

(d) Multi-object

Figure 7.9. 3D visualization illustrating grasping points sampled from the **Posterior:** $\mathbf{x} \sim \hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$, obtained using HMC for the four baseline scenarios depicted in Figure 7.5. Using an ensemble of five **Point-wise features** ratio to estimate the posterior. The color gradient ranging from blue to red indicates the relative posterior values assigned to each point. The grey-colored points correspond to the 3D point cloud of the object reconstructed using the occupancy network.

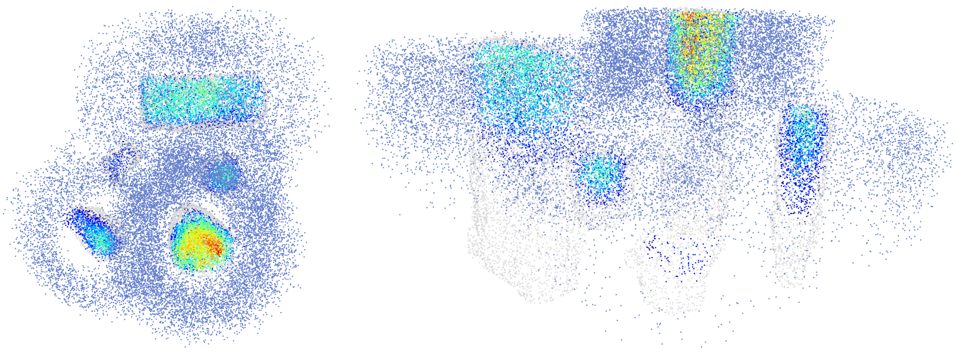
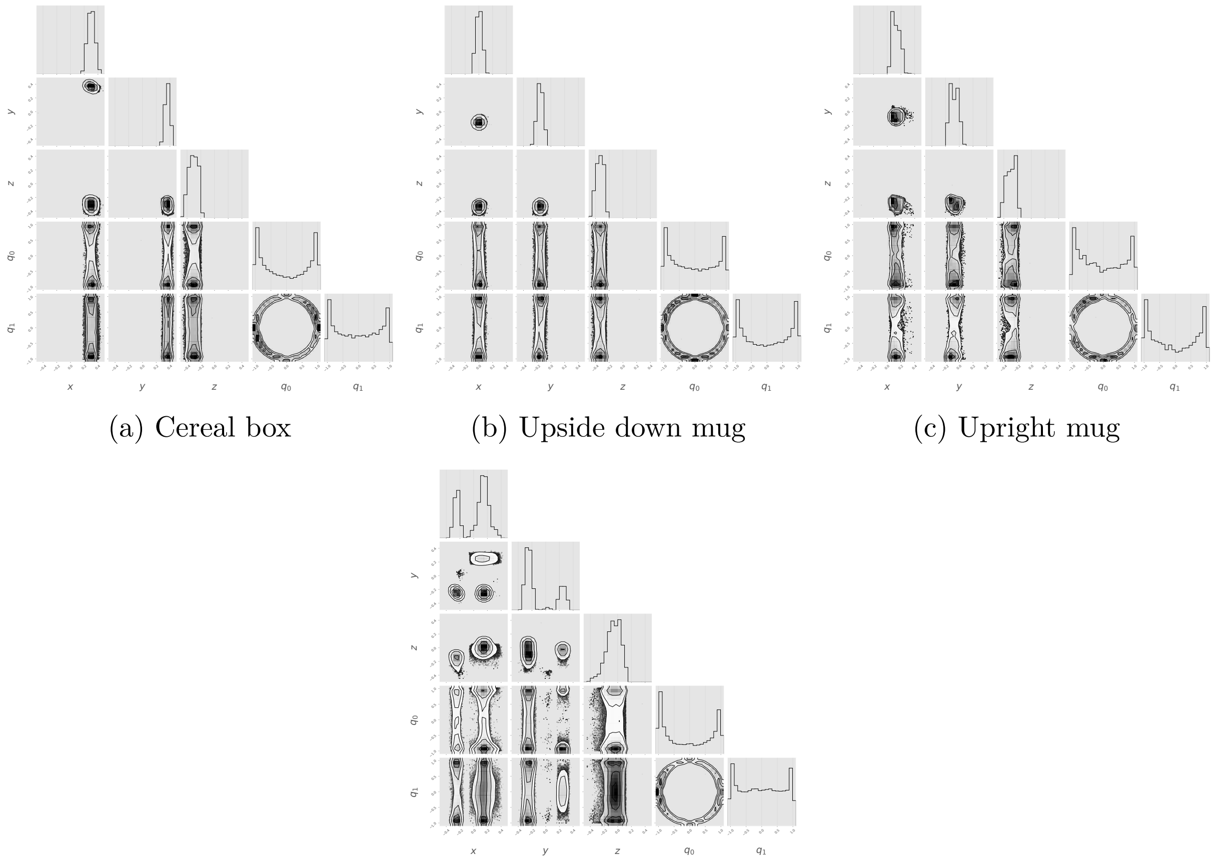


Figure 7.10. Top and side view of the "Multi-object" scene, illustrating gripper pose sampled from the **Posterior:** $\mathbf{h} \sim \hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$. Using an ensemble of five **Point-wise features** ratio to estimate the posterior. The light blue points represent a pair of fingertips, indicating the orientation of the hand.



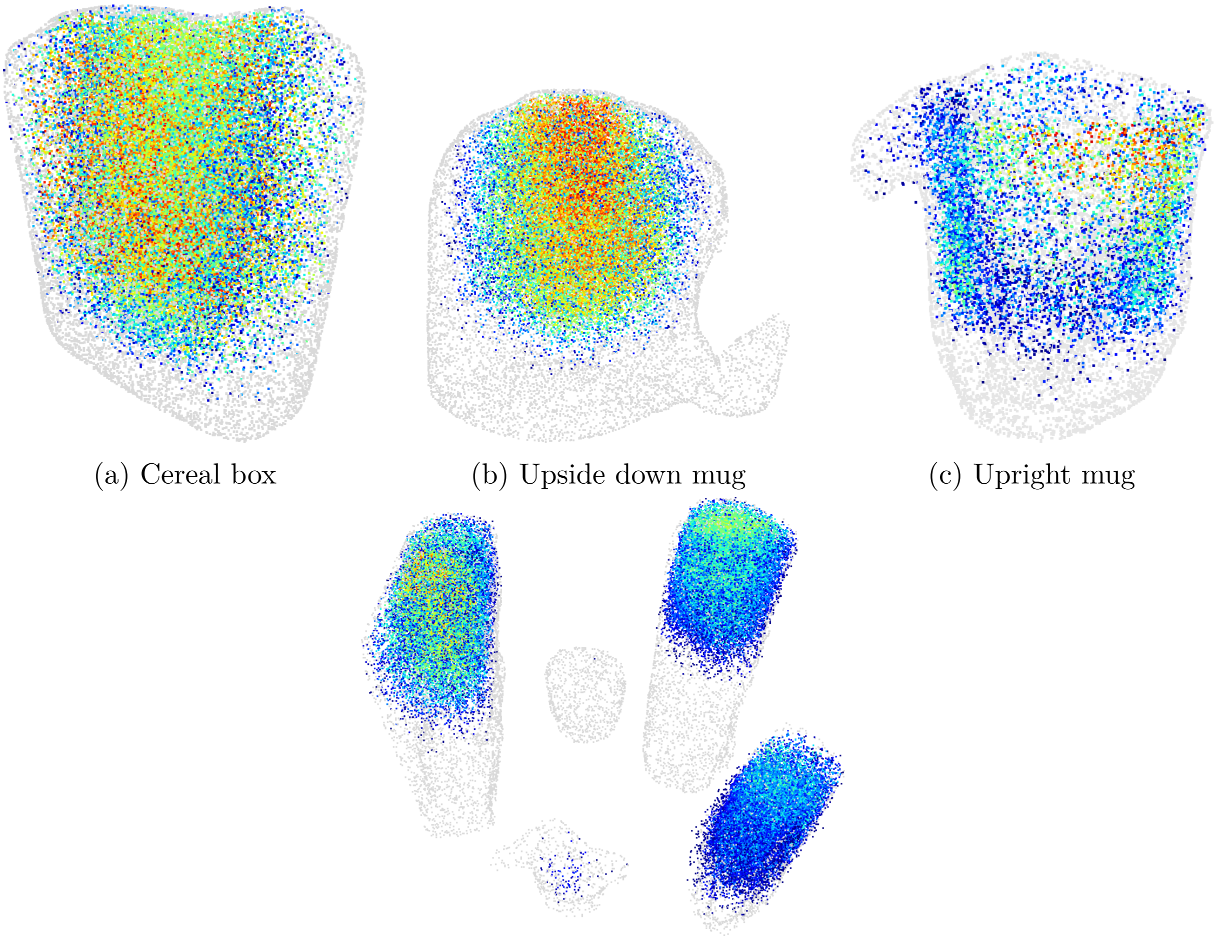
(a) Cereal box

(b) Upside down mug

(c) Upright mug

(d) Multi-object

Figure 7.11. Corner plot illustrating hand poses sampled from the **Posterior:** $\mathbf{h} \sim \hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$, obtained using geodesic likelihood-free HMC for the four baseline scenarios depicted in Figure 7.5. Using an ensemble of five **Cropped & Rotated features** ratio to estimate the posterior.



(a) Cereal box

(b) Upside down mug

(c) Upright mug

(d) Multi-object

Figure 7.12. 3D visualization illustrating grasping points sampled from the **Posterior:** $\mathbf{x} \sim \hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$, obtained using HMC for the four baseline scenarios depicted in Figure 7.5. Using an ensemble of five **Cropped & Rotated features** ratio to estimate the posterior. The color gradient ranging from blue to red indicates the relative posterior values assigned to each point. The grey-colored points correspond to the 3D point cloud of the object reconstructed using the occupancy network.

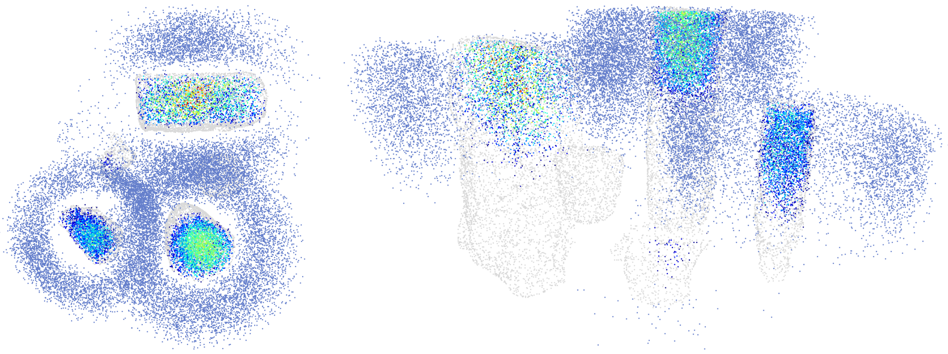


Figure 7.13. Top and side view of the "Multi-object" scene, illustrating gripper pose sampled from the **Posterior:** $\mathbf{h} \sim \hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$. Using an ensemble of five **Cropped & Rotated features** ratio to estimate the posterior. The light blue points represent a pair of fingertips, indicating the orientation of the hand.

