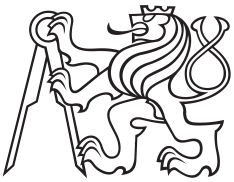
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Visuo-Haptic Uncertainty-Driven Object Shape Completion

Bc. Lukáš Rustler

Supervisor: Mgr. Matěj Hoffmann, Ph.D.
Supervisor–specialist: Jens Lundell
Field of study: Cybernetics and Robotics
January 2022

Acknowledgements

I want to thank everyone who helped me with this work. Firstly, to Matěj Hoffmann, who gave me the opportunity to be part of this project, supervised the thesis and supported me during the whole time—from technical to formal aspects of this work. I also want to thank Jens Lundell, whose knowledge and experience helped me overcome many problems, and the results would not be possible without him. My gratitude also belongs to Jan Behrens, who provided the base of a simulation environment and gave me an introduction to its functionality. I also thank Martin Šrámek, who helped me create a model of an additional finger for the robot.

I would also like to thank my family for continuous support through the years and my friends for their encouragement. And finally, I thank CTU in Prague for being such a good *alma mater*.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských prací.

V Praze dne 04. ledna 2022

.....
Lukáš Rustler

Abstract

Shape completion is the task of reconstruction of an object using incomplete information. Recently, advancements in single-view visual-only shape completion have been made. However, the objects can self-occlude or be adversarial for cameras. In this work, we propose a novel active visuo-haptic shape completion method (Act-VH). The method combines a point cloud from a stationary camera and data from haptic exploration with a robotic manipulator. A state-of-the-art implicit surface neural network is used to complete the shapes. The location to explore is actively computed from the reconstruction uncertainty. We evaluate the method both in simulation, for which we present a unique environment, and in the real world. In both worlds, real-time visualizations of current progress are provided. The performance is compared to baseline methods in terms of reconstruction accuracy and exploration efficiency. The results show that Act-VH outperforms all baselines in both criteria. In addition, we perform grasp experiments on real-world objects, reaching a significantly higher grasp success rate than the baseline. Finally, Act-VH is compared to an active visual-only method that can obtain information from occluded parts. Together, this work opens up the door for using active visuo-haptic shape completion in real-world robotic tasks.

Keywords: Perception for Grasping and Manipulation, RGB-D Perception, Deep Learning for Visual Perception, Shape Completion, Shape Reconstruction

Supervisor: Mgr. Matěj Hoffmann, Ph.D.

Abstrakt

Modelováni tvaru objektu je úloha zabývající se rekonstrukcí objektu z neúplné informace. V nedávné době pokročila rekonstrukce z neúplné, pouze vizuální informace. Nicméně předměty se můžou překrývat nebo být "nepřátelské" pro kameru. Jako řešení navrhujeme naši metodu nazvanou aktivní visuo-haptické modelování tvaru (Act-VH). Tato metoda kombinuje mrak bodů ze stacionární kamery a data z haptické explorace pomocí robotického manipulátoru. Moderní neuronová sít na principu implicitních povrchů je použita pro doplňování tvarů. Místo pro exploraci je aktivně počítáno z nejistoty rekonstrukce. Metoda je posouzena v simulaci, pro kterou představujeme unikátní simulační prostředí, a v reálném světě. V obou světech jsou dostupné vizualizace aktuálního pokroku v reálném čase. Přesnost rekonstrukce a efektivita explorace je porovnána s existujícími metodami. Výsledky ukazují, že Act-VH ostatní metody překonává. Kromě toho provádíme sérii uchopovacích experimentů, ve kterých jsme dosáhli vyšší úspěšnosti než existující metody. Nakonec je naše metoda porovnána s aktivní, pouze vizuální metodou, která je schopná získat informaci i ze zakrytých části. Celkově je tato práce vstupním bodem pro použití visuo-haptického doplňování tvarů v robotických úlohách v reálném světě.

Klíčová slova: Vnímání pro uchopování a manipulaci, Vnímání pomocí RGB-D, Hluboké učení pro vizuální vnímání, Modelování tvaru objektů, Rekonstrukce objektů

Překlad názvu: Visuo-haptické modelování tvaru objektu řízené nejistotou

Contents

Chapter 1

Introduction

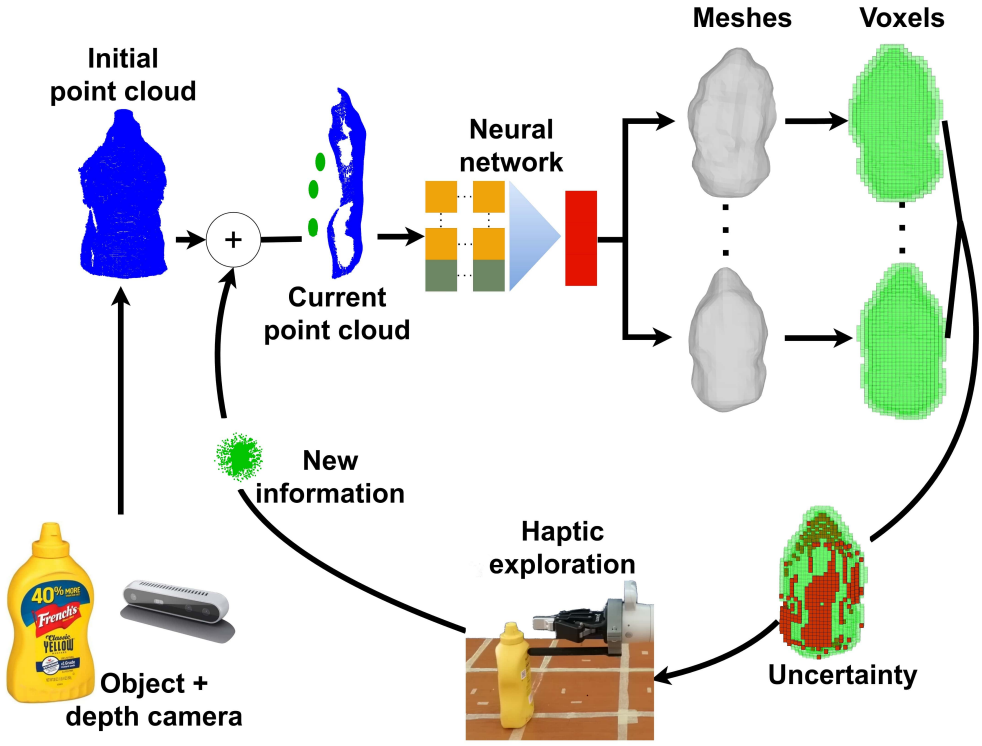


Figure 1.1: Schematic operation of our active visuo-haptic shape completion method (Act-VH). An initial depth image is used to construct an initial point cloud. Neural network (IGR) generates several possible shape reconstructions, whose discrepancy is used to form a single object reconstruction with uncertainty. The voxel with the highest uncertainty is selected for haptic exploration, and a new points are added to the object representation. This process is repeated, further refining the shape reconstruction. From Rustler *et al.* [1].

Shape completion is the task of reconstruction of an object using incomplete information. The output object is usually a triangular mesh or a voxel grid. Shape completion is an active research problem with potential in a wide range of applications and influences robotic tasks, such as grasping. Nowadays, the information is often acquired by visual sensors, which provide RGB images with or without depth data. However, using visual-only data is complicated. For example, when only a stationary camera is available, the object self-occludes, *i.e.*, information from only one side of the object is available. One can collect more

samples, but it requires more cameras, a movable camera, or a turntable under the object. Additionally, even with current cameras, a lot of noise is present when capturing adversarial objects such as transparent or dark items. A straightforward solution is to explore the unseen parts by touch and reconstruct the shape with visuo-haptic data. However, current visuo-haptic methods use a heuristic to choose where to touch [2] or require an impractical number of touches with enormous computational time [3, 4].

We provide a solution to these issues with our Act-VH method. Act-VH is an active visuo-haptic shape completion method, which iteratively refines the output shape. The method is depicted in Fig. 1.1. Firstly, depth information of the scene is used to construct an initial point cloud. Then, a state-of-the-art deep implicit surface neural network (Implicit Geometric Regularization for Learning Shapes (IGR) [5]) is used to generate a set of possible shapes, which is utilized to obtain a single shape with uncertainty. The segment of the object with the highest uncertainty is selected for haptic exploration, providing new information which is merged to the point cloud. The entire process is repeated, further refining the reconstruction.

A manuscript based on the visuo-haptic shape completion part of this work is currently under review for *The IEEE International Conference on Robotics and Automation (ICRA) 2022 with Robotics and Automation Letters (RA-L) option* [1]. Figures that were used in this manuscript as well as here include a reference to [1] in their captions. An accompanying video is here: `https://youtu.be/Ft1PUYRNFHw`. The simulation environment and the data from experiments pertaining to [1] are available at `https://github.com/ctu-vras/visuo-haptic-shape-completion`. Additional internal code is available in the attachment of this work. Data collected during the experiments (point clouds and ROSbags) can be downloaded from `https://drive.google.com/drive/folders/1Du8hVDbsFYqEvb-hwS6ISpnaE4RGJW8c?usp=sharing`.

Chapter 2

Related Work

Data for shape completion can be obtained from several inputs. The most obvious is purely visual input from RGB or depth sensors. In combination with robotic arms and hands, the data can be also collected by haptic exploration. In many recent works, these two inputs are combined. The data can be collected only once or gathered actively—usually known as active perception [6].

2.1 Visual-only Shape Completion

Visual-only approach to shape completion is the most common one. Usually sensors with depth perception are used. This approach is beneficial because of the global nature of the information gained—a single sample contains a lot of data. The first visual approaches were geometry- or template-based. The geometry ones assume the objects are symmetric and complete the reconstruction by mirroring it through its axis of symmetry, as done by Bohg *et al.* [7]. Other work from Schnabel *et al.* [8] uses primitive shapes detected in the input point cloud. Template-based methods just search in a database of known objects to find the most similar one. Example of template-based method was proposed by Pauly *et al.* [9]. It is clear that these approaches will not work in the general case, *e.g.*, the geometry-based method will perform poorly for non-symmetric objects and the template-based will fail if there is incorrect or no match in the database.

With progress in Machine Learning (ML), approaches based on ML were proposed [2, 5, 10–15], using mainly Gaussian Processes (GPs). The early approach by Li *et al.* trained a Gaussian Process Implicit Surface (GPIS) to reconstruct objects [10]. The problem with GPIS is in its poor scaling over a lot of points in the input point cloud. Thus, the input needs to be down-sampled which results in loss of information and details of the reconstruction. More recent works utilize Deep Learning (DL) methods, mainly Convolutional Neural Networks (CNNs) [2, 11–13]. Using CNN, the objects are represented as voxel grids. Using voxel grids is useful, for example, to represent probabilistic attributes of the reconstruction. However, to preserve fine details, the grids would have to be sampled with very high resolution. And it is, even with modern computers, very computationally demanding because the requirements for memory grow cubically with the resolution.