7.6 Ablation study

	N° blocks	-	SR ↑
Point-wise features	3		**71.26**
Point-wise features	5		68.72
	fingertips	palm	SR ↑
Hands features	✓	✗	78.25
Hands features	✗	✓	72.54
Hands features	✓	✓	**79.23**
	k-NN.	-	SR ↑
Graph features	10		74.75
Graph features	1024		82.27
Graph features	2048		**85.58**
	N° conv.		SR ↑
Cropped features	0		72.54
Cropped features	1		85.52
Cropped features	2		**88.16**
Cropped features	3		69.24
	N° conv.		SR ↑
Rotated & Cropped features	0		75.99
Rotated & Cropped features	1		88.92
Rotated & Cropped features	2		**91.37**
Rotated & Cropped features	3		71.64

Table 7.3. Ablation study of our different architectures. SR is averaged over 200 rounds using objects from the test set and the packed scenario.

We conducted an ablation study to validate the design parameters of our architecture, as presented in Table 7.3. Starting with the *Point-wise features* the "N° blocks" column indicates the number of residual blocks used in the architecture. Based on the table, we determined that using three blocks yielded the best performance, and consequently, we maintained this number for the other models.

Regarding the *Hand features* model, we explored different configurations by considering only the finger tips or the palm. Our results indicate that incorporating these two features results in better performance, and that using the palm alone in addition to the point features does not bring much improvement, as it is simply a vertical translation. It is worth noting that the increase in success rate compared to the *Point-wise features* model across all three versions emphasizes the significance of integrating hand-related information.

In the context of the *Graph features*, the "k-NN" column represents the number of points in the point cloud used when querying a specific position $\mathbf{x}$. Our findings suggest that a larger number of points generally leads to improved success rate.

For the last two models, namely *Cropped features* and *Rotated & Cropped features*, we tried varying the number of convolutions applied to the cropped feature planes $\mathbf{c}_{xy}$. Based on the success rate, using two convolution layers produces the best results.

7.7 Discussion

Building upon the previous method proposed by Marlier et al. [2], our improved model surpasses its performance. Specifically, we achieve a 97% success rate on one object, outperforming the 91% success rate of the previous model. Moreover, while the previous model was limited to one object, ours can handle multiple objects while maintaining its performance. Furthermore, the quality of the samples drawn from the prior distribution improves significantly, with the sample success rate rising from 1% to 17%. However, theses results must be interpreted with caution, as the previous model had 6 degrees of freedom, which allows more ways of grasping the object but also makes the learning process more difficult. In our experiments, we performed the grasp with 4 degrees of freedom.

We demonstrate quantitatively and qualitatively that incorporating point-wise and local information about the grasping position significantly improves the performance of our method. The posterior estimated by the trained ratio and based on an occupancy prior is both consistent and efficient, underlining the significance of our approach and providing valuable explainability compared to other black box methods. Its stochastic nature can be advantageous in cases where a grasp attempt fails, as it allows for the selection of an alternative grasp point, whereas other methods usually predict a unique point.

In addition, our model achieves results comparable to those of benchmark methods, indicating the effectiveness of our approach. Furthermore, our approach demonstrates remarkable transferability from simulation to reality. Despite having been trained entirely on simulated data, the ratio and the occupancy network successfully maintain, and even improve, their performance when applied to real-world scenarios. This result underlines the robustness and generalizability of our method.

Nevertheless, we observed that the gripper used in our experiments was not fully adapted to the task, resulting in failures not due to predicted grasp points, but rather due to the gripper colliding with objects. Furthermore, the main drawback of our method is the time required to estimate the maximum a posteriori. Optimization involves making 300 passes on the network, resulting in a considerable computation time of around 24 seconds using a Quadro RTX 6000 GPU.

Chapter 8

Conclusion

Our work directly addresses deficiencies of previous simulation-based inference methods for robotic grasping, which relied on an uniform prior for the grasping position on the object bounding box. By incorporating a more informative prior, we demonstrate significant improvements in our results. In addition, our model achieves a success rate comparable to that of the most recent method.

One notable weaknesses of the previous approach was the long acquisition time required. We solved this problem by capturing a partial view of the scene which provides sufficient information for our model to accurately represent the entire scene. This approach significantly reduces the acquisition time while improving performance.

In addition to the aforementioned results, our method offers a high degree of transferability from simulation to reality. This means that the model trained on simulated data can be successfully applied to real-world scenarios, underlining the robustness and generalizability of our approach.

Nevertheless, there is still room for improvement in our approach. The development of more efficient methods for determining the optimal hand pose is an area worthy of further investigation. In addition, it is essential to address the challenges associated with controlling the complete 6 degrees of freedom of the robotic hand. This could involve the design of new features, such as the creation of a grid of voxels around the grasping point, as an alternative to the current approach of using cropped and rotated features of xy planes.

Bibliography

[1] Norman Marlier, Olivier Brüls, and Gilles Louppe. “Simulation-based Bayesian inference for multi-fingered robotic grasping”. In: *CoRR* abs/2109.14275 (2021). arXiv: 2109.14275. URL: https://arxiv.org/abs/2109.14275 (page 2).

[2] Norman Marlier, Olivier Brüls, and Gilles Louppe. “Simulation-based Bayesian inference for robotic grasping”. 2023. arXiv: 2303.05873 [cs.RO] (pages 2, 44, 52, 54, 58, 69).

[3] Norman Marlier et al. “Implicit representation priors meet Riemannian geometry for Bayesian robotic grasping”. In: *arXiv preprint arXiv:2304.08805* (2023) (pages 4, 7, 40, 44, 52, 58, 63).

[4] Martın Abadi et al. “Tensorflow: a system for large-scale machine learning.” In: *Osdi*. Vol. 16. 2016. Savannah, GA, USA. 2016, pp. 265–283 (page 6).

[5] Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660 (pages 6, 13).

[6] Panos Achlioptas et al. “Learning representations and generative models for 3d point clouds”. In: *International conference on machine learning*. PMLR. 2018, pp. 40–49 (page 6).

[7] Haoqiang Fan, Hao Su, and Leonidas J Guibas. “A point set generation network for 3d object reconstruction from a single image”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 605–613 (page 6).

[8] Andrew Brock et al. “Generative and discriminative voxel modeling with convolutional neural networks”. In: *arXiv preprint arXiv:1608.04236* (2016) (page 6).

[9] Matheus Gadelha, Subhransu Maji, and Rui Wang. “3d shape induction from 2d views of multiple objects”. In: *2017 International Conference on 3D Vision (3DV)*. IEEE. 2017, pp. 402–411 (page 6).

[10] Despoina Paschalidou et al. “Raynet: Learning volumetric 3d reconstruction with ray potentials”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3897–3906 (page 6).

[11] Angjoo Kanazawa et al. “End-to-end recovery of human shape and pose”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7122–7131 (page 6).

[12] Anurag Ranjan et al. “Generating 3D faces using convolutional mesh autoencoders”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 704–720 (page 6).

[13] Nanyang Wang et al. "Pixel2mesh: Generating 3d mesh models from single rgb images". In: *Proceedings of the European conference on computer vision (ECCV).* 2018, pp. 52–67 (pages 6, 7).

[14] Lars Mescheder et al. "Occupancy networks: Learning 3d reconstruction in function space". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2019, pp. 4460–4470 (pages 7, 9, 10, 13, 14, 17, 18, 29, 33, 86).

[15] Ben Mildenhall et al. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *CoRR* abs/2003.08934 (2020). arXiv: 2003.08934. URL: https://arxiv.org/abs/2003.08934 (pages 7–9).

[16] Jeong Joon Park et al. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2019 (pages 7, 8).

[17] Songyou Peng et al. "Convolutional occupancy networks". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16.* Springer. 2020, pp. 523–540 (pages 7, 18, 29, 33, 46).

[18] Brian Curless and Marc Levoy. "A volumetric method for building complex models from range images". In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996) (page 8).

[19] Diana Werner, Ayoub Al-Hamadi, and Philipp Werner. "Truncated Signed Distance Function: Experiments on Voxel Size". In: 8815 (Oct. 2014), pp. 357–364. DOI: 10.1007/978-3-319-11755-3_40 (page 8).

[20] Rohan Chabra et al. "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16.* Springer. 2020, pp. 608–625 (page 8).

[21] Kyle Genova et al. "Deep Structured Implicit Functions". In: *CoRR* abs/1912.06126 (2019). arXiv: 1912.06126. URL: http://arxiv.org/abs/1912.06126 (page 9).

[22] M. Levoy and P. Hanrahan. "Light field rendering". In: SIGGRAPH '96 () (page 9).

[23] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. "Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations". In: *CoRR* abs/1906.01618 (2019). arXiv: 1906.01618. URL: http://arxiv.org/abs/1906.01618 (page 9).

[24] Steven J. Gortler et al. "The Lumigraph". In: SIGGRAPH '96. 1996. DOI: 10.1145/237170.237200 (page 9).

[25] Thomas Müller et al. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: *ACM Trans. Graph.* 41.4 (July 2022), 102:1–102:15. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127 (page 10).

[26] Erwin Coumans and Yunfei Bai. "PyBullet, a Python module for physics simulation for games, robotics and machine learning". http://pybullet.org. 2016–2020 (page 10).

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". 2015. arXiv: 1505.04597 [cs.CV] (pages 11, 46).

[28] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456 (pages 11, 14).

[29] Lei Huang et al. “Normalization Techniques in Training DNNs: Methodology, Analysis and Application”. In: *CoRR* abs/2009.12836 (2020). arXiv: 2009.12836. URL: https://arxiv.org/abs/2009.12836 (page 11).

[30] Jeffrey Mahler et al. “Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics”. 2017. arXiv: 1703.09312 [cs.RO] (page 11).

[31] M. Fischler and R. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Communications of the ACM* 24.6 (1981), pp. 381–395. URL: /brokenurl#%20http://publication.wilsonwong.me/load.php?id=233282275 (page 11).

[32] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (page 13).

[33] Harm De Vries et al. “Modulating early visual processing by language”. In: *Advances in Neural Information Processing Systems* 30 (2017) (page 13).

[34] Ivaxi Sheth et al. “Pitfalls of Conditional Batch Normalization for Contextual Multi-Modal Learning”. In: *ArXiv* abs/2211.15071 (2022) (page 13).

[35] Congyue Deng et al. “Vector neurons: A general framework for so (3)-equivariant networks”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 12200–12209 (pages 14–18, 20, 29, 33).

[36] Yann LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE*. Vol. 86. 11. 1998, pp. 2278–2324. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665 (page 18).

[37] Yunlu Chen et al. “3D Equivariant Graph Implicit Functions”. In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part III*. Springer. 2022, pp. 485–502 (pages 20, 21, 33, 49).

[38] Yunlu Chen et al. “3D Equivariant Graph Implicit Functions”. In: *ECCV* (2022) (pages 21, 29).

[39] Berk Calli et al. “Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols”. In: *arXiv preprint arXiv:1502.03143* (2015) (page 22).

[40] Michel Breyer et al. “Volumetric Grasping Network: Real-time 6 DOF Grasp Detection in Clutter”. In: *Conference on Robot Learning*. 2020 (pages 22, 54, 57).

[41] Tariq M. Khan, Antonio Robles-Kelly, and Syed S. Naqvi. “T-Net: A Resource-Constrained Tiny Convolutional Neural Network for Medical Image Segmentation”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2022, pp. 644–653 (pages 23, 24).

[42] François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258 (page 23).

[43] Xumin Yu et al. “Pointr: Diverse point cloud completion with geometry-aware transformers”. In: *Proceedings of the IEEE/CVF international conference on computer vision.* 2021, pp. 12498–12507 (pages 31, 32, 88).

[44] Angel X Chang et al. “Shapenet: An information-rich 3d model repository”. In: *arXiv preprint arXiv:1512.03012* (2015) (page 31).

[45] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. “The frontier of simulation-based inference”. In: *Proceedings of the National Academy of Sciences* (2020) (page 42).

[46] Joeri Hermans, Volodimir Begy, and Gilles Louppe. “Likelihood-free MCMC with Amortized Approximate Ratio Estimators”. In: *Proceedings of the 37th International Conference on Machine Learning.* Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 4239–4248. URL: http://proceedings.mlr.press/v119/hermans20a.html (page 42).

[47] Kyle Cranmer, Juan Pavez, and Gilles Louppe. “Approximating likelihood ratios with calibrated discriminative classifiers”. In: *arXiv preprint arXiv:1506.02169* (2015) (page 42).

[48] Joshua S Speagle. “A conceptual introduction to markov chain monte carlo methods”. In: *arXiv preprint arXiv:1909.12313* (2019) (page 44).

[49] Radford M Neal et al. “MCMC using Hamiltonian dynamics”. In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2 (page 44).

[50] Joeri Hermans et al. “Averting a crisis in simulation-based inference”. In: *arXiv preprint arXiv:2110.06581* (2021) (page 47).

[51] Andreas Ten Pas et al. “Grasp pose detection in point clouds”. In: *The International Journal of Robotics Research* 36.13-14 (2017), pp. 1455–1473 (page 57).

[52] Norman Marlier, Olivier Brüls, and Gilles Louppe. “Simulation-based Bayesian inference for multi-fingered robotic grasping”. In: *arXiv preprint arXiv:2109.14275* (2021) (page 58).

[53] William E. Lorensen and Harvey E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 163–169. ISBN: 0897912276. DOI: 10.1145/37401.37422. URL: https://doi.org/10.1145/37401.37422 (page 86).

Appendix A

Setup

The robotic arm is a UR5 with a three-finger adaptive robot gripper from Robotiq attached to the end effector. However, for the purposes of this project, we opted to close the two adjacent fingers to simulate a two-finger clamp. While the robot has the capability to operate in 6 degrees of freedom (DoF), we specifically considered a 4 DoF configuration for this master thesis.

In the setup, objects are placed on a tabletop, and the robotic arm captures the scene using a depth camera (Intel RealSense) mounted on its flange. The depth image is captured from a predefined pose. Prior to my involvement in the project, the real setup was meticulously replicated in a simulation environment using PyBullet. Furthermore, the real robot is controlled using ROS2, enabling smooth integration between simulation and real-world execution.

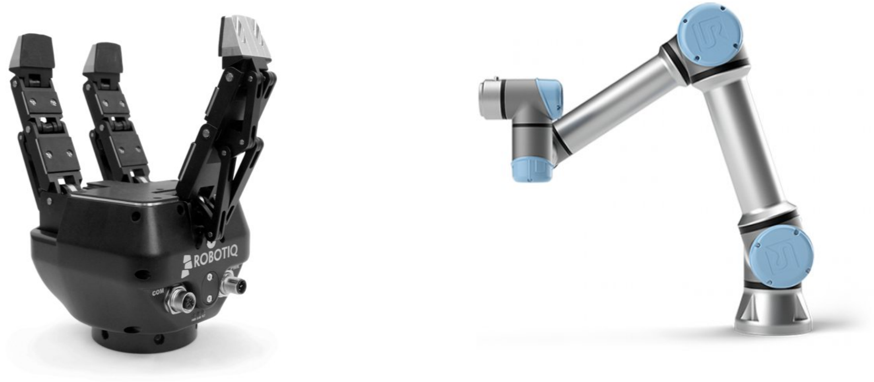


Figure A.1. (left) Three finger adaptive robot gripper from Robotiq. (right) The UR5.

Appendix B

Implicit representation priors meet Riemannian geometry for Bayesian robotic grasping

Implicit representation priors meet Riemannian geometry for Bayesian robotic grasping

Norman Marlier[1]* Julien Gustin[2] Olivier Brüls[3] Gilles Louppe[4]

***Abstract*— Robotic grasping in highly noisy environments presents complex challenges, especially with limited prior knowledge about the scene. In particular, identifying good grasping poses with Bayesian inference becomes difficult due to two reasons: i) generating data from uninformative priors proves to be inefficient, and ii) the posterior often entails a complex distribution defined on a Riemannian manifold. In this study, we explore the use of implicit representations to construct scene-dependent priors, thereby enabling the application of efficient simulation-based Bayesian inference algorithms for determining successful grasp poses in unstructured environments. Results from both simulation and physical benchmarks showcase the high success rate and promising potential of this approach.**

I. Introduction

Grasping is a fundamental skill for any robotic system. While current methods are effective for highly constrained tasks in structured environments, new and complex applications require increased flexibility and more advanced algorithms to account for the uncertainties that emerge in unstructured and noisy environments. Bayesian inference offers a well-principled approach to address these uncertainties; however, robotic tasks present unique challenges that make Bayesian inference difficult to apply, particularly for sampling-based algorithms. Firstly, many Bayesian approaches assume that the likelihood is tractable and can be evaluated, which is seldom the case in robotics. Secondly, parameters may span a vast space, leading to inefficient sampling strategies. Lastly, parameters of interest often belong to smooth Riemannian manifolds, further complicating the inference procedure. In this paper, we address these challenges by designing informative scene-dependent priors and using simulation-based inference algorithms combined with geometric sampling methods. Our contributions are summarized as follows:

- We integrate simulation-based Bayesian inference methods [1] with 3D implicit representations for robotic grasping.
- We adapt geodesic Monte Carlo [2] with a neural ratio estimator to sample on Riemannian manifolds with an intractable likelihood.
- We validate our method through simulated and real experiments, demonstrating promising grasping performance.

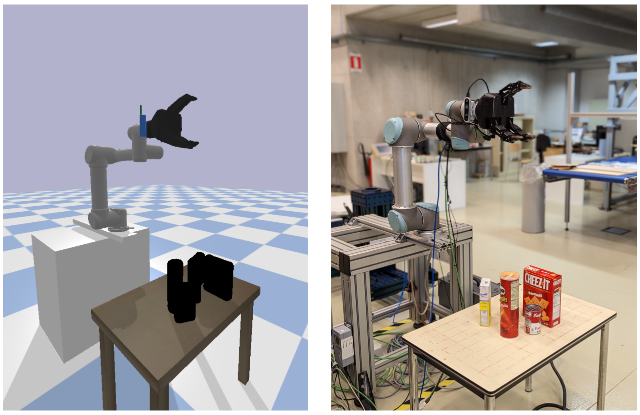


Fig. 1: Our benchmark scene. (left) The simulated environment. (right) The real setup.

II. Problem statement

We consider the problem of planning 4-DoF hand configurations for a robotic gripper handling various unknown objects on a table, observed with a depth camera. A benchmark scene is shown in Fig. 1.

A. Notations

Frames We use several reference frames in our work. The world frame $\underrightarrow{\mathcal{F}}_{\mathrm{W}}$ and the workspace frame $\underrightarrow{\mathcal{F}}_{\mathrm{S}}$ can be chosen freely and are not tied to a physical location. The world frame is used for the robot and the sensor, while the workspace frame is used for our inference system. $\underrightarrow{\mathcal{F}}_{\mathrm{C}}$ and $\underrightarrow{\mathcal{F}}_{\mathrm{E}}$ correspond respectively to the camera and the tool centre point.

Hand configuration The hand configuration $\mathbf{h} \in \mathcal{H} = \mathbb{R}^3 \times \mathbb{S}^1$ is defined as the pose $(\mathbf{x}, \mathbf{q}) \in \mathbb{R}^3 \times \mathbb{S}^1$ of the hand, where $\mathbf{x}$ is the vector $\vec{SE}$ expressed in $\underrightarrow{\mathcal{F}}_{\mathrm{S}}$ and $\mathbf{q}$ is the planar rotation represented with complex numbers defined in $\underrightarrow{\mathcal{F}}_{\mathrm{S}}$.

Binary metric A binary variable $S \in \{0, 1\}$ indicates if the grasp fails $(S = 0)$ or succeeds $(S = 1)$.

Observation Given the depth image I with its corresponding transformation camera to world $\mathbf{T}_{\mathrm{WC}}$ and camera intrinsic matrix K, we construct a point cloud $\mathbf{P} \in \mathbb{R}^{2048\times 3}$ expressed in $\underrightarrow{\mathcal{F}}_{\mathrm{S}}$.

Occupancy A binary variable $o \in \{0, 1\}$ indicates if a point $\mathbf{p} \in \mathbb{R}^3$ is occupied by any object of the scene.

Latent variables Unobserved variables $\mathbf{z}$ capture uncertainties about the nonsmooth dynamics of contact, the sensor noise, as well as the number of objects and their geometry.