However, in the world of shape reconstruction, new approaches based on implicit surfaces were proposed [5, 14, 15]. These methods work directly with point clouds (with different needs of point cloud preprocessing), returning triangle meshes. The methods produce reconstructions with higher quality, while preserving better efficiency than voxel grid

methods. They are not originally meant for shape completion from incomplete input, but can be also utilized that way.

Even though shape completion results of the mentioned methods are impressive, it will probably never reach the quality of reconstruction created from a full point cloud. However, to capture the complete point cloud, the camera or the object must be moveable. It can be done by choosing the next-best-view [16–18], using a turntable under the objects [19–22] or using a coordinate measuring machine with a laser scanner [23].

2.2 Haptic-Only Shape Completion

Current robots are already capable of haptic exploration and can be also used for shape completion. With modern force and tactile sensors, haptic exploration can be more accurate than the visual methods. Moreover, the robots are able to explore all parts of the object, even from behind. Most recent haptic-only approaches use ML-based methods similar to the ones used in visual-only approaches. Ottenhaus *et al.* [24] proposed a method based on implicit shape potentials, Yi *et al.* [25] on GPs, Driess *et al.* [26] on GPIS's and Dragiev *et al.* [27] on Gaussian Process Implicit Shape Potentials (GPISPs). The advantage of Gaussian-based methods is the ability to compute uncertainty and actively gather new information to reduce it. Even though touch is usually more precise than visual sensors, it also captures more local information—explores smaller regions. Consequently, haptic-only completion requires a lot of touches (tens to hundreds), which is both impractical and time-demanding.

2.3 Visuo-Haptic Shape Completion

Combining the previous approaches could theoretically preserve the advantages and address the limitations at the same time. Again, methods based on GPIS's [28, 29] and GPs [4] are proposed. These methods require dense information all around the object. With a fixed camera capturing only one side of the object, a lot of surface remains to be explored by touch. CNN-based methods [3, 30] can be used with fewer touches, but suffer from low resolution of the reconstruction. Smith *et al.* proposed approaches [31, 32] based on Graph Neural Network (GNN). Reconstructions by these methods have a higher resolution, but are nonsmooth and, for now, evaluated only in a simulation.

An important part of haptic exploration is the decision where to touch. One can touch the object randomly as done by Smith *et al.* [31], or always select a position opposite of the camera (from "behind") as Watkins-Vall *et al.* [30]. These are, however, not so effective. Uncertainty from Gaussian distribution can be used to explore more effectively [3, 4, 28, 29], or it can be learned where to touch as in Smith *et al.* recent work [32].

The main contributions of the work presented in this thesis are: (i) a novel active visuo-haptic shape completion method; (ii) a visuo-haptic simulation environment; (iii) real–time visualizations suitable for benchmarking; (iv) an empirical evaluation of the proposed method against state of the art, presenting, both in simulation and on real hardware, improvements in terms of reconstruction accuracy and grasp success rates.

Chapter 3

Shape Completion Methods

This chapter gives an overview of state-of-the-art methods for shape reconstruction. Our experiments to determine the most suitable one for this work are presented. Examples of two groups of shape completion approaches are given, together with experimental results and comparison.

3.1 Neural Network Types

Shape completion method is an essential part of the pipeline. We tested and compared multiple state-of-the-art networks based on two principles: (i) voxel-based; and (ii) implicit surface based.

For each network, a new dataset was created from the same training samples—in a format needed by the given network. The tested networks are described below, and a table with the properties of each network can be found in Section 3.2.

3.1.1 Voxel-based Networks

The network introduced by Lundell *et al.* [13] was selected as a representative of this type of networks. The authors have already proved that the network can be used for shape completion from an incomplete point cloud. However, as our experiments showed, the method is impractical for iterative shape refining. The network is trained from multiple incomplete views of the same object. To allow incorporation of tactile information, the network would have to be trained evenly with simulated tactile information all around the objects (a similar thing was done in [30]), which is unfeasible in the general case.

The network utilizes a voxel grid as input. The main disadvantage of this approach is the cubically growing computation and memory requirements which result in low object resolution. As such, fine object details are not preserved, which is important when, for instance, sampling grasps.

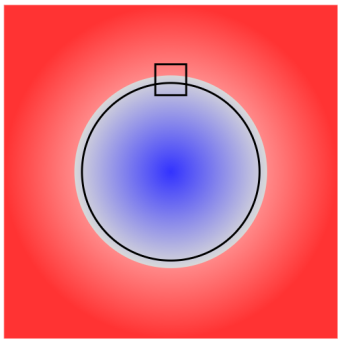
3.1.2 Implicit surface networks

These networks, in contrast to voxel-based methods, can reconstruct visuo-haptic data without explicit training. Firstly, we evaluated the DeepSDF [14] network, which learns

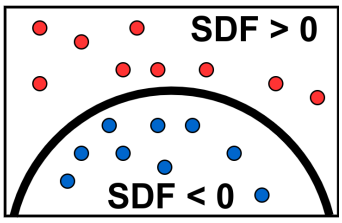
the Signed Distance Function (SDF) defined as:

$$SDF(\mathbf{x}) = s, \tag{3.1}$$

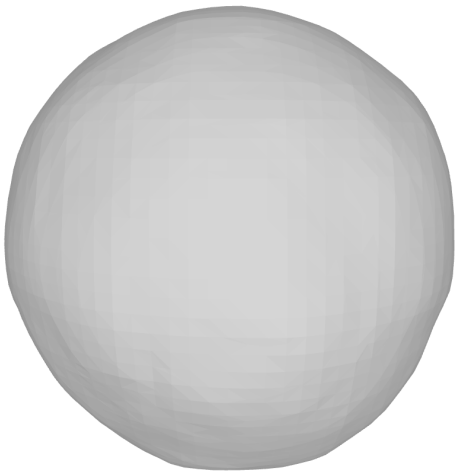
where $\mathbf{x} \in \mathbb{R}^3$ is a three-dimensional position of a point in a point cloud and $s \in \mathbb{R}$ is a signed distance to the surface. The signed distance is used to determine which points are "inside" (negative) and "outside" (positive) of the surface. The desired triangle mesh can be obtained with Marching Cubes algorithm [33] from an isosurface of $SDF(\cdot) = 0$, *i.e.*, from all points with zero distance to the surface. Implicit surface principle shown on a ball can be seen in Fig. 3.1.



(a) : 2D depiction of the signed distance field.



(b) : Zoomed cross-section with implicit surface SDF = 0 (black semicircle), red points "outside" (SDF > 0) and blue point "inside" (SDF < 0).

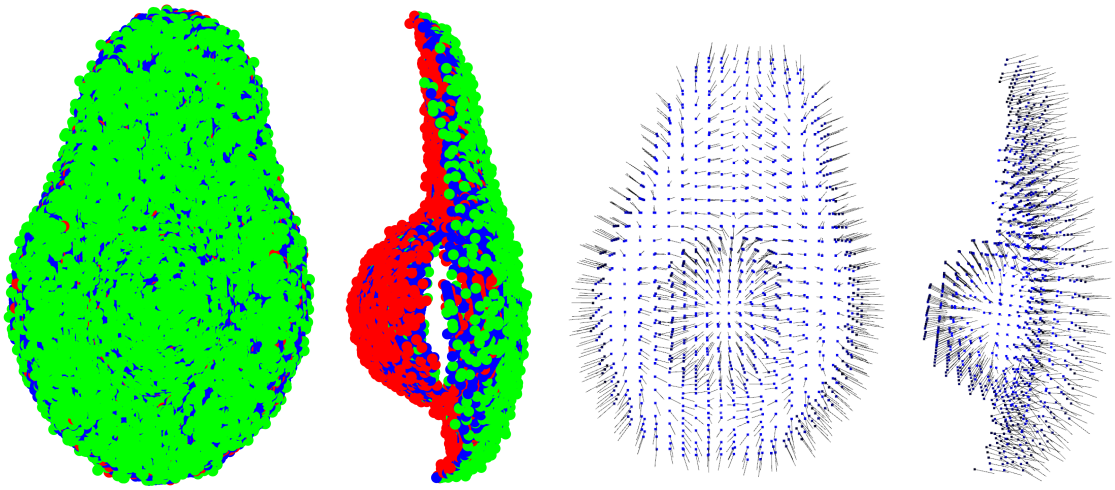


(c) : Triangle mesh reconstructed from implicit surface SDF = 0.

Figure 3.1: 2D signed distance field (a) with a cross-section showing "outside" and "inside" points (b). Triangle mesh reconstructed from isosurface SDF = 0. Inspired by Park *et al.* [14].

The principle of SDF allows to train the network with the whole point cloud and then use just part of it to reconstruct the mesh, which enables integration of tactile information. However, DeepSDF is heavily dependent on preprocessing of the input. The network expects, for both training and prediction, a point cloud with around 500 000 points randomly sampled with *Normal distribution* around the surface. Each sample consists of a point location in 3D space and a signed distance to the surface. See Fig. 3.2a for an example.

Computation of signed distances to the surface is easily achievable when sampling the whole mesh for training but more problematic when creating the input from incomplete point cloud captured with a camera, as it is not clear what is "inside" and "outside". For this reason, we started to experiment with IGR [5]. This work uses the same network structure as DeepSDF [14] but works with raw point clouds—only 3D position and 3D normal for each point are needed. See Fig. 3.2 for comparison with DeepSDF.



(a) : Preprocessed point cloud for DeepSDF. Only points near the surface (blue) are shown, with positive (green) and negative (red) distance to the surface.

(b) : Preprocessed point cloud for IGR with the points (blue) and normals (black).

Figure 3.2: Example of point clouds created for neural networks. Front and side view of incomplete objects.

The network is a Multi-Layer Perceptron (MLP) modeled as a function $f(\mathbf{x};\boldsymbol{\theta})$. The parameters $\boldsymbol{\theta}$ are trained such that f is approximately the SDF to a plausible surface $\mathcal{M}$. The surface $\mathcal{M}$ is defined by the point cloud $\mathcal{X} = \{\mathbf{x}_c\}_{c\in C} \subset \mathbb{R}^3$ and, optionally, a set of normals $\mathcal{N} = \{\mathbf{n}_c\}_{c\in C} \subset \mathbb{R}^3$, where $c \in C$ is the number of points in the point cloud. The loss function of the network is then defined as

$$\ell_{rec}(\boldsymbol{\theta}) = \ell_{\mathcal{X}}(\boldsymbol{\theta}) + \lambda\, \mathbb{E}_{\mathbf{x}} \left(\|\nabla_{\mathbf{x}} f(\mathbf{x};\boldsymbol{\theta})\| - 1\right)^2, \tag{3.2}$$

where λ is parameter and $\lambda > 0$, and

$$\ell_{\mathcal{X}}(\boldsymbol{\theta}) = \frac{1}{|C|} \sum_{c\in C} \left(|f(\mathbf{x}_c;\boldsymbol{\theta})| + \tau \,\|\nabla_{\mathbf{x}} f(\mathbf{x}_c;\boldsymbol{\theta}) - \mathbf{n}_c\|\right), \tag{3.3}$$

where $\tau = 1$ when normals are available.