*The authors come from the University of Liège, Belgium
[1] norman.marlier@uliege.be

B. Grasping as inference

We formulate the problem of grasping as the Bayesian inference of the hand configuration $\mathbf{h}^*$ that is a posteriori the most likely given a successful grasp, an occupancy o and a point cloud $\mathbf{P}$. That is, we are seeking the maximum a posteriori (MAP) estimate

$$\mathbf{h}^* = \arg\max_{\mathbf{h}} \; p(\mathbf{h}|S=1, o=1, \mathbf{P}), \tag{1}$$

from which we then compute the joint trajectory

$$\tau_{1:m} = \Lambda(\tau_0, \text{IK}(\mathbf{h}^*), \mathbf{P}), \tag{2}$$

where IK is an inverse kinematic solver, $\tau_{1:m}$ are waypoints in the joint space, $\tau_m = \text{IK}(\mathbf{h}^*)$ with $\mathbf{h}^*$ expressed in $\underrightarrow{\mathcal{F}}_{\text{W}}$ and Λ is a path planner.

III. Implicit representation of priors and posteriors for robotic grasping

From the Bayes rule, the posterior of the hand configuration is

$$p(\mathbf{h}|S, o, \mathbf{P}) = \frac{p(S \mid \mathbf{h}, o, \mathbf{P})}{p(S \mid o, \mathbf{P})} p(\mathbf{h} \mid o, \mathbf{P}), \tag{3}$$

which can be rewritten as the product of the likelihood-to-evidence ratio r and a scene-dependent prior

$$p(\mathbf{h}|S, o, \mathbf{P}) = r(S \mid \mathbf{h}, o, \mathbf{P}) p(\mathbf{h} \mid o, \mathbf{P}). \tag{4}$$

A. Priors

Position The scene-dependent prior over the position $\mathbf{x}$ is the distribution $p(\mathbf{x}|o, \mathbf{P}) = \frac{p(o|\mathbf{x},\mathbf{P})}{p(o|\mathbf{P})} p(\mathbf{x})$, where $p(o|\mathbf{x}, \mathbf{P})$ is the likelihood of the occupancy o, $p(\mathbf{x})$ is uniform over the workspace, and $p(\mathbf{x}|\mathbf{P})$ is simplified to $p(\mathbf{x})$ by independence.

We model the occupancy likelihood $p(o|\mathbf{x}, \mathbf{P})$ using a Convolutional Occupancy Network [3]. This network computes the occupancy by first producing three canonical features planes $\mathbf{c}_{xy}(\mathbf{P}), \mathbf{c}_{xz}(\mathbf{P})$ and $\mathbf{c}_{yz}(\mathbf{P})$. Then, the bilinear interpolations of the three planes are used to compute $\psi(\mathbf{P}, \mathbf{x}) = \mathbf{c}_{xy}(\mathbf{P})(\mathbf{x}) + \mathbf{c}_{xz}(\mathbf{P})(\mathbf{x}) + \mathbf{c}_{yz}(\mathbf{P})(\mathbf{x})$. These point-wise features at point $\mathbf{x}$ are finally processed by a fully connected network, outputting the occupancy probability.

This implicit representation allows us to sample interesting grasping positions from $p(\mathbf{x}|o, \mathbf{P}) \propto p(o|\mathbf{x}, \mathbf{P})p(\mathbf{x})$. We use Hamiltonian Monte Carlo (HMC) to take advantage of the differentiability of the occupancy network.

Orientation The prior of the orientation $\mathbf{q}$ is a uniform distribution over the unit circle $\mathbb{S}^1$. This prior is *invariant* to any rotation $\mathbf{R} \in \text{SO}(2)$ applied to $\mathbf{q}$, satisfying $p(\mathbf{q}) = p(\mathbf{Rq})$. This property enables free selection of the reference frame on the table. Additionally, the prior can be extended to SO(3) by using quaternions on $\mathbb{S}^3$.

Hand configuration Finally, the prior of the hand configuration is $p(\mathbf{h} \mid o, \mathbf{P}) = p(\mathbf{x} \mid o, \mathbf{P})p(\mathbf{q})$.

B. Ratio

The likelihood function $p(S \mid \mathbf{h}, o, \mathbf{P})$ and the evidence $p(S \mid o, \mathbf{P})$ are both intractable. However, drawing samples from forward models remains feasible with physical simulators, hence enabling likelihood-free Bayesian inference algorithms. In particular, the likelihood-to-evidence ratio $r(S \mid \mathbf{h}, o, \mathbf{P})$ (Eq. (4)) can be approximated by a neural network $r_\phi(S \mid \mathbf{h}, o, \mathbf{P})$ using amortized neural ratio estimation [5]. Here, instead of using only point-wise features $\psi(\mathbf{P}, \mathbf{x})$ as input for the ratio, we add a crop of the features plane $\mathbf{c}_{xy}(\mathbf{P})$, centred at $\mathbf{x}$, sized by the gripper and rotated by $\mathbf{q}$. These features $\Psi(\mathbf{P}, \mathbf{h})$ local in the neighbourhood of the grasping point are nearly equivariant to a 2D transformation applied to the object, *i.e* $\mathbf{T}\Psi(\mathbf{P}, \mathbf{h}) \approx \Psi(\mathbf{TP}, \mathbf{h}), \mathbf{T} \in \text{SE}(2)$.

C. Posteriors

Given our scene-dependent prior and our likelihood-to-evidence ratio, we approximate the posterior over the hand configurations as

$$\hat{p}(\mathbf{h} \mid S, o, \mathbf{P}) = r_\phi(S \mid \mathbf{h}, o, \mathbf{P}) p(\mathbf{h} \mid o, \mathbf{P}). \tag{5}$$

This approximation defines an implicit function [6] on the product of manifolds $\mathbb{R}^3 \times \mathbb{S}^1$ that is both fully tractable and differentiable, allowing the use of gradient-based methods for computing the MAP and sampling. Therefore, we can use Markov Chain Monte Carlo methods to sample from our posterior approximation $\hat{p}(\mathbf{h} \mid S, o, \mathbf{P})$. In particular, based on [5], we use a likelihood-free version of HMC by replacing the intractable likelihood with the ratio. The potential energy function is defined as $U(\mathbf{h}) \triangleq -\log p(S \mid \mathbf{h}, o, \mathbf{P})$ and its difference is $U(\mathbf{h}_t) - U(\mathbf{h}') = \log r(S \mid \mathbf{h}_t, \mathbf{h}')$. The gradient used in the integration step is given by $\nabla_{\mathbf{h}} U(\mathbf{h}) = -\nabla_{\mathbf{h}} \log r(S \mid \mathbf{h}, o, \mathbf{P})$. To account for the geometry of the parameter space, we then further extend our likelihood-free HMC with a geodesic integrator. The geodesic Monte Carlo scheme uses geodesic flow to perform the integration while staying on the manifold. To this end, orthogonal projections and geodesics are needed in a closed form. Finally, geodesic Monte Carlo can be applied to a product of manifolds $\mathcal{M}_1 \times \mathcal{M}_2 : \{(x_1, x_2) : x_1 \in \mathcal{M}_1, x_2 \in \mathcal{M}_2\}$, such as $\mathbb{R}^3 \times \mathbb{S}^1$ in our specific case. Geodesic flow can be executed in parallel; only the evaluation of the ratio requires both variables $\mathbf{x}$ and $\mathbf{q}$. In this manner, we can sample from the posterior density defined on a smooth manifold with closed-form geodesic. The full sampling procedure is summarized in Algorithm 1 of Appendix A.

IV. Experiments

We assess our approach on a robotic grasping task in both simulation and real-world settings. We generate data in a *packed* scenario, as defined in [7]. Additional experiments can be found in Appendix B and C.

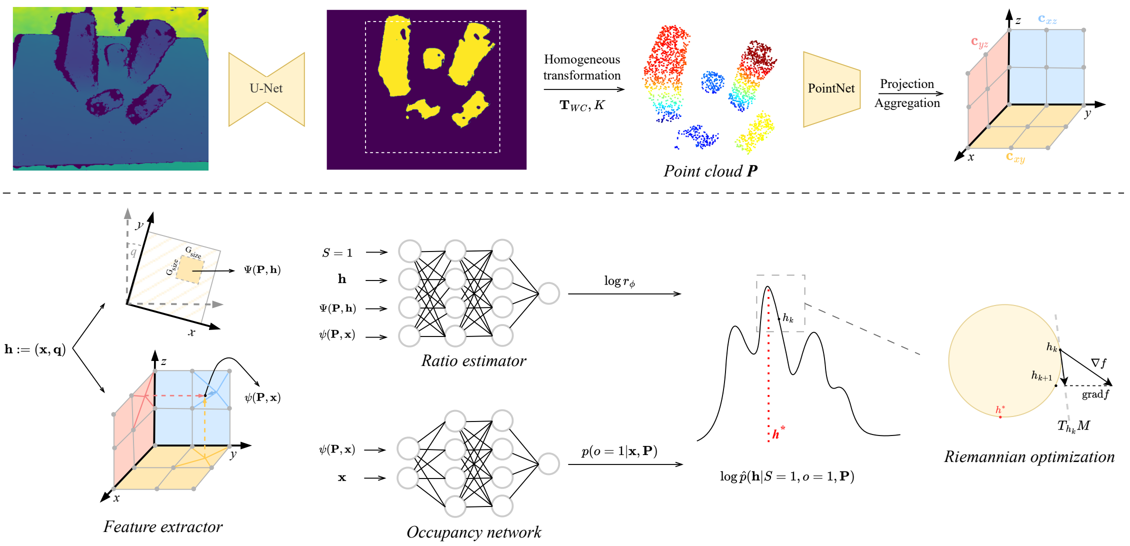

U-Net
Homogeneous transformation
$\mathbf{T}_{WC}, K$
Point cloud $\boldsymbol{P}$
PointNet
Projection
Aggregation
$\mathbf{c}_{yz}$
$\mathbf{c}_{xz}$
$\mathbf{c}_{xy}$
z
y
x
$\mathbf{h} := (\mathbf{x}, \mathbf{q})$
G_{size}
$\Psi(\mathbf{P}, \mathbf{h})$
$\psi(\mathbf{P}, \mathbf{x})$
Feature extractor
$S = 1$
$\mathbf{h}$
$\Psi(\mathbf{P}, \mathbf{h})$
$\psi(\mathbf{P}, \mathbf{x})$
Ratio estimator
$\log r_\phi$
$\psi(\mathbf{P}, \mathbf{x})$
$\mathbf{x}$
Occupancy network
$p(o = 1|\mathbf{x}, \mathbf{P})$
h_k
h^*
$\log \hat{p}(\mathbf{h}|S = 1, o = 1, \mathbf{P})$
h_k
h_{k+1}
∇f
grad f
h^*
$T_{h_k}M$
Riemannian optimization

Fig. 2: Our grasp inference pipeline, as run on the scene of Fig. 1. It begins with a noisy depth image of the scene, from which we first separate the objects from the background using a U-Net [4]. We then generate three canonical feature planes following the approach in [3]. To evaluate a given $\mathbf{h}$, we extract point-wise $\psi(\mathbf{P}, \mathbf{x})$ and local $\Psi(\mathbf{P}, \mathbf{h})$ features and feed them to the ratio and occupancy networks. Using the resulting differentiable posterior and Riemannian optimization, we finally identify the most plausible hand configuration $\mathbf{h}^*$.