The first term in Eq. (3.2) forces $\nabla_{\mathbf{x}} f$ to be close to the normals and f to vanish on $\mathcal{X}$, *i.e.*, to make $f = 0$ on $\mathcal{X}$. The second term is called *Eikonal term* and is used because solution f (in the sense of [34]) to the *Eikonal equation*

$$\|\nabla_{\mathbf{x}} f(\mathbf{x};\boldsymbol{\theta})\| = 1, \tag{3.4}$$

will be a SDF and also a global minimum of the loss in Eq. (3.2).

Even though the principle is valid for learning a single shape, using auto-decoder setup from Park *et al.* [14] (see Section 3.1.2), the network can be used for learning multiple shapes. One can introduce the so-called latent vector $\mathbf{z_j} \in \mathbb{R}^d$, where d is usually 128 or 256, and $j \in J$ corresponds to training samples. The function of MLP then changes to

$f(\mathbf{x}; \boldsymbol{\theta}, \mathbf{z}_j)$. Using latent codes also allows us to train the network from point clouds of the whole objects and then supply only an incomplete point cloud to the prediction phase. Using a partial point cloud will change the problem to finding a latent code $\hat{\mathbf{z}}_i$, where $i \in I$ are test samples, which can be achieved with gradient descent optimization. The optimization itself is initialized with a random latent code $\hat{\mathbf{z}}_i, 0$ and fine-tuned with

$$\hat{\mathbf{z}}_{i,t} = \hat{\mathbf{z}}_{i,t-1} - \alpha \nabla_{\hat{\mathbf{z}}_{i,t-1}} \ell(\boldsymbol{\theta}, \hat{\mathbf{z}}_{i,t-1}), \tag{3.5}$$

where α is the step-size and $\nabla_{\hat{\mathbf{z}}_{i,t-1}}$ is the gradient of the following loss function

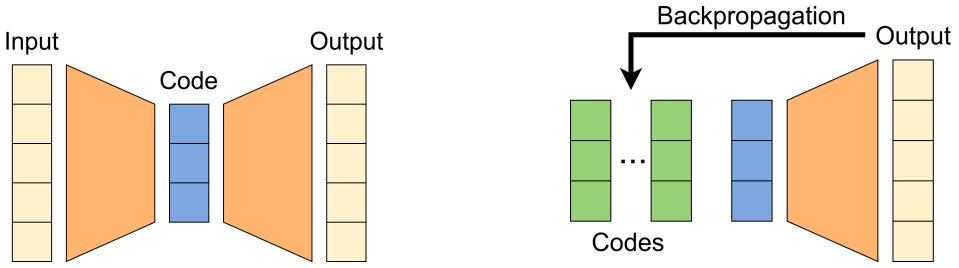
$$\ell(\boldsymbol{\theta}, \hat{\mathbf{z}}_{i,t-1}) = \ell_{rec}(\boldsymbol{\theta}, \hat{\mathbf{z}}_{i,t-1}) + \gamma \left\| \mathbf{z}_{i,t-1} \right\|, \tag{3.6}$$

where $\gamma = 0.01$, and $\ell_{rec}(\boldsymbol{\theta}, \hat{\mathbf{z}}_{i,t-1})$ is the loss in Eq. (3.2).

The disadvantage of this method is a longer computation time resulting from the gradient optimization.

Auto-decoder Architecture

Some other networks (including the one by Lundell *et al.* [13]) use Auto-encoder architecture, which produces the latent vector during its operation from input data points. The Auto-decoder architecture used in [14] accepts randomly generated latent vectors as input together with data points—one latent vector for each data point. The vectors, together with decoder weights, are then optimized using backpropagation—during interference, decoder weights are fixed, and an optimal latent vector is estimated. This solution is more compact and omits the necessity for training the encoder, which is not used in the prediction phase. Schematics of both architectures can be seen in Fig. 3.3.



(a) : Auto-encoder. **(b) :** Auto-decoder.

Figure 3.3: Auto-encoder and Auto-decoder architectures. From Park *et al.* [14].

3.2 Networks Comparison

Comparison for the three tested networks can be found in Table 3.1. Even though the information in the table are similar at first sight, our experiments showed that the difference in input makes the most significant difference. This highly supports IGR, which uses a point cloud with normals, over DeepSDF, which requires sampled points with signed distances to the surface. IGR also outperforms the network from Lundell *et al.* because of the natural integration of tactile information. For these reasons, we decided to use IGR for this work.

Network	Lundell *et al.* [13]	DeepSDF [14]	IGR [5]
Type	Voxel-based	Implicit surface	Implicit surface
Input	Voxel grid	Samples with normal distribution around the surface with signed distances to it	Point cloud with normals
Architecture	Auto-encoder	Auto-decoder	Auto-decoder
Allows integration of tactile information	No	Yes	Yes
Train samples	Incomplete objects	Whole objects	Whole objects
Level of details	Low	High	High
Computation time	Under 1 second	Seconds	Seconds

Table 3.1: Properties of the tested networks. The run times of DeepSDF and IGR strongly depend on the hardware.

3.3 Implementation Details

Our implementation is based on a publicly available GitHub [35] project from the authors of IGR [5]. The original implementation was improved to better fit our problem, mainly by adding a custom dataset preprocessing and the ability to provide reconstructions usable for uncertainty computation (see Section 5.2). We named our implementation as Shape Completing IGR (CIGR).

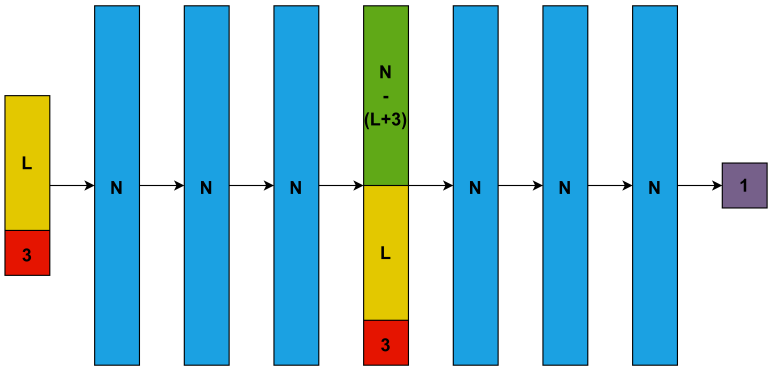


Figure 3.4: Structure of the network. All layers are fully connected and all connections, except the last one, are followed by Softplus activation function. L stands for latent size (256 in our implementation) and N stands for number of neurons (512 in our implementation).

The network itself was run in Python3 with the use of PyTorch 1.0.0. We kept the structure of the network to be the same as in both IGR and DeepSDF (8 fully connected layers with 512 neurons, with a skip connection in the 4th layer; see Fig. 3.4). We tested some other parameters (on NVIDIA GeForce GTX 1080 Ti), eventually resulting in:

- batch size: 8;
  - note that this value is mostly influenced by the available memory on the graphics card;
- latent size: 256;

- learning rate: 0.005, decreasing by half every 500 iterations;
- epochs: 3500.

3.4 Datasets

We used two publicly available mesh datasets, the *YCB* dataset [36] (available also in physical version and used primarily for real interference) and the *Grasp Database* dataset [37] (used primarily for neural network training). Both of the datasets include triangle meshes with mainly household objects—examples of the objects can be seen in Fig. 4.2.

The meshes were used as ground truth for evaluation. However, the neural network takes point cloud as input, so we created a new dataset from the meshes. The procedure of point cloud creation is depicted in Algorithm 1. The procedure applies for both training and validation datasets, with the difference in used point clouds—whole for training and incomplete for validation. Each mesh needs to be normalized (lines 5-6) for correct training. Then each mesh is sampled with 100000 points (lines 7). The number of points was chosen experimentally to be high enough to densely cover the whole surface of the mesh but, on the other hand, not too high to make the training unfeasibly long. Each of the sampled meshes is then rotated to 16 different views (line 9). The rotations are always about x-, y- or z-axis, with rotation angles being multiplies of 90°. The reasoning behind selecting exactly these rotations is that the training time is growing with the number of training samples, so we wanted to limit it by selecting only an essential subset of different views. And eventually, the resulting point clouds are saved to separate files.

Algorithm 1 Dataset creation for CIGR

1: **Inputs:** set of triangle meshes $\mathbf{M}$, set of rotation matrices $\mathbf{R}$;
2: **Output:** set of point clouds $\mathbf{P}$;
3: $\mathbf{P} \leftarrow$ empty set of point clouds;
4: **for each** $m \in \mathbf{M}$ **do**
5: $m \leftarrow$ center m at the origin;
6: $m \leftarrow$ scale m so that its longest dimension is between -1 and 1;
7: $\mathbf{s} \leftarrow$ sample m with 100000 points evenly distributed over its surface;
8: **for each** $\mathbf{r} \in \mathbf{R}$ **do**
9: $\mathbf{p} \leftarrow \mathbf{r} \cdot \mathbf{s}$; ▷ Rotate $\mathbf{s}$ with $\mathbf{r}$
10: $\mathbf{P} \leftarrow \mathbf{P} + \mathbf{p}$; ▷ Add $\mathbf{p}$ to $\mathbf{P}$
11: **end for**
12: **end for**
13: **return** $\mathbf{P}$;

Chapter 4

Materials and Methods

This chapter gives an overview of the software and hardware used in this work. Basic principles of the pipeline are given, and the reader is introduced to a detailed description of the work done.

4.1 Software and Implementation

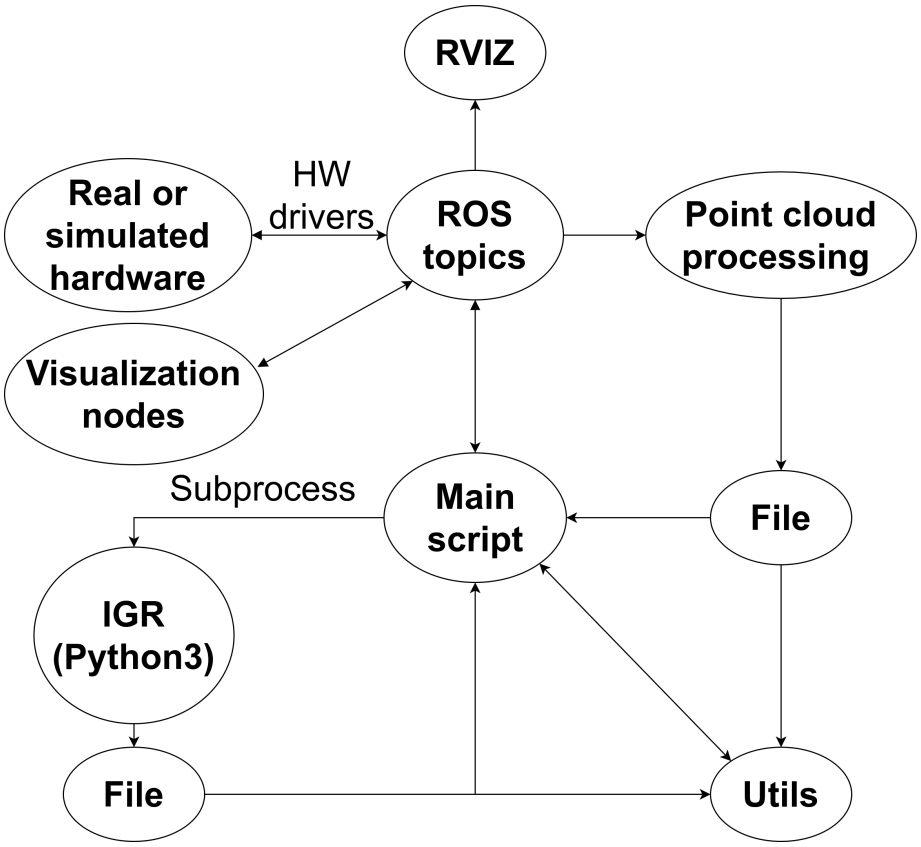


Figure 4.1: High-level schematic of pipeline interconnection.

The individual modules are interconnected using Robot Operating System (ROS) [38]. ROS allows us to freely connect different modules, written in different programming languages (Python2, C++). Also, we designed the whole pipeline to be easily convertible to other hardware (or simulated hardware) by just replacing the corresponding drivers. A

high-level schematic of the entire pipeline is shown in Fig. 4.1.

Even though ROS2 had already been available and Python2 had already reached the end of its life, we decided to stick to the original ROS. The main reason is that we want to, eventually, integrate this work in team-wide projects which use the same software version. Also, drivers for our robotic arm were—at the time of the start of the project—only available for ROS Melodic and older. The pipeline was therefore implemented and tested with ROS Melodic on Ubuntu 18.04 (ROS Kinetic with Ubuntu 16.04 could work in theory, but one would need to implement functions not yet available in this version of ROS) with Python2.7. However, as has been stated in Section 3.3, the neural network uses Python3 (tested with version 3.6). To overcome this, the network is called using Python subprocess library. The implementation can be found in the attachment of this work.

4.1.1 ROS Overview

To allow the reader better orientation in the upcoming text, we provide a list of basic ROS terms.

- nodes - standalone programs; usually simple; used, for example, to read data from sensors and publish it to topics;
- topics - used for data transport between nodes; named "ports"; nodes can subscribe or publish to it;
  - *E.g.,* the robot controller reads data from joints and publish it to */joint_states* topic, which is subscribed by a node that detects collision from joint torques;
  - there can be more nodes subscribing or publishing to one topic;
  - each topic has a given message type;
- messages - data types used to pass data to/from topics; ROS includes predefined messages for basic types or users can create their own; can be nested;
  - *E.g., sensor_msgs/JointState* message is defined as:
    - *std_msgs/Header* header (*std_msgs/Header* is a nested message type)
    - *string[]* name
    - *float64[]* position
    - *float64[]* velocity
    - *float64[]* effort
- services - special type of node, which is idle most of the time and is activated by calling from other nodes; request and reply system; similar to functions;
  - requests and replies are defined in *.srv* files similar to messages;
  - convenient, for example, when one needs to call a function written in C++ from Python;
    - *E.g.,* our pipeline includes service *parametrize_cartesian_path*, that can parametrize the trajectory in such a way not possible in Python;

- subscribing - a way to listen to a given topic, *i.e.*, a way to read data from a topic; usually achieved by a class instance periodically running in the background, which triggers a callback when new data are available;
  - can be synchronous or asynchronous;
  - data are received as a message;
- publishing - a way to write data to a given topic; data are sent as messages;
- ROSbag - a way to save data from ROS; whole topics are saved; rosbags can then "replay" the experiments and can be used without hardware that collected the data; can be replayed at arbitrary speed;
- RVIZ - visualization tool from ROS; can visualize the robot or data from topics, *e.g.*, with markers or arrows;
- launch files - files written in XML; allow to run more nodes or even other launch files at once; allow to set ROS parameters; accept arguments;
- parameters - nodes can share parameters on the ROS parameter server; can be set from launch files or from nodes.

4.2 Robot and Cameras

In this work, a 7 Degrees of Freedom (DoFs) robotic manipulator *Kinova Gen3* with a *Robotiq 2F-85* gripper was used. The robot is equipped with a torque sensor in each joint, which allows us to detect collisions without any external sensor. The arm also contains a built-in camera *Intel RealSense D410* on its wrist, which was used for to obtain point clouds from behind the object in the accompanying experiments.

To control the robotic arm, an official *ROS Kortex* driver from Kinova Robotics was used [39]. The driver provides access to controllers the robot (Cartesian and Joint space control was used in this work) and to read internal variables of the robot, *e.g.*, Cartesian position or joint torques. The gripper was controlled with a custom utility using messages and services definitions from *ROS Kortex*—the gripper was controlled in velocity mode.

The robot is mounted on a table, together with a second camera *Intel RealSense D435*. The second camera is fixed to the table and serves to capture single-view information. The setup can be seen in Fig. 4.2. Both of the cameras work on active Infrared (IR) stereoscopic depth principle, allowing to easily capture point cloud of the scene. The stereoscopic vision is based on the principle of having two sensors capturing the same scene with some displacement, and estimating the depth from disparities between matching key-points in both images. The disparities can be found by, for example, Sum of Squared Differences block-matching algorithm [40]. With estimated disparities, the depth can be computed as

$$z = \frac{f \cdot B}{disparity}, \tag{4.1}$$

where f is a focal length and B is a baseline (distance between the two cameras). To properly establish the key-points, the images should come from two exactly parallel views, *i.e.*, there should be only horizontal displacement between the sensors. If it is not the case, the images should be rectified, *i.e.*, reprojected to a common plane. Additionally, the active IR technology adds new details to the scene outside the visible spectrum by projecting an IR grid—with a secondary benefit of adding extra light when light conditions are poor.

Figure 4.2: Real world setup with the robot, camera and used dataset. The black ellipse contains objects used for reconstruction accuracy experiments and the blue one includes objects for grasping. From Rustler *et al.* [1].

Data from the camera were collected using *ROS wrapper for Intel Real Sense Devices* [41]. The wrapper provides comfortable access to the point cloud or to both RGB and depth information individually. To work with the point clouds, the PCL library [42] (in both Python and C++) and Open3D library [43] (in Python) were used.

4.3 Simulation

Even though we had the real hardware available from the start of the project, we decided to test the principles in simulation before moving to the real world. For this purpose, a simulation environment was created—with physics simulation using MuJoCo [44]. We based the environment on Github repository of Jan Behrens [45]. The simulation includes the same robot and gripper as in the real world. MuJoCo engine simulates all physical aspects of the movements and collisions, including joint torques. In combination with OpenGL, the environment also includes a virtual camera rendering an image with depth information from the simulation. Graphical output is visualized using RVIZ—a screenshot can be seen in Fig. 4.3.

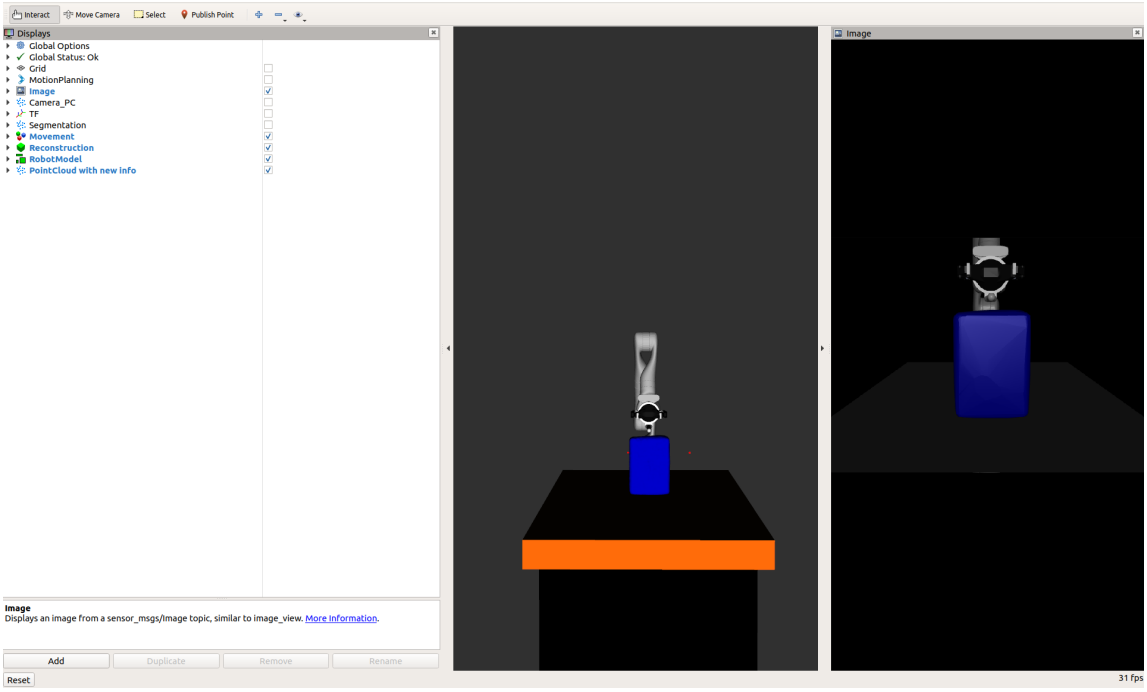


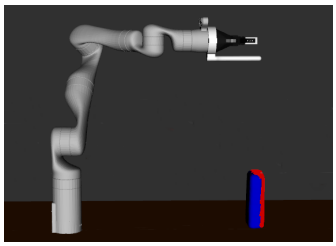
Figure 4.3: Screenshot of RVIZ with simulation environment. Options of the environment (left, movable model of the robot with object (middle) and virtual camera (right).

The simulation is implemented as a ROS package. *Kortex* driver [39] for the real Kinova arm can be used to send commands to the simulation environment. This allowed us to implement and test algorithms in the safety of the simulation and then directly transfer it to the real arm by telling the driver to communicate with the physical hardware.

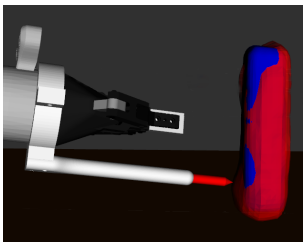
We also added multiple graphical real-time features selectable in the RVIZ environment, making the simulation an ideal benchmarking tool. Moreover, the features can be also used in the real world. Some examples are shown in Fig. 4.4 and all the features are listed below:

- MotionPlanning – basic ROS interface to move the robot by hand;
- TF – basic ROS visualization of transformation frames;
- Image – image from the virtual camera;

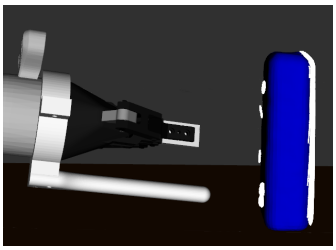
- Camera point cloud – point cloud from the camera;
- Segmentation – segmented point cloud of the scene;
- Movement – an arrow showing a direction of the linear movement towards the object;
- Reconstruction – mesh showing the current reconstruction;
- RobotModel – model of the robot;
- Point cloud with new information – point cloud including also the haptic data.



(a) : Segmentation of the point cloud (red).