A. *Grasp inference pipeline*

Starting from the depth image I, we remove the background and extract only pixels of the objects with a segmentation model based on a U-Net architecture [4]. Then, we convert I to a point cloud $\mathbf{P}$ with 2048 points, which passes through an encoder and produces three canonical feature planes. We then extract point-wise $\psi(\mathbf{P}, \mathbf{x})$ and local $\Psi(\mathbf{P}, \mathbf{h})$ features to evaluate the occupancy and the ratio networks. To smooth the posterior approximation, we use an ensemble of 6 ratio models. Finally, we compute the MAP by maximizing the log posterior density [8]. To this end, we use a Riemannian gradient ascent which preserves the nonlinearity of $\mathbb{S}^1$. A visual summary of our method is given in Fig. 2.

Our likelihood-free geodesic Monte Carlo is used to sample plausible hand configurations $\mathbf{h} \sim \hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$ for successful grasps, as shown in Fig. 3. Although our conditional prior distributes density across everywhere on the objects, the posterior assigns minimal density to the bottom of objects when multiple objects are present on the table. This occurs due to potential collisions between the gripper and the table, or the gripper and other objects. Regarding rotation, the posterior resembles the prior because multiple objects on the table, some with axial symmetry, allow for a wide range of grasp orientations. When only a single object is used, distinct modes can be observed, indicating that our posterior captures meaningful orientations, as shown in Fig. 4.

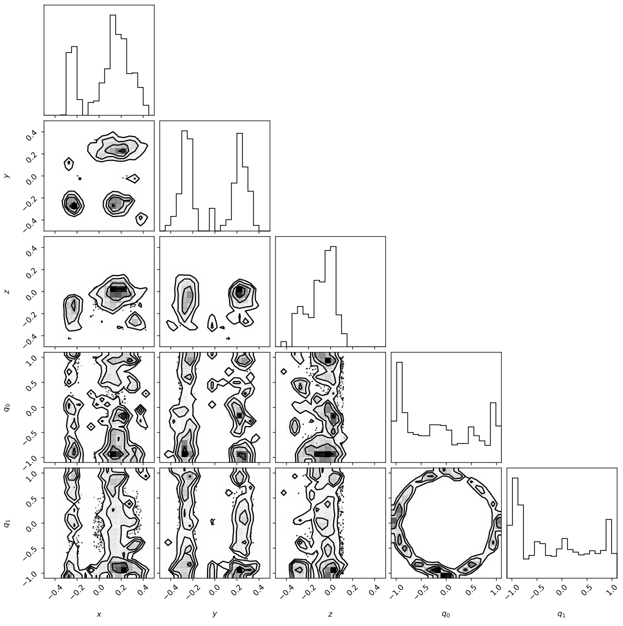


Fig. 3: Estimated posterior distribution $\hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$ of plausible successful hand configurations, for the scene shown in Fig. 1.

B. *Simulation results*

To compare our approach, we evaluated it against Grasp Pose Detection (GPD) [9] and Volumetric Grasping Network (VGN) [7] in terms of success rate and percent cleared [7] using the same dataset and a similar scenario (Table I). Our model achieved a high success rate of 91.1%, which is very close to VGN's best success rate of 91.5%. However, our model operated in a more constrained setup with 4 DoF instead of 6, limiting

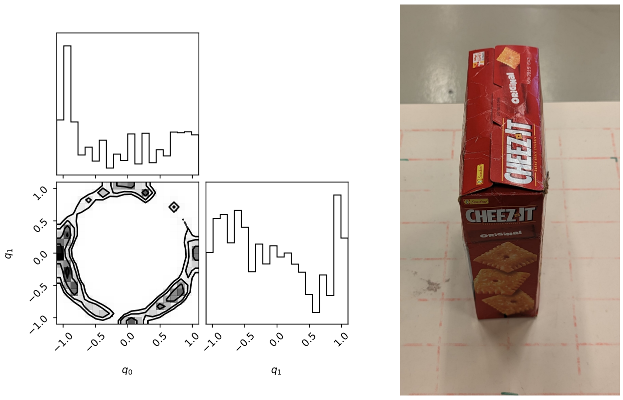


Fig. 4: (left) Estimated posterior distribution of the orientation $\hat{p}(\mathbf{q} \mid S = 1, o = 1, \mathbf{P})$. (right) Single object.

TABLE I: Success rates (%) and % cleared for picking experiments for the packed scenario with 5 objects over 200 rounds.

Method	Success rate	% cleared
Simulation results		
GPD [9]	73.7	72.8
VGN ($\varepsilon = 0.95$) [7]	91.5	79
VGN ($\varepsilon = 0.9$) [7]	87.6	80.4
VGN ($\varepsilon = 0.85$) [7]	80.4	79.9
Ours	91.1	77
Real-world results		
Ours	95.6	88

the gripper's movement and increasing the risk of collisions. This limitation resulted in a slightly lower percent cleared compared to VGN. Furthermore, the gripper used was primarily designed for larger objects, making it challenging to handle smaller ones.

C. *Real-world results*

We adopt the setup from [8]. We perform 20 rounds with 5 objects among 13 (see Fig. 5), using a protocol similar to the simulation experiments. The objects are chosen based on their availability in the lab and whether they were seen or unseen during training. For novel objects, we achieve a success rate of **95.6**% and a percent cleared of **88**%, showing the strong adaptability and performance of our approach. The discrepancy between the simulation and real-world setup is overcome without any decrease in performance. In both simulation and real-world settings, the majority of failure cases are due to insufficient friction forces, causing the objects to slip.

V. Related Work

Grasp sampling strategies that generate data for machine learning methods can be categorized based on their coverage of the hand configuration space $\mathcal{H}$ [10]. Simple strategies like uniform sampling provide direct density estimation but are highly inefficient. Heuristic methods, on the other hand, mainly rely on object geometry. Notably, the most efficient strategies [11], [12] are not suitable for complex settings such as multi-fingered grippers. Our occupancy networks-based approach offers direct density estimation, efficient sampling, and does not depend on specific object or gripper assumptions.

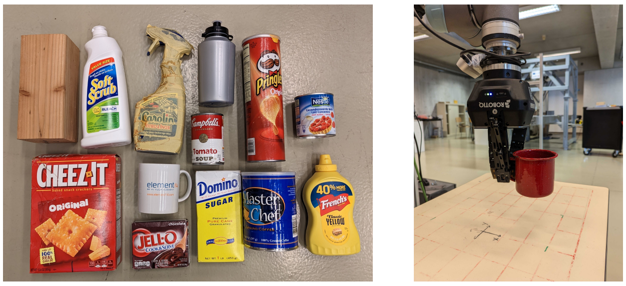


Fig. 5: (left) Object assets used in the real setup. (right) Successful grasp of a mug.

Representing a 3D scene as a parameterized function using neural networks has recently gained substantial interest. Occupancy networks [13] determine whether a point is occupied or not. However, they lack *equivariance* for translations and rotations. To overcome this limitation, translation-equivariant feature planes are derived from convolutional networks, resulting in Convolutional Occupancy Networks [3]. Achieving rotation-equivariance is more challenging, but innovative solutions have recently emerged [14], [15].

Various methods exist for sampling from a distribution defined on a Riemannian manifold. Adapting Markov Chain Monte Carlo methods [16] to Riemannian manifolds allows sampling from a differentiable and tractable likelihood. Our approach eliminates the need for an explicit likelihood. Normalizing flows [17] which target density defined on manifolds can rapidly sample from the posterior distribution. However, it remains unclear how to use their gradients for Riemannian optimization.

Finally, probabilistic approaches for grasping problems typically depend on explicit likelihood functions that model the probability of success or a grasp quality metric related to an observation and a grasp pose [7], [18]–[20]. Closer to our work, a similar study [8] uses simulation-based inference to compute the maximum a posteriori through Riemannian gradient ascent. However, this approach uses a heuristic prior and cannot accommodate multiple objects.

VI. Conclusion

We have shown that Bayesian inference can be effectively applied to robotic grasping in complex and noisy environments. Through innovative enhancements, we have improved the sample efficiency of the inference pipeline. Our approach can manage tasks with escalating complexity and proves valuable for real-world robotic applications. Future research will focus on controlling the complete 6-DoF of the robotic hand.

Acknowledgement

Norman Marlier acknowledges the Belgian Fund for Research training in Industry and Agriculture for its financial support (FRIA grant). Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region.

References

[1] K. Cranmer, J. Brehmer, and G. Louppe, “The frontier of simulation-based inference,” *Proceedings of the National Academy of Sciences*, 2020.

[2] S. Byrne and M. Girolami, “Geodesic monte carlo on embedded manifolds,” *Scandinavian Journal of Statistics*, vol. 40, no. 4, pp. 825–845, 2013.

[3] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger, “Convolutional occupancy networks,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 2020, pp. 523–540.

[4] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” 2015.

[5] J. Hermans, V. Begy, and G. Louppe, “Likelihood-free MCMC with amortized approximate ratio estimators,” in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 4239–4248. [Online]. Available: http://proceedings.mlr.press/v119/hermans20a.html

[6] K. Murphy, C. Esteves, V. Jampani, S. Ramalingam, and A. Makadia, “Implicit-pdf: Non-parametric representation of probability distributions on the rotation manifold,” *arXiv preprint arXiv:2106.05965*, 2021.

[7] M. Breyer, J. J. Chung, L. Ott, S. Roland, and N. Juan, “Volumetric grasping network: Real-time 6 dof grasp detection in clutter,” in *Conference on Robot Learning*, 2020.

[8] N. Marlier, O. Brüls, and G. Louppe, “Simulation-based bayesian inference for robotic grasping,” *arXiv preprint arXiv:2303.05873*, 2023.

[9] A. Ten Pas, M. Gualtieri, K. Saenko, and R. Platt, “Grasp pose detection in point clouds,” *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, 2017.

[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,” in *Robotics Research: The 19th International Symposium ISRR*. Springer, 2022, pp. 890–905.

[11] 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.

[12] X. Yan, J. Hsu, M. Khansari, Y. Bai, A. Pathak, A. Gupta, J. Davidson, and H. Lee, “Learning 6-dof grasping interaction via deep geometry-aware 3d representations,” 2018.

[13] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, “Occupancy networks: Learning 3d reconstruction in function space,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4460–4470.

[14] Y. Chen, B. Fernando, H. Bilen, M. Nießner, and E. Gavves, “3d equivariant graph implicit functions,” in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part III*. Springer, 2022, pp. 485–502.

[15] C. Deng, O. Litany, Y. Duan, A. Poulenard, A. Tagliasacchi, and L. J. Guibas, “Vector neurons: A general framework for so (3)-equivariant networks,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 200–12 209.

[16] C. Liua and J. Zhub, “Geometry in sampling methods: A review on manifold mcmc and particle-based variational inference methods,” *Advancements in Bayesian Methods and Implementations*, vol. 47, p. 239, 2022.

[17] D. J. Rezende, G. Papamakarios, S. Racaniere, M. Albergo, G. Kanwar, P. Shanahan, and K. Cranmer, “Normalizing flows on tori and spheres,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 8083–8092.

[18] Q. Lu, K. Chenna, B. Sundaralingam, and T. Hermans, “Planning multi-fingered grasps as probabilistic inference in a learned deep network,” in *Robotics Research*. Springer, 2020, pp. 455–472.

[19] J. Cai, J. Cen, H. Wang, and M. Y. Wang, “Real-time collision-free grasp pose detection with geometry-aware refinement using high-resolution volume,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1888–1895, 2022.

[20] M. Van der Merwe, Q. Lu, B. Sundaralingam, M. Matak, and T. Hermans, “Learning continuous 3d reconstructions for geometrically aware grasping,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 11 516–11 522.

Appendix

A. Likelihood-free geodesic Monte Carlo

Algorithm 1: Likelihood-free geodesic Hamiltonian Monte Carlo

```
Input: A Manifold $\mathcal{M}$
       Initial parameters $\mathbf{h}_0$
       Prior $p(\mathbf{h} \mid \mathbf{P})$
       Momentum distribution $p(\mathbf{v})$
       Trained classifier $d_\phi(S, \mathbf{h}, \mathbf{P})$
       Observations $S, \mathbf{P}$
Output: Markov chain $\mathbf{h}_{1:T}$

$t \leftarrow 0$
$\mathbf{h}_t \leftarrow \mathbf{h}_0$
for $t < T$ do
    $\mathbf{v}_t \sim p(\mathbf{v})$
    $\mathbf{v}_t \leftarrow \boldsymbol{\pi}_{\mathbf{h}_t}(\mathbf{v}_t)$
    $k \leftarrow 0$
    $\mathbf{v}_k \leftarrow \mathbf{v}_t$
    $\mathbf{h}_k \leftarrow \mathbf{h}_t$
    for $k < L$ do
        $\mathbf{v}_k \leftarrow \mathbf{v}_k + \frac{\epsilon}{2} \nabla_{\mathbf{h}_k} \log r(S \mid \mathbf{h}_k, \mathbf{P})$
        $\mathbf{v}_k \leftarrow \boldsymbol{\pi}_{\mathbf{h}_k}(\mathbf{v}_k)$
        $\mathbf{h}_k \leftarrow \gamma(\epsilon), \gamma(0) = \mathbf{h}_k$
        $\mathbf{v}_k \leftarrow \dot{\gamma}(\epsilon), \dot{\gamma}(0) = \mathbf{v}_k$
        $\mathbf{v}_k \leftarrow \mathbf{v}_k + \frac{\epsilon}{2} \nabla_{\mathbf{h}_k} \log r(S \mid \mathbf{h}_k, \mathbf{P})$
        $\mathbf{v}_k \leftarrow \boldsymbol{\pi}_{\mathbf{h}_k}(\mathbf{v}_k)$
        $k \leftarrow k + 1$
    end
    $\lambda_k \leftarrow \log r(S \mid \mathbf{h}_k, \mathbf{P}) + \log p(\mathbf{h}_k \mid \mathbf{P}) - \frac{1}{2} \mathbf{v}_k^T \mathbf{v}_k$
    $\lambda_t \leftarrow \log r(S \mid \mathbf{h}_t, \mathbf{P}) + \log p(\mathbf{h}_t \mid \mathbf{P}) - \frac{1}{2} \mathbf{v}_t^T \mathbf{v}_t$
    $\rho \leftarrow \min(\exp(\lambda_k - \lambda_t), 1)$
    $\mathbf{h}_{t+1} \leftarrow \begin{cases} \mathbf{h}_k & \text{with a probability } \rho \\ \mathbf{h}_t & \text{with a probability } 1 - \rho \end{cases}$
    $t \leftarrow t + 1$
end
return $\mathbf{h}_{1:T}$
```

B. Sampling the orientation: toy problem

Given a model parameter sample $q_\theta \in \mathbb{S}^d$, the forward generative process is defined as:

$$\nu = q_\theta \tag{6}$$

$$\kappa = 20 \tag{7}$$

$$q_x \sim \exp(\kappa \nu^T q_x) \tag{8}$$

with the prior $p(q_\theta) \triangleq \texttt{SphericalUniform}(d)$. It follows the true posterior $p(q_\theta \mid q_x) \propto \exp(\kappa q_x^T q_\theta)$. We use a MLP of 3 layers with 64 neurons to approximate the likelihood-to-evidence ratio. All the activation functions are ReLU except the last one which is linear. We train the ratio with 1000000 samples with a batch size of 8000 for 50 epochs. For the geodesic HMC, we use 100 chains and 2000 transitions with a burn in of 1000. The integration parameters are $\epsilon = 0.01, L = 20$. The approximate posterior shares the same structure that the true posterior, demonstrating its accuracy (Fig. 6). Moreover, we conduct a quantitative analysis by computing the Maximal Mean Discrepancy (MMD) for $\mathbb{S}^1$ and $\mathbb{S}^3$. We obtain respectively $0.0028 \pm 5.84e^{-6}$ and $0.01 \pm 3e^{-5}$ for an identity kernel between the two geodesic means for 10 different observations q_x.

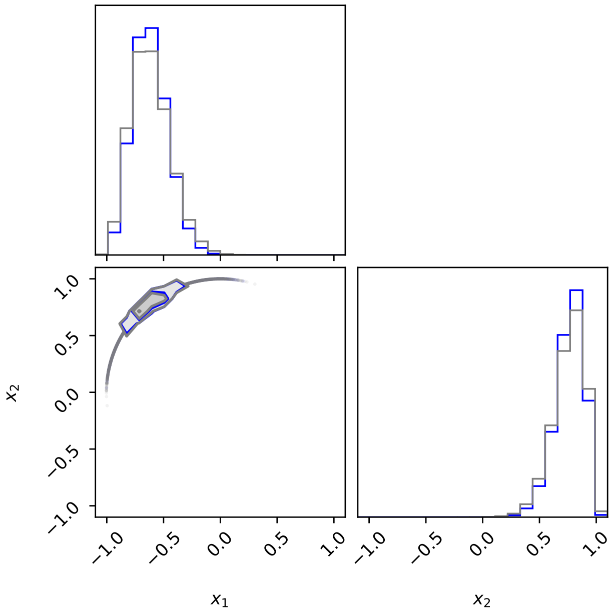


Fig. 6: Posterior for $\mathbb{S}^1$. In blue, the distribution obtained through GMC with the ratio and in grey, the ground truth posterior.

C. Sampling the position: multiple objects scene

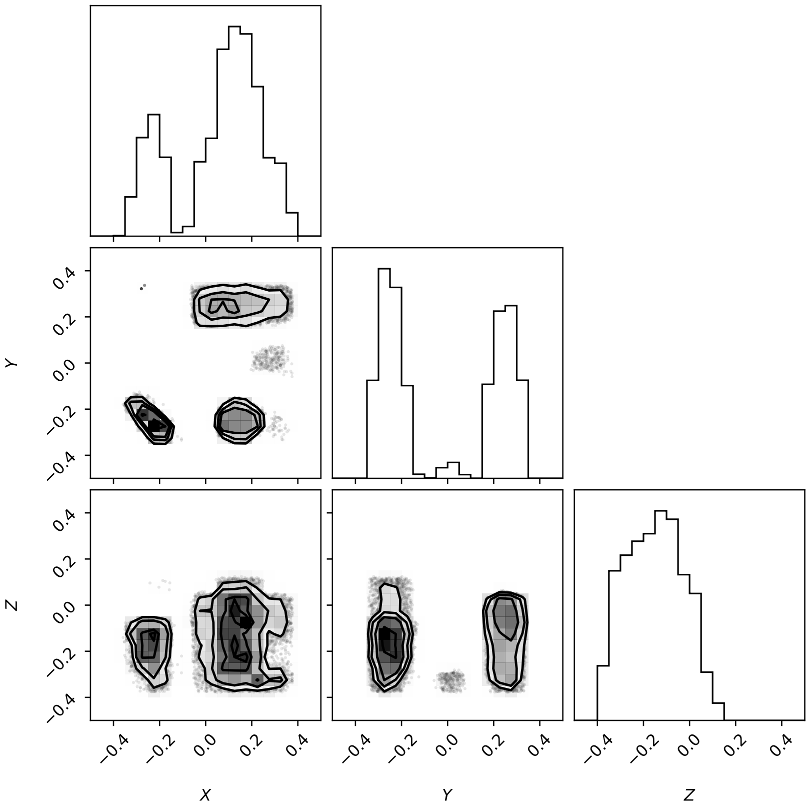


Fig. 7: Approximate posterior $p(\mathbf{x}|o = 1, \mathbf{P})$ of a scene with 5 different objects.

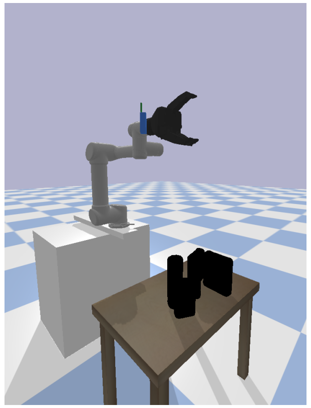
In this experiment, we have a scene containing five objects with varying levels of difficulty, as shown in Fig. 2. We use a convolutional occupancy network [3] that is trained for 120,000 iterations with a batch size of 32 samples. The network has a resolution of 128 and a feature dimension of 32. To sample from the posterior $p(\mathbf{x}|o = 1, \mathbf{P}) \propto p(o = 1|\mathbf{x}, \mathbf{P})p(\mathbf{x})$, as we previously explained, we employed HMC with specific hyper-parameters. These include 100 chains of 5000 transitions and a burn-in of 1000. The integration parameters are $\epsilon = 0.01$, $L = 20$. The chains' initials point are sampled uniformly in the bounding box of the objects. The corner plot in Fig. 7 illustrates the resulting posterior, which aligns with the location and shape of four out of the five objects and the potential grasping point. However, the occupancy of the fifth object cannot be recovered due to either being regarded as noise or having a much smaller density compared to the other objects.