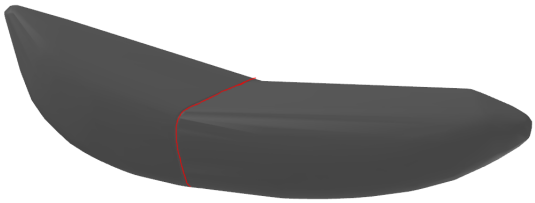
(b) : Current reconstruction and direction arrow (both red).



(c) : Point cloud with haptic information.

Figure 4.4: Features of the simulation environment.

The most notable difference between the real world and the simulation is in the interpretation of objects. In the simulation, the objects are modeled in MuJoCo as its convex hulls. That caused a problem for collision detection because the arm was not able to go through holes or to touch curved objects because the convex hull did not include the holes. The problem was solved by convex decomposition of the object by V-HACD [46]. The object in the simulation is then represented as a set of convex models with fixed transformations. An example of convex decomposition of a triangle mesh can be seen in Fig. 4.5a. The banana is divided to two convex objects—the connection is highlighted with red line to be better visible. Convex hull of the same triangle mesh is in Fig. 4.5b. We can see that if the decomposition was not used, the space under the curvature of the banana would be inaccessible.



(a) : Convex decomposition of banana mesh. The connection between two convex parts is highlighted with red line.

(b) : Convex hull of banana mesh.

Figure 4.5: Convex decomposition and convex hull of banana mesh.

4.4 Point Clouds

From the real Intel RealSense cameras, the point cloud can be published directly from official ROS Wrapper. Both intrinsic and extrinsic parameters of the camera were calibrated before the experiments. Intrinsic parameters were calibrated by the official *Intel RealSense D400 Series Dynamic Calibration Tool*[1]. Extrinsic parameters were calibrated using *easy_handeye* calibration tool [47] and fine-tuned by hand. In the case of the simulation, the point cloud is computed directly when the image is rendered. Segmentation of the object from the scene is done using *Point cloud segmentation* package [48]. The package is implemented as a ROS node and includes classical segmentation methods, *e.g.*, region growing or plane segmentation. The nodes subscribe to a selected point cloud topic and publish segmented point cloud on a new topic. Segmentation can be computationally demanding, so the node works on demand, *i.e.*, it works only when the output topic has at least one subscriber. Output of the segmentation can be changed by user using arguments when launching the node, or using services. The user can select which segmentation method to use, which part of the scene to use for segmentation, what is the base frame of the output point cloud and other parameters of the individual segmentation methods.

We experimented with all segmentation methods and the best results were obtained with region growing and plane segmentation. The plane segmentation[2] is based on the iterative finding of the biggest plane (using RANSAC) of the current point cloud and removing all points in the plane from the point cloud until the last plane is found or no more points remain. The method fails in a situation when the desired object is planar, *e.g.*, box or block of wood. On the other hand, the region growing algorithm[3] is based on "clustering" of neighboring points based on a criterion (angle between normals in our case). The region growing algorithm can remove the table and background the same as plane segmentation but is more successful when the object is planar. Having that in mind, all experiments were conducted with the region growing algorithm.

The point cloud from the segmentation node can miss some parts, be scattered, and contain outliers. To overcome these issues, we firstly collect multiple samples (the point cloud can slightly vary over time) and then apply the following filters:

- VoxelGrid filter – for downsampling of the point cloud; creates 3D voxel grid over the point cloud and then points of a new point cloud are approximated as a centroid of individual voxels;
  - leaf size parameter decides the size of boxes in the voxel grid;
- Moving Least Squares (MLS) – used for smoothing of point cloud;
  - the smoothing is controlled by *smoothing factor* and polynomial order of the fitted curve;

[1] https://www.intel.com/content/www/us/en/download/645988/intel-realsense-d400-series-dynamic-calibration-tool.html
[2] https://pcl.readthedocs.io/projects/tutorials/en/latest/planar_segmentation.html
[3] https://pcl.readthedocs.io/projects/tutorials/en/latest/region_growing_segmentation.html

- Statistical Outlier Removal (SOR) – used for removing outliers; assumes that the distribution of points in the point cloud is Gaussian and removes all points with a mean distance (from its neighbors) bigger than a threshold.

All the filters are implemented using PCL library [42]. The postprocessing itself is implemented as a service and the user can set the *leaf size* parameter for VoxelGrid filter and *smoothing factor* for MLS. Example of different stages of the postprocessing can be seen in Fig. 4.6. Note that the "wobbly behavior" in Fig. 4.6a is most probably caused by imperfect calibration of the intrinsic parameters of the camera. However, we were not able to calibrate the parameters better and we decided to stick to it as the results are still less noisy than with the default parameters.

(a) : Multiple outputs from segmentation merged to one point cloud.

(b) : Point cloud after MLS.

(c) : Point cloud after SOR.

Figure 4.6: Point cloud postprocessing shown on output from segmentation.

4.5 Contact Detection

Neither the manipulator nor the gripper has any natural source of tactile sensing. Since the robot is equipped with joint torque sensors, we decided to use them. There are works using particle filters [49] or neural networks [50] for contact detection, however, these methods are more focused on the precise localization of the collision. We were more focused on a binary classification of the contacts, so the detection using *Cumulative sum (CUSUM)* algorithm [51] was selected. This algorithm can detect changes in a signal (joint torque values in our case) and is originally defined in Algorithm 2. The algorithm also utilizes drift d to compensate natural changes in the signal, *e.g.*, torque needed to make a movement.

However, the procedure in Algorithm 2 is used in offline detection in single-dimensional signals. We want to detect changes in real-time and in any joint of the robot. So we changed the definition to Algorithm 3. The difference is that signal and both changes are now defined as multi-dimensional vectors. Also, the vectors contain values only from the last time step, which removes computationally demanding addition of new data to the vectors. In ROS, the functionality is achieved with Python class acting as a subscriber to a topic containing external torques of the robot. The class contains the vectors as internal

Algorithm 2 Two-sided CUSUM Algorithm

Inputs: threshold t, drift d;
$s_0 \leftarrow$ signal at time zero;
$g_0^+ \leftarrow 0$; ▷ No positive change at time zero
$g_0^- \leftarrow 0$; ▷ No negative change at time zero
$k \leftarrow 1$; ▷ Start detecting changes from time one
while $g_k^+ < t$ and $g_k^- < t$ **do**
    $s_k \leftarrow$ signal at time k;
    $g^+[k] \leftarrow max(g_{k-1}^+) + s_k - d, 0)$;
    $g^-[k] \leftarrow max(g_{k-1}^-) - s_k - d, 0)$;
end while

variables. After the threshold is exceeded, the class sends a stop command to the robot and unsubscribes from the topic.

Algorithm 3 Online Multi-dimensional Two-sided CUSUM Algorithm

Inputs: threshold t, drift d;
$\mathbf{s}_{last} \leftarrow$ signal at time zero;
$\mathbf{g}_{last}^+ \leftarrow \mathbf{0}$; ▷ Vector of zeros with length corresponding to DoFs
$\mathbf{g}_{last}^- \leftarrow \mathbf{0}$;
while all($\mathbf{g}_{last}^+ < t$) and $all(\mathbf{g}_{last}^- < t)$ **do** ▷ function *all* returns True if all values of the vector meet the given condition
    $\mathbf{s} \leftarrow$ current;
    $\mathbf{g}^+ \leftarrow max(\mathbf{g}_{last}^+) + \mathbf{s} - d, 0)$;
    $\mathbf{g}^- \leftarrow max(\mathbf{g}_{last}^-) - \mathbf{s} - d, 0)$;
end while

The problem with the real robot (the simulated robot can return external torques directly) is that the torque values $\boldsymbol{\tau}$ returned by the robot contain both internally generated and external torques:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{dyn} + \boldsymbol{\tau}_{ext} = \mathbf{H}\ddot{\boldsymbol{q}} + \mathbf{h} + \mathbf{g} + \boldsymbol{\tau}_{ext}, \tag{4.2}$$

where $\boldsymbol{\tau}$ is a vector of torques in each joint, $\mathbf{H}$ is inertia matrix, $\mathbf{h}$ is vector of Coriolis and centrifugal forces, $\mathbf{g}$ is vector of gravity forces, $\mathbf{q}$ is vector of joint positions, $\boldsymbol{\tau}_{ext}$ is vector of external torques in each joint and $\boldsymbol{\tau}_{dyn}$ is vector of torques resulting from the robot dynamics. We are only interested in $\boldsymbol{\tau}_{ext}$, which can be easily done as

$$\boldsymbol{\tau}_{ext} = \boldsymbol{\tau} - \boldsymbol{\tau}_{dyn}. \tag{4.3}$$

However, there is no way how to obtain $\boldsymbol{\tau}_{dyn}$ directly from the robot. Therefore, we decided to compute the values in real-time with a dynamics simulator implemented in *OpenRAVE* [52]. These values are not precise, as torques in real world depends on properties which are not—and probably can not be—simulated in *OpenRAVE*, *e.g.*, temperature of the joints. In some cases, the inaccuracies make the robot stop too early or too late, but these are rare and it is still the best way to obtain the external torques.

4.6 Custom Finger

Initially, we wanted to exploit a closed gripper as a poking end-effector to keep the setup as self-contained as possible. However, after some experiments we noticed that the gripper is too wide and it requires a more complicated control of the movements to make it accurate enough. As a solution to this problem, we designed a custom finger-like end-effector made with a 3D printer. Visualization of the finger is shown in Fig. 4.7. Adding the finger helped to get more accurate, point-like contacts. Yet, it is not an ideal solution as it brings new sources of uncertainty, *e.g.*, error in placement on the real arm, position errors coming from insufficient stiffness of the 3D print, or collisions while grasping. However, overall, introducing the finger was still beneficial as it helped to fine-tune the pipeline.

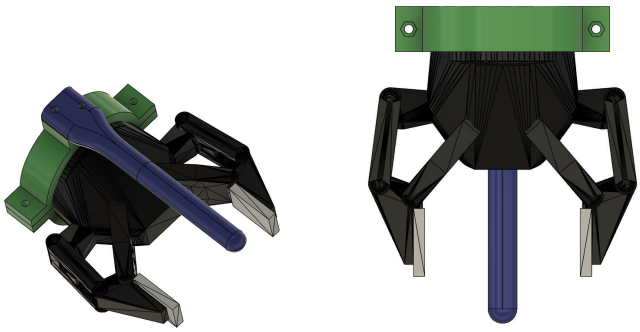


Figure 4.7: The custom made holder (green) and finger-like end-effector (blue) used for haptic exploration.

4.7 Evaluation Methods

To evaluate the performance of our approach, evaluation methods need to be selected. However, we have our ground truth in the form of triangular meshes and it is not straightforward to compare two meshes. In the literature, several comparison methods are used. Jaccard similarity (JS) was used by Lundell *et al.* [13]. The same method together with Hausdorff distance was used by Watkins-Vals *et al.* [30]. In Fan *et al.* [53], Chamfer distance (CD) was proposed as a robust distance metric. We decided to proceed with JS and CD. Both of the methods were used in *ShapeNet challenge*[4], and detailed description can be seen below:

- Jaccard similarity (JS) - measure of similarity, in %;
  - Intersection over Union, defined as

$$J(\mathbf{S}_1, \mathbf{S}_2) = \frac{|\mathbf{S}_1 \cap \mathbf{S}_2|}{|\mathbf{S}_1 \cup \mathbf{S}_2|}, \tag{4.4}$$

    where $\mathbf{S}_1, \mathbf{S}_2$ are two sets. In our case, $\mathbf{S}_1$ and $\mathbf{S}_2$ are voxel grids created from a reconstructed mesh and ground truth mesh;

[4] https://shapenet.cs.stanford.edu/iccv17/

- Pros:
  - easy to compute;
  - level of detail can be tuned by the resolution of voxel grids;
    - a resolution of $40 \times 40 \times 40$ was used;
- Cons:
  - in edge cases can be high even for non-similar objects;

- Chamfer distance (CD) - measure of distance, in mm;
  - for two sets defined as

$$d_{CD}(\mathbf{S}_1, \mathbf{S}_2) = \frac{1}{N} \sum_{\mathbf{x} \in \mathbf{S}_1} \min_{\mathbf{y} \in \mathbf{S}_2} \|\mathbf{x} - \mathbf{y}\| + \frac{1}{M} \sum_{\mathbf{y} \in \mathbf{S}_2} \min_{\mathbf{x} \in \mathbf{S}_1} \|\mathbf{x} - \mathbf{y}\|, \tag{4.5}$$

    where N, M are number of elements in $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. In this case, $\mathbf{S}_1$ and $\mathbf{S}_2$ are point clouds created by sampling the reconstructed and ground truth meshes;
    - vertices of the meshes can be also used, but the results are more robust when using point clouds, as the vertices can come from different distributions;
  - Pros:
    - can be computed fast even for bigger point clouds
      - fast K-nearest neighbor implementation can be used;
    - robust against a small number of outliers;
  - Cons:
    - for a higher number of outliers can be big even for similar objects;
    - for precise results, the number of sampled points from each mesh has to be large.

4.8 Baselines

To compare with other works, we prepared baselines of two types: (i) reconstruction baselines; (ii) haptic exploration policy baselines. The first group is intended for testing how good our reconstruction is compared to existing methods and to see if the iterative refining of shapes can also improve the reconstruction using these methods. The second group should show how efficient is our method in terms of haptic exploration.

The reconstruction baselines can be further divided based on their ability to estimate shape from incomplete point clouds:

- No estimation:
  - can only reconstruct parts of the objects with known information;
  - Poisson reconstruction (Poisson) [54]

        - solves a regularized optimization problem to obtain a smooth surface over given points;
        - efficient for whole point cloud;
    - Ball Pivoting Algorithm (BPA) [55]
        - rolls a "ball" with pre-defined radius over the points and if any 3 points are inside the ball, it creates a triangle;

- Partial estimation:
    - can partially estimate the whole shape, even though it is still limited by known points;
    - Convex Hull (Hull) [56]
        - simple creation of convex hull over point cloud;
        - can not predict holes;
    - Alpha Shapes (Alpha) [55]
        - generalization of convex hull;
        - helps to smooth the surface;

- Full estimation:
    - can predict the whole object;
    - Gaussian Process Implicit Surfaces (GPIS) [57]
        - uses Gaussian Processes (GP);
        - the reconstruction itself is created as an isosurface from mean μ and variance σ^2 of the GP;
        - our implementation is based on [58];
        - one can imagine it as a sheet of paper which is bent around the known points.

All baselines, except GPIS, were implemented using Open3D library for Python [43].

The second group of baselines geared to compare different haptic exploration policies consists of:

- GPIS:
    - even though we also use it as a reconstruction baseline, Yi *et al.* [25] proposed a method for tactile exploration using GP as a source of uncertainty;
    - for each point $\mathbf{x}$, GPIS returns a Gaussian defined by mean $\mu(\mathbf{x})$ and variance $\sigma^2(\mathbf{x})$;
    - the idea is to explore the location with the highest standard deviation $\sigma(\mathbf{x})$, as it can be seen as a measure of uncertainty;

- Random touching:
    - the algorithm takes the first feasible random position;

- the mesh is transformed to a voxel grid and all occupied voxels are taken as possible locations for exploration;
  - note that the meshes were created using IGR;
- the set of possible locations is shuffled and the locations are one by one tested for feasibility;
  - shuffled with *Numpy*[5] function based on pseudo-random number generator;

4.9 Grasping

To further evaluate the performance of our approach, we also tested the resulting reconstructions in grasping experiments. GraspIt! [59] was used to sample the grasp proposals. The Robotiq gripper was not available in the GraspIt! database, so we had to create a model for it. We used its ROS model and set its joints in GraspIt! to act like in the real world. An example of grasp proposal for a rectangular object from GraspIt! interface with the gripper can be seen in Fig. 4.8.

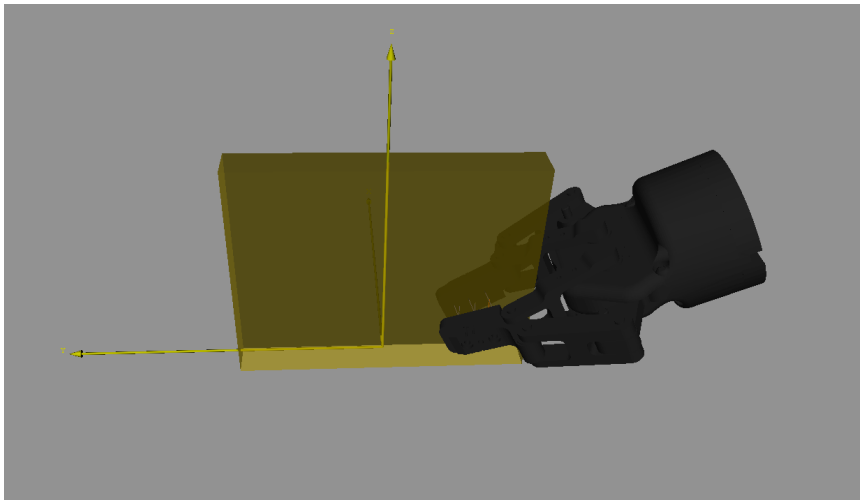


Figure 4.8: An example from GraspIt! interface with the gripper holding a rectangular object.

The grasping procedure is written in Algorithm 4. We prepared GraspIt! world with the gripper and a table with the same position as in the real world. The model of a table is used for disallowing the planner to plan grasps at which the gripper would collide with the table. The reconstructed mesh is then imported to this world (lines 2-4). To sample the grasps, the Simulated Annealing planner is used and the grasps are then sorted by their ϵ-quality [60] (line 5-6). The ϵ-quality is computed from Convex Hull of contact wrenches and the closer the ϵ is to one, the more efficient the grasp should be. The Simulated Annealing planner is from the family of Eigengrasp planners. The idea of these planners is to reduce the dimensionality of search space by reducing the number of DoFs by using only the "effective" ones [61]. For example, the human hand can be reduced to two Eigengrasps and our gripper can be controlled by only one. The planner searches for the Eigengrasps and a 6D pose of the gripper (position and orientation). We run the planner for 70000

[5] https://numpy.org/doc/stable/reference/random/generated/numpy.random.shuffle.html

iterations using "Hand Contacts" search energy. The energy serves for guiding the gripper towards the selected object through a given objective—the selected formulation tries to get the gripper as close to the object as possible.

The sorted proposals are then one by one tested for feasibility. If a feasible grasp is found, the robot is moved 10cm from the grasp position along a direction vector—z-axis of the gripper. Then it is moved linearly to grasp position (lines 9-10). Closing of the gripper is initialized until a predefined threshold of the current in its joints is exceeded (lines 11-13). To evaluate quality of the grasp, the robot is moved 10 cm upwards and backwards. Finally, the last joint of the arm is rotated by 90° in both directions (lines 14-15).

Algorithm 4 Grasping

```
1:  Inputs: mesh m, current threshold t;
2:  load GraspIt! interface;
3:  load world with the gripper;
4:  import m to the world;
5:  G ← results from Simulated Annealing;
6:  G ← sort G by ϵ-quality;
7:  for each g in G do
8:      if g is feasible then
9:          go 10 cm from the grasp;
10:         go to grasp;
11:         while gripper current < t do
12:             close the gripper;
13:         end while
14:         move 10 cm upwards and backwards;
15:         rotate the last joint by ±90°;
16:         break
17:     end if
18: end for
```

Chapter 5

Visuo-Haptic Uncertainty-Driven Object Shape Completion

Using real world measurements S always comes with noise and uncertainty, even more when our work combines visual and haptic data. The visual and haptic data are represented as v and h, respectively. Visual data suffer from light conditions or error in calibration. Haptic data can be influenced by inaccuracies in contact detection and joint position readings. The goal is to complete a shape of an object O. The object can be modeled in several ways. Throughout this work (as can be seen in Fig. 1.1), it can have a form of point cloud (input), SDF (output from the neural network), mesh (grasping and comparison with ground truth) and voxel grid (uncertainty computation). The object O can be probabilistically modeled as

$$\mathsf{P}(O|S), \tag{5.1}$$

where O represents the occupancy of the object, which is modeled as a voxel grid $O = \{O^k\}$ where k is the index of a voxel such that $\mathsf{P}\{O^k\}$ is the probability that voxel k is part of the object.

Our goal is to use haptic exploration in such a way that, at each time t, the new touch improves the reconstruction as much as possible. Mathematically expressed as

$$\underset{h_t \in H}{\operatorname{argmin}} \operatorname{Var}(O_t|v, h_{1:t-1}, h_t), \tag{5.2}$$

where Var is a variance defined in Eq. (5.3), $h_{1:t-1}$ are the data from previously executed haptic explorations and H is the set of all possible haptic explorations. Visual data v are the same through all haptic explorations and are captured in time $t = 0$, when the Eq. (5.2) does not include any haptic information.

Probabilistic model of the shape of a 3D object is difficult to express, mainly due to the high dimensionality. Therefore, we rather decided to approximate the model with a set of samples $o^{1:S}$ from an underlying generative shape distribution $\mathsf{P}(O_t|v, h_{1:t-1})$. The sampling itself is described in more detail in Section 5.1.

Having the approximate model, one still needs to define the haptic exploration h_t that is solution of Eq. (5.2). The h_t is, in fact, a position in the space where we want to explore the object. To select h_t, we chose to compute the variance of the shapes

$$\operatorname{Var}(o^{1:S}) = \frac{\sum_{i=1}^{S}(x_i - \overline{x})^2}{S-1}. \tag{5.3}$$

Assuming a set of samples are given in the form of voxel grids, we are looking for the voxel k with the highest variance, formally expressed as

$$\underset{k \in K}{\operatorname{argmax}} \operatorname{Var}(O^k), \tag{5.4}$$

where K is number of voxels, k is a voxel that minimizes the Eq. (5.4) and $\operatorname{Var}(O^k)$ is the variance of the shape samples o^s for that voxel. The voxel k is the desired position of touch, *i.e.*, the haptic exploration h_t to maximize Eq. (5.2). However, generally there can be more than one voxel k with the highest variance. The exact procedure of selecting the final voxel can be found in Section 5.2.