Implicit representation priors meet Riemannian geometry for Bayesian robotic grasping

Norman Marlier* Julien Gustin* Olivier Brüls Gilles Louppe

University of Liège, Belgium

Robotic grasping



We formulate the problem of grasping as the Bayesian inference of the hand configuration $\mathbf{h} := (\mathbf{x}, \mathbf{q})$ that is a posteriori the most likely given a successful grasp $S = 1$, an occupied point o and a point cloud $\mathbf{P}$.

Probabilistic modeling

We solve the grasping problem by computing the maximum a posteriori

$$\mathbf{h}^* = \arg\max_{\mathbf{h}} \; p(\mathbf{h}|S = 1, o = 1, \mathbf{P})$$

From the Bayes rule, the posterior of the hand configuration is

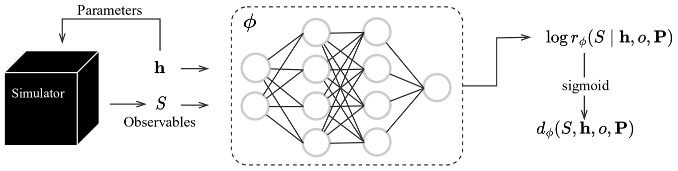
$$p(\mathbf{h} \mid S, o, \mathbf{P}) = \frac{p(S \mid \mathbf{h}, o, \mathbf{P})}{p(S \mid o, \mathbf{P})} p(\mathbf{h} \mid o, \mathbf{P})$$

which can be rewritten as the product of the likelihood-to-evidence ratio r and a scene-dependent prior

$$p(\mathbf{h}|S, o, \mathbf{P}) = r(S \mid \mathbf{h}, o, \mathbf{P}) p(\mathbf{h} \mid o, \mathbf{P}).$$

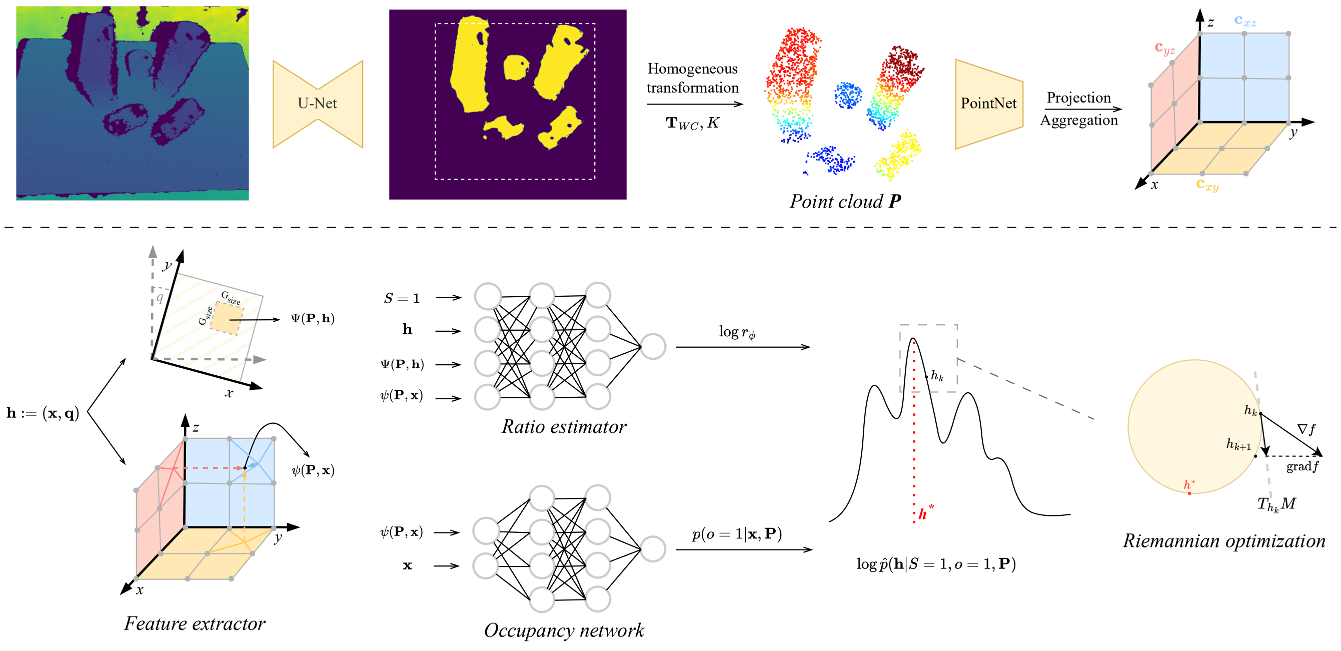
Neural Ratio Estimation

Neural ratio estimation consists in training a classifier d_ϕ to discriminate between samples from the joint density, $p(S, \mathbf{h} \mid o, \mathbf{P})$, and the marginal densities, $p(S \mid o, \mathbf{P}) p(\mathbf{h} \mid, o, \mathbf{P})$.


Parameters
Simulator
$\mathbf{h}$
S
Observables
ϕ
$\log r_\phi(S \mid \mathbf{h}, o, \mathbf{P})$
sigmoid
$d_\phi(S, \mathbf{h}, o, \mathbf{P})$

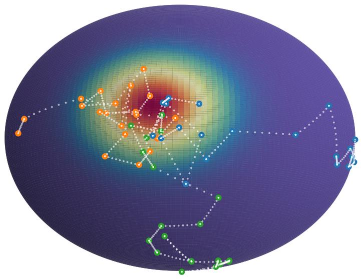
Classifer

Grasp inference pipeline

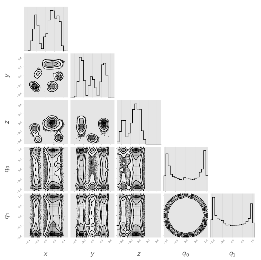


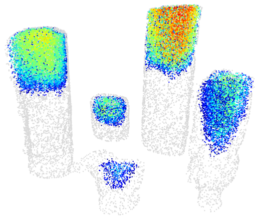
Manifold sampling

By integrating likelihood-free Hamiltonian Monte Carlo and geodesic Monte Carlo, we are able to sample from the posterior density defined on a smooth manifold using a closed-form geodesic.



Approximate posterior





Experimental results

Method	Success rate (%)	% cleared
Simulation results		
GPD	73.7	72.8
VGN ($\varepsilon = 0.95$)	91.5	79
VGN ($\varepsilon = 0.9$)	87.6	80.4
VGN ($\varepsilon = 0.85$)	80.4	79.9
Ours	91.1	77
Real-world results		
Ours	95.6	88

Take-home messages

- Our implicit prior captures relevant 3D information about the scene, enabling full Bayesian inference for complex tasks.
- Our approach directly models variables on their respective manifolds, effectively handling intrinsic constraints.
- Our approach overcomes the simulation-real-world discrepancy without performance degradation.

IEEE International Conference on Robotics and Automation 2023 Geometric Representations Workshop

norman.marlier@uliege.be

Appendix C

Multiresolution IsoSurface Extraction

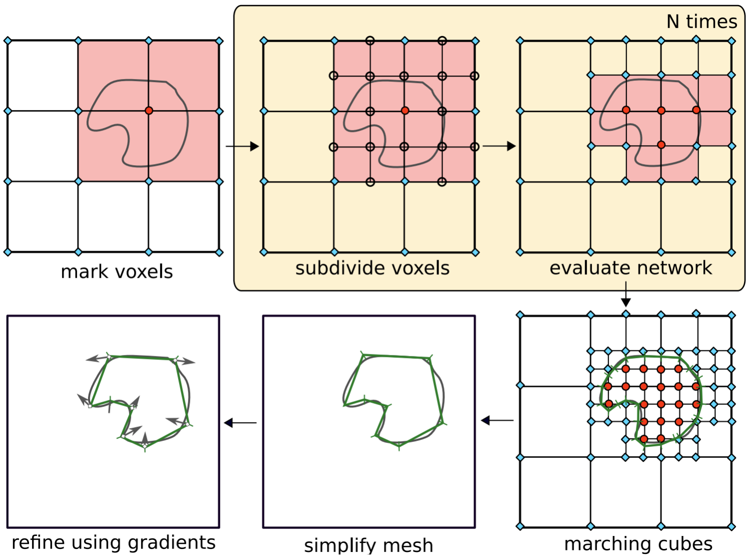


Figure C.1. The Multiresolution IsoSurface Extraction (MISE) algorithm [14] marks points as either occupied (red circles) or unoccupied (cyan diamonds). The active voxels are subdivided into smaller sub-voxels until the desired resolution is reached. The resulting structure is then processed using the marching cubes algorithm [53], along with refinement techniques like mesh simplification and gradient refinement.

In their work, Meschder et al. [14] introduced the Multiresolution IsoSurface Extraction (MISE) algorithm for extracting meshes from an implicit function f_θ, where $f_\theta : \mathbb{R}^3 \times \mathcal{P} \to [0, 1]$. The MISE algorithm begins by discretizing the three-dimensional space into a resolution of 32^3, following the recommendation in the original paper [14]. Each point $\mathbf{x}$ in $\mathbb{R}^3$ is then classified as occupied or unoccupied based on the value of $f_\theta(\mathbf{x}, \mathbf{P})$ and a threshold value τ. Next, voxelization is performed, identifying voxels that contain at least two adjacent grid points with different occupancy values. These voxels are marked as active. The process continues by subdividing the active voxels into eight smaller voxels, and this subdivision is repeated until the desired resolution is achieved. Finally, the marching cubes algorithm [53] is employed to extract a watertight mesh from the active voxels.