In conclusion, we propose Algorithm 5 that combines the principles mentioned above. Some steps are illustrated in the accompanying video at `https://youtu.be/Ft1PUYRNFHw`. The algorithm mimics Fig. 1.1 and for a given number of haptic explorations, it firstly reconstructs the object with current information (lines 8-14) and then finds the best position for haptic exploration (lines 15-18)—collision box estimated from current shape is inserted into the planner so that the robot can avoid the object. The latent code $\hat{\mathbf{z}}_0$ for the first iteration is initialized randomly. In other cases, the optimization is started with $\hat{\mathbf{z}}_g$ from the previous iteration. With the information after all touches, the final shape is reconstructed (lines 19-23).

Algorithm 5 Visuo-Haptic Uncertainty-Driven Object Shape Completion

1: **Inputs:** point cloud $\mathbf{P}$, number of haptic explorations M, number of gradient-descent steps G, steps before storing latent shape L;
2: **Output:** Final shape completion O ;
3: $\mathbf{H} \leftarrow$ empty set of haptic data;
4: $\mathbf{P}_0 \leftarrow \mathbf{P}$;
5: $\hat{\mathbf{z}}_0 \leftarrow$ Sample initial latent code;
6: **for** $m \leftarrow 1, \ldots, M$ **do**
7: $\quad \mathbf{Z} \leftarrow$ empty set of latent codes;
8: $\quad$ **for** $g \leftarrow 1, \ldots, G$ **do**
9: $\quad\quad \hat{\mathbf{z}}_g \leftarrow$ Optimize $\hat{\mathbf{z}}_{g-1}$ over $\mathbf{P}_{m-1}$ using Eq. (3.5);
10: $\quad\quad$ **if** $g \mod L == 0$ **then**
11: $\quad\quad\quad \mathbf{Z} \leftarrow \mathbf{Z} + \hat{\mathbf{z}}_g$;
12: $\quad\quad$ **end if**
13: $\quad\quad \mathbf{V} \leftarrow$ Reconstruct shapes from $\mathbf{Z}$ and calculate their variance;
14: $\quad$ **end for**
15: $\quad h_m \leftarrow$ Calculate next touch using Eq. (5.4) and $\mathbf{V}$;
16: $\quad \mathbf{H} \leftarrow$ Execute h_m and record the touch point;
17: $\quad \mathbf{P}_m \leftarrow \mathbf{P} + \mathbf{H}$;
18: $\quad \hat{\mathbf{z}}_0 \leftarrow \hat{\mathbf{z}}_g$;
19: **end for**
20: **for** $g \leftarrow 1, \ldots, G$ **do**
21: $\quad \hat{\mathbf{z}}_g \leftarrow$ Optimize $\hat{\mathbf{z}}_{g-1}$ over $\mathbf{P}_I$ using Eq. (3.5);
22: **end for**
23: $O \leftarrow$ Reconstruct shape using $\hat{\mathbf{z}}_g$;
24: **return** O;

5.1 Sampling of Shapes

The sampling process $o^s \sim \mathsf{P}(O_t|v, h_{1:t-1})$ to obtain samples $o^{1:S}$ using CIGR can be done with two alternatives. The more obvious, having in mind that the output from CIGR is stochastic, is to sample multiple latent codes $\hat{\mathbf{z}}_{i,t}$ (as described in Eq. (3.5)) and use them to reconstruct the meshes, *i.e.*, to run the network S times. The mean $\overline{x}$ in Eq. (5.3) is then computed as an average shape from all samples. However, this approach is time consuming as it requires multiple runs of CIGR and cannot be easily parallelized due to a high demand on computational resources. Also, as shown in Fig. 5.1, when searching for the voxel with the highest variance, there are fewer selected voxels than with the second method. At first sight, that may not seem as a problem, but not all voxels are reachable by the robot and it is more robust to have more options. The second issue is that the most uncertain voxels tend to be on the front side of the object, where we already have information from vision. The first issue could be fixed by not taking only the voxels with the highest variance, but using an interval. However, that would conflict with the idea of employing only the most beneficial haptic exploration.

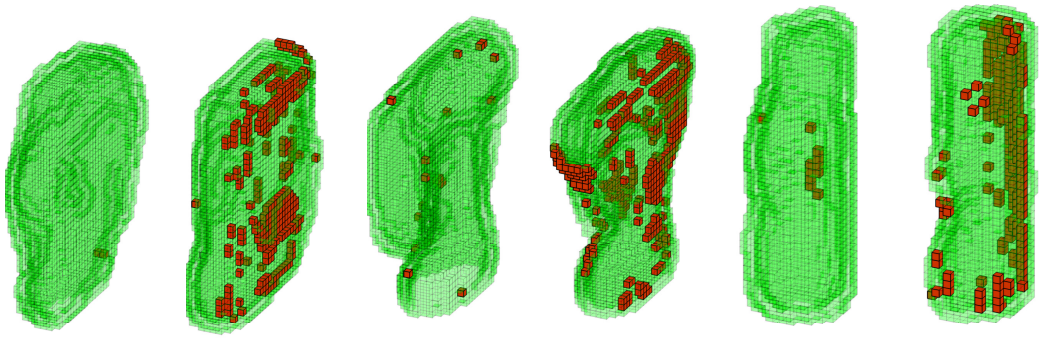


Figure 5.1: Different uncertainty methods. First voxel grid in each pair corresponds to the sampling from multiple latent codes and the second one is created by sampling single latent code at different time steps. From Rustler *et al.* [1].

The second alternative is to compute just one latent code $\hat{\mathbf{z}}_{i,t}$ and take intermediate samples at different time steps t of the later part of gradient optimization, *i.e.*, run the network only once, and save samples during the optimization. We took samples near the end of the optimization, *e.g.*, if we optimize for 800 iterations, we would take samples at iterations 650, 700, 750 and 800 (iteration 800 is used as mean $\overline{x}$ in variance defined in Eq. (5.3)). Although the first alternative has a stronger mathematical background, this one performed better in empirical comparison. Looking at Fig. 5.1, one can see that this method produces more uncertain voxels, and the voxels are also more concentrated on unseen parts of the objects. The reason behind this behavior probably comes from the nature of gradient descent. The optimizer will always go in the direction of the steepest descent, with bigger steps in the beginning and smaller steps around the minimum. When optimizing the latent code from an incomplete point cloud, we observed that there is not enough information for the optimizer to find the exact solution, and it often oscillates around the minimum. We assume that during the oscillation, the known parts are still slightly changing depending

on the current batch, but more changes should occur for unexplored areas, as they will probably change almost randomly because the current latent code is not dependent on them. The parts which change the most have the biggest variance and are recognized as the most uncertain by the pipeline. This resembles Metropolis sampling [62] in that samples are generated from a supposedly converged Markov chain, however, in our case we do not use the Metropolis rejection rule in the optimization process. Yet, the stochasticity is introduced through sampling mini-batches in the optimization.

For both methods, the final shape depends on the latent code at step zero (randomly initialized or carried over from the previous iteration of shape refining) and the outputs may slightly vary—both shape and uncertain voxels—every time the same input is reconstructed and evaluated. However, for the second alternative, the uncertain areas are created more consistently in the unexplored parts. An experiment comparing these two methods is described later in Section 6.1.1, specifically in Fig. 6.6. Considering also the fact that the second method is significantly faster (the difference is in tens of seconds), we ended up using that alternative.

5.2 Impact Point Computation

The whole process of impact point selection is described in Algorithm 6. The first step for uncertainty computation is the transfer from the triangle mesh space to the voxel grid space and vice versa. In our implementation, transfer to voxel grids is achieved with *Binvox* utility [63]—to read the voxel grids from file a Python library *binvoxrw* [64] was used. We experimented with different resolutions of the grids and a resolution of 40^3 was selected, as it has the best quality-speed ratio. With the mesh transformed to a voxel grid, we can easily compute the variance from Eq. (5.4) over each voxel (lines 3-4)—only voxels that are occupied in the voxel grid of the mean mesh are taken into account. The transfer back to the triangle mesh space can be done easily, because we know the position of the mesh in the world coordinates and we can compute the voxel to meter ratio from the bounding boxes of the meshes and voxel grids.

In the general case, there are multiple voxels with maximal variance. To select only one, the voxels are divided into clusters with *MeanShift* clustering [65]—implemented in *scikit-learn*[1] library for Python. For each cluster, a centroid is computed as the mean of all voxels in the cluster and converted back to the world coordinates. We required the touches to be as robust as possible. To accomplish that, the clusters are sorted by their "flatness" (lines 5-6) to avoid, for example, sliding of the end-effector over the edge. We define flatness as the sum of angles between the normals of the 10 nearest triangles on the mesh to the computed impact position—if all normals have the same direction, the surface is flat. The angle Φ between two vectors a and b is defined as

$$\Phi(a, b) = atan_2(|a \times b|, a \cdot b), \tag{5.5}$$

where $atan_2$ is the arctangent function that returns the correct sign by choosing the quadrant correctly; $\times$ stands for cross product and $\cdot$ stands for the dot product of two vectors.

[1] https://scikit-learn.org

We do not want the robot to plan a trajectory directly to the impact position. Instead, a start position 10 cm away along a normal from the closest triangle is prepared. Robot is moved to this position and then it starts a linear movement to the original impact point. However, not all positions are feasible for the robot. Therefore, the start positions are one by one (in order sorted by flatness) tested for feasibility until a feasible position is found or no position is available (lines 7-13).

Algorithm 6 Impact Point Computation

1: **Inputs:** mesh m;
2: **Outputs:** impact position $\mathbf{p}$, start position $\mathbf{s}$;
3: $V \leftarrow$ voxelization of m;
4: $\mathbf{K} \leftarrow$ voxels with maximal variance using Eq. (5.4);
5: $\mathbf{C} \leftarrow$ clusters from $\mathbf{K}$ using MeanShift clustering;
6: $\mathbf{P} \leftarrow$ sorted impact points computed as centroids from $\mathbf{C}$;
7: **for each** $\mathbf{p} \in \mathbf{P}$ **do**
8: $\quad \mathbf{s} \leftarrow$ position 10 cm from $\mathbf{p}$;
9: $\quad$ **if** $\mathbf{s}$ is feasible for the robot **then**
10: $\quad\quad$ **return** $\mathbf{p}, \mathbf{s}$; ▷ Return and end the function
11: $\quad$ **end if**
12: **end for**
13: **return** $\varnothing, \varnothing$; ▷ If no feasible position found, return empty sets

5.3 Visual-only Approaches

We decided to also develop a method for visual-only shape completion. It is not a novel problem and it has been already studied. Our goal was not to develop something new, but to have a comparison with our visuo-haptic approach, both in performance and complexity. Our robotic manipulator is equipped with a built-in camera on its wrist, which is great to be used to scan objects. The idea is to collect samples all around a object to get a full point cloud. However, most of the depth sensors require a minimal distance from the scanned objects (in tens of cm; 18 cm for our sensor). For that reason, it is difficult or even unfeasible for small and middle-range robots to scan the whole object. For example, in [17], *KUKA KR16* with reach of 1612 mm was used—almost double the reach our manipulator has (902 mm). Other authors [19, 20] resolved this issue by adding a programmable rotor which can turn the object. We did not want to include another mechanical part to our setup, so we decided to combine the cameras on the table and on the robot instead, which allowed us to scan the "front" side of the object even though the manipulator could not see it.

We present two different approaches. One with fixed positions around the object (Passive visual-only shape completion (Pas-VO)) and one which uses the uncertainty computed in Algorithm 5 (Active visual-only shape completion (Act-VO)).

5.3.1 Passive Visual-only Shape Completion

The procedure is described in Algorithm 7. Firstly, a point cloud from the fixed camera on the table is taken and the object is segmented from the scene (lines 3-5). The bounding box of the object is computed and the center of the object is estimated by deducing 5 cm in the x-axis (lines 6-7)—collision box around the estimated center is added to the planner so that the robot can avoid the object. In [20], the object was scanned from four views on a circle. We scan the object from a smaller distance, so we decided to use the view from the table camera and five more views on a circle around the computed center. The views are taken each 45° in the range from 90° to 270°—angle 0° approximately corresponds to the view from the camera on the table (line 8). The positions of the arm during scanning can be seen in Fig. 5.2. At each position, the sample is taken, the object is segmented from the scene and the resulting point cloud is saved to a file (lines 9-14).

Algorithm 7 Passive Visual-only Shape Completion

1: **Inputs:** radius r;
2: **Outputs:** final point cloud $\mathbf{f}$;
3: $\mathbf{p} \leftarrow$ point cloud captured by the table camera;
4: $\mathbf{o} \leftarrow$ segment the object from $\mathbf{p}$;
5: save $\mathbf{o}$ to file;
6: $\mathbf{b} \leftarrow$ bounding box of the captured point cloud;
7: $\mathbf{c} = \mathbf{b} - [5\,\text{cm}, 0\,\text{cm}, 0\,\text{cm}]$; ▷ Deduct 5 cm in the x-axis
8: $\mathbf{A} \leftarrow$ set of positions on a circle with center in $\mathbf{c}$ and radius r;
9: **for each** $\mathbf{a} \in \mathbf{A}$ **do**
10: move to $\mathbf{a}$;
11: $\mathbf{s} \leftarrow$ take a sample from the camera on the wrist;
12: $\mathbf{o} \leftarrow$ segment the object from $\mathbf{s}$;
13: save $\mathbf{o}$ to file;
14: **end for**
15: $\mathbf{f} \leftarrow$ combine the individual samples to one point cloud with ICP;
16: $\mathbf{f} \leftarrow$ apply postprocessing on $\mathbf{f}$;
17: **return** $\mathbf{f}$;

All individual samples are combined. In theory, one could transform the point clouds to the world frame (using positions of the arm camera computed from forward kinematics) and create a full point cloud. However, our robot is not meant for industrial usage and thus its encoders are not perfectly precise. In addition, the two cameras are not calibrated together perfectly. The errors are relatively small (in the range of millimeters), but it is still too much for precise reconstruction. Thus, the samples need to be fitted. We experimented with the basic Iterative Closest Point (ICP) algorithm [66], generalized and plane ICP [67], colored ICP [68] or multiway reconstruction of indoor scenes [69]. However, the experimental results showed that the samples are fitted the best when classic ICP with a small number of iterations (1-5) is used—the same approach was used in [17]. And finally, the same postprocessing as in Section 4.4 is applied on the output point cloud.

Note that the object is not lifted during scanning and thus the bottom part of the object is not included in the final point cloud.

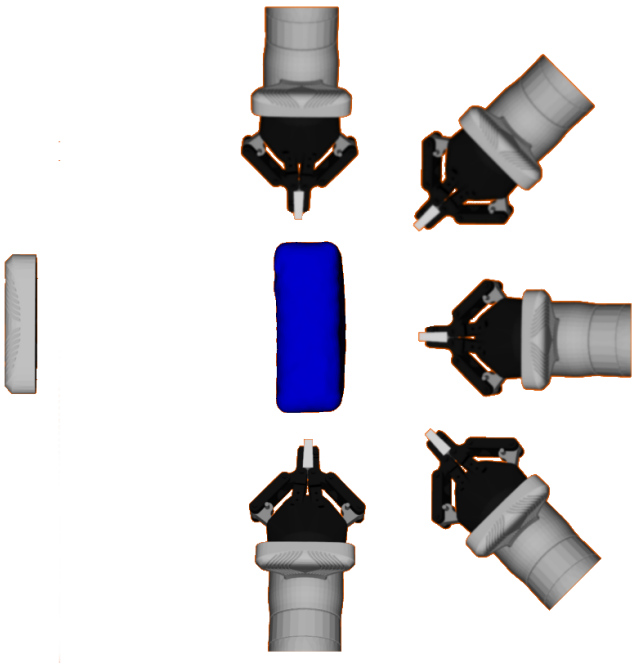


Figure 5.2: Position of the static camera on the table and approximate positions of the robotic arm during scanning (90° to 270° on a circle around the object).

5.3.2 Active Visual-only Shape Completion

The procedure is almost the same as in Algorithm 5 for visuo-haptic shape completion—meshes and voxel grids are used to compute the uncertainty. The main difference is that instead of haptically exploring the object (line 16), a point cloud from the camera on the wrist of the robot is saved and added to the main point cloud. Unlike the approach with fixed position described in Section 5.3.1, guiding the arm with uncertainty does not ensure that there will be overlaps between individual samples. Because of that, the ICP (or other fitting algorithms) cannot be used. That results in possible noise and errors. However, as this method is used only for comparison and is not the main contribution of this work, further improvements are left for future work.

We wanted to capture as big a part of the object as possible, but sometimes only views from a lower distance to the object are feasible. The position of the end-effector is computed the same as in Algorithm 6, only instead of testing a position 10 cm from the impact point for feasibility, positions 20, 15 and 10 cm are tested. Note that in this case, the robot takes the samples at those positions and does not continue to the impact point calculated in the same way as during haptic exploration. The distances are expressed for the end-effector, but the camera is another 15 cm further along the z-axis of the end-effector, so the possible views are from 35, 30 or 25 cm. View from 35 cm usually cover bigger part of the object, but view from 25 cm is still beneficial.

Chapter 6

Experiments and Results

This section describes the experiments and contains an analytical evaluation of the results. Experiments in both real and simulated world will be presented, evaluating the reconstruction efficiency with various methods. The performance is also tested with grasping. In Fig. 6.1 we provide examples of reconstructed objects from all baselines. Active visuo-haptic shape completion (Act-VH), described in Fig. 1.1, includes our implementation of IGR named Shape Completing IGR (CIGR). However, to avoid misunderstandings, we decided to use the term Act-VH when talking about the whole pipeline. And CIGR is used when referring only to the reconstruction method.

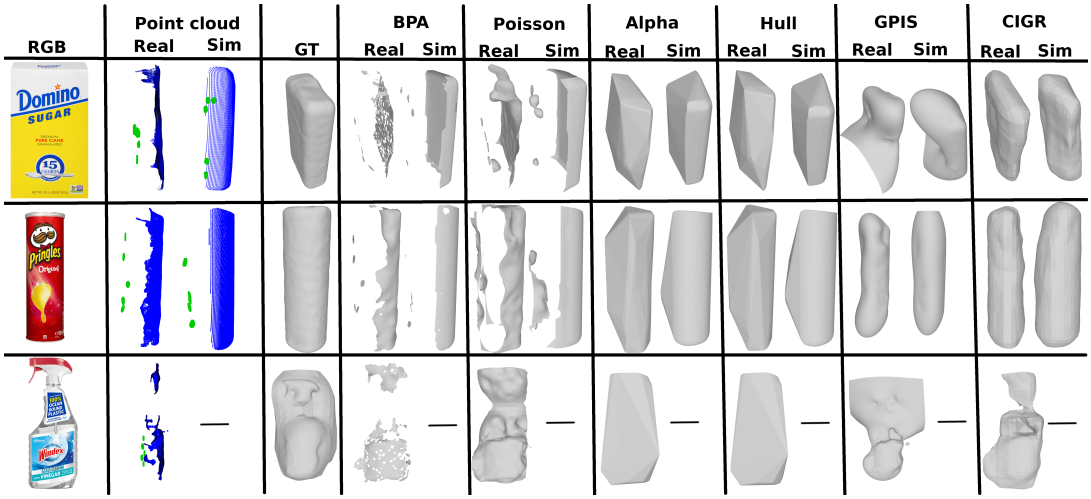


Figure 6.1: Reconstruction examples in simulation (Sim) and real world (Real) with all methods on three objects: a basic rectangular object (upper row), the object for which Act-VH achieved the worst grasp success rate (middle row), and an adversarial object (bottom row). There is no reconstruction in simulation for the adversarial object as no correct ground truth importable to the simulator is available. Touches are highlighted in the point clouds with green color. Act-VH with 3D printed finger was used to collect the point clouds. From Rustler *et al.* [1].

The shape completion neural network was trained on a dataset created by the procedure described in Section 3.4. In total, 87 unique objects were used for the training. Each object was rotated into 16 different views, resulting in 1392 training samples. A set of objects not used to create the training dataset were used to test the performance—the holdout set. Throughout the Chapter, three datasets created from the holdout set are used for the experiments:

- simulation dataset:
  - created using the procedure from Section 3.4;

- 35 objects in total;
  - some nonsymmetric objects are included in the dataset twice, but with different rotations;

- real dataset - reconstruction:
  - 10 objects from *YCB* dataset with different properties;
    - selected to contain objects in the range from symmetric and flat (boxes) to more complex (power drill);
  - can be seen in Fig. 4.2;

- real dataset - grasping:
  - two of the objects from the reconstruction dataset had to be changed, because all of their dimensions were bigger than the gripper;
    - we decided to do grasping after the reconstruction experiments were done, so the two objects were changed instead of doing everything again.

Note that the objects were firmly attached to the table with double-sided tape during all touch experiments. Allowing the objects to move would increase the complexity of the task. The pose of the object is found from the visual input from the camera on the table and we assume that it is not changing during the run of the pipeline.

6.1 Haptic Exploration with the 3D Printed Finger

This experiment should show how our method Act-VH performs in terms of reconstruction accuracy, *i.e.*, how good is the reconstruction from CIGR. We started with simulated experiments. Each of the 35 holdout objects were inserted into the simulation and the iterative shape completion with Act-VH was performed. Shape completion of each object was performed three times, *i.e.*, 105 runs of the pipeline were performed. Mean is computed over the samples, as using, for example, median is not statistically stable when only three samples are available. In addition, the same was performed with random and GPIS policy. In total, 315 iterative shape completions were performed. All point clouds were also reconstructed with the remaining reconstruction baselines (listed in Section 4.8). The experiments were also conducted in the real world, now with 10 objects from YCB dataset—30 runs of the pipeline were made. The real world experiments were performed only with the best policy from the simulation experiments, *i.e.*, our proposed