Appendix D

Additional figures

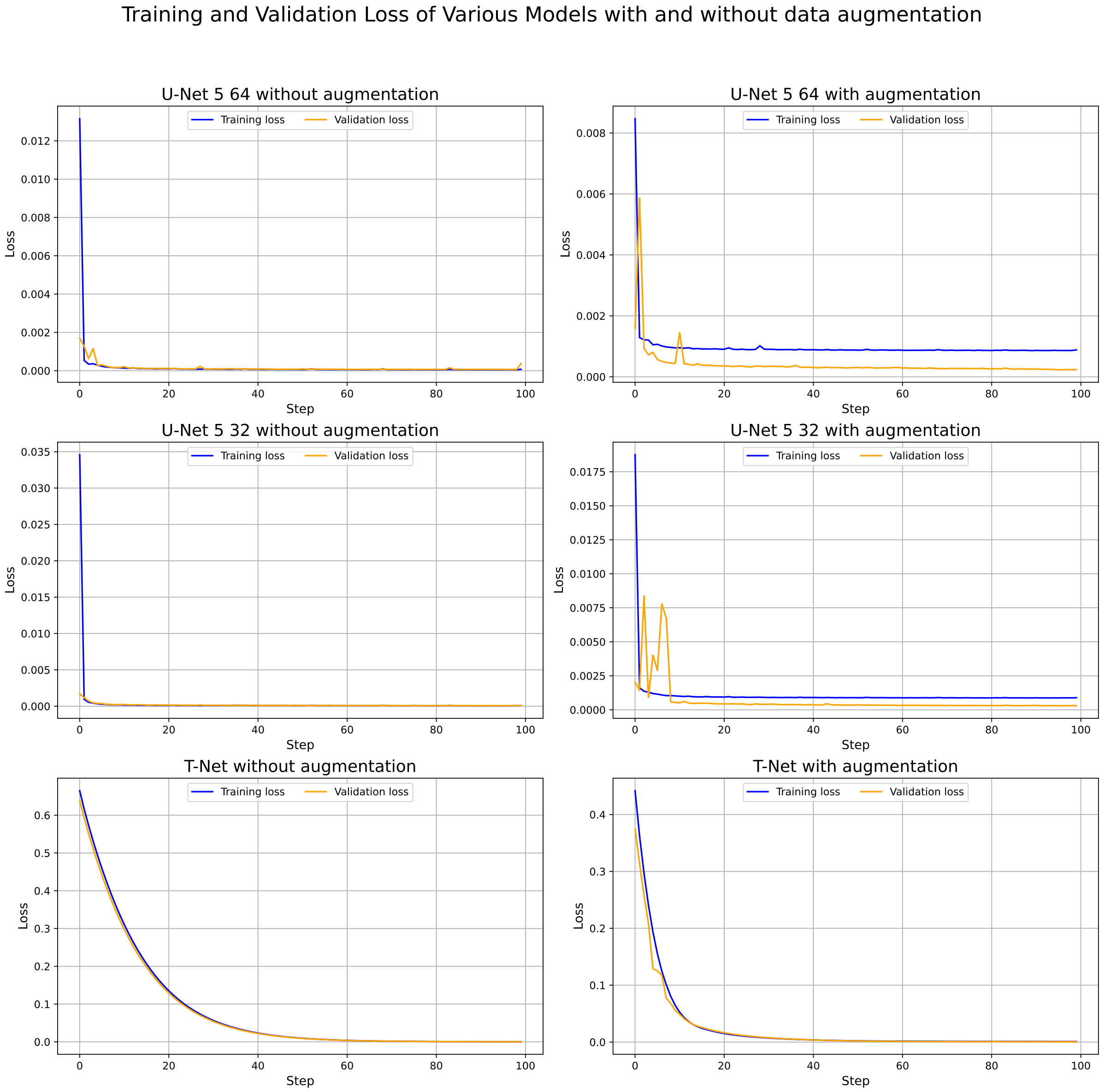


Figure D.1. Training and validation losses for different segmentation models with and without data augmentation. The loss used is the binary cross-entropy.

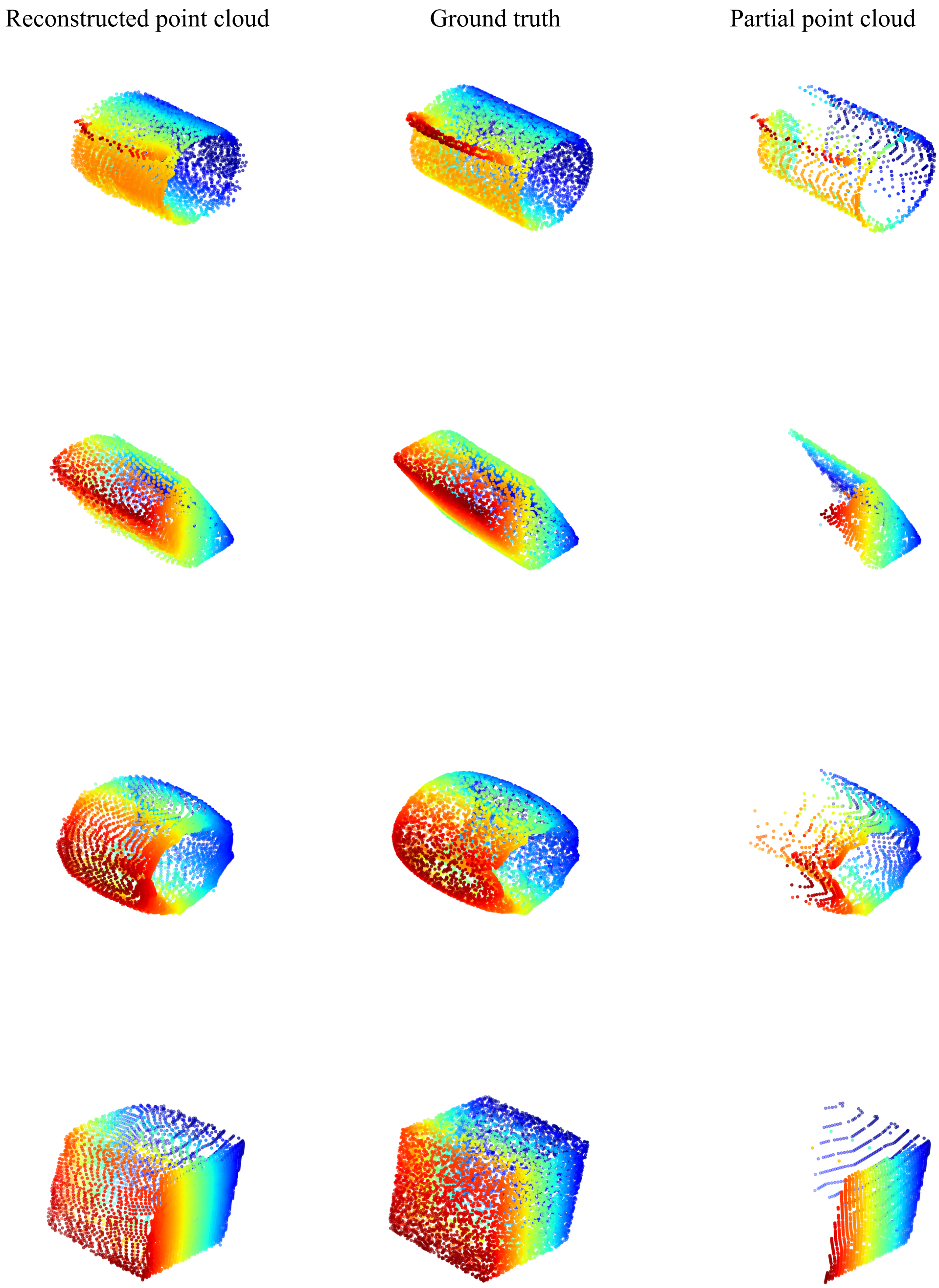


Figure D.2. Examples of point clouds reconstructed with PoinTr [43].

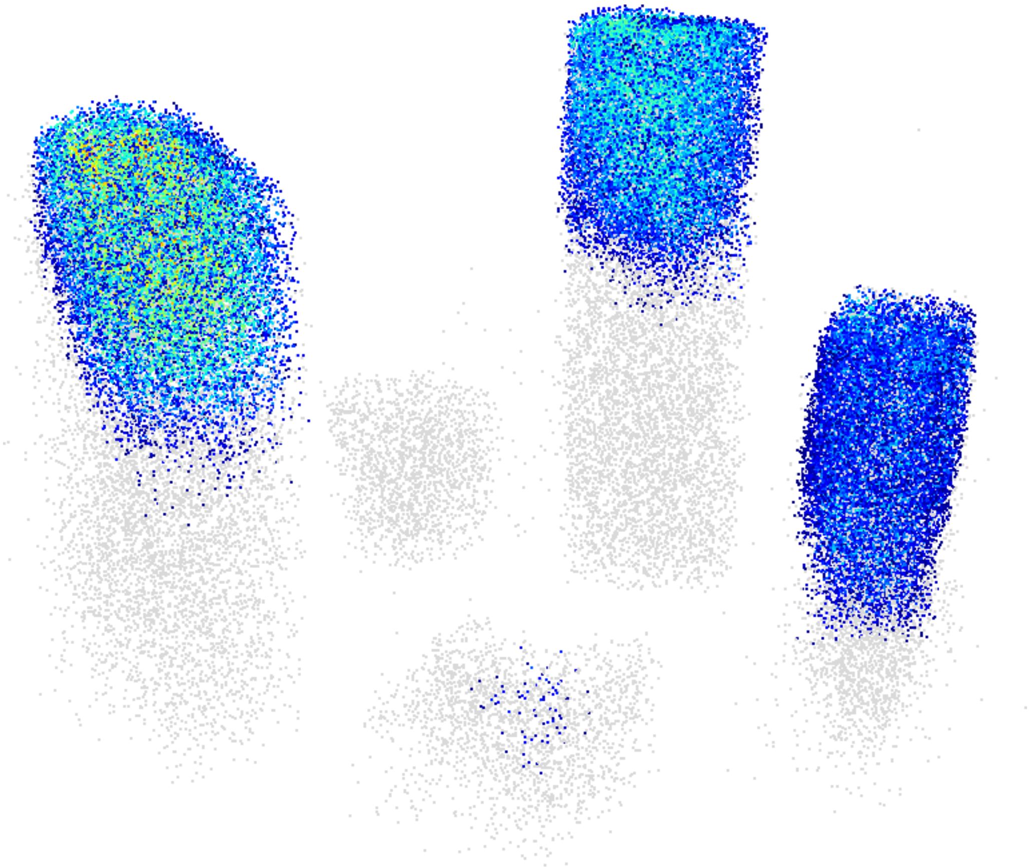


Figure D.3. 3D visualization of the prior and posterior distributions of potential grasping points. Grasping points sampled from $\hat{p}(\mathbf{h} \mid S = 1, o = 1, \mathbf{P})$ are colored with a gradient from blue to red, indicating their relative posterior values. Grey points represent grasping points sampled from the prior distribution $p(\mathbf{x} \mid o = 1, \mathbf{P})$. The ratio used for the posterior estimation is an ensemble of 5 *Cropped & Rotated features* model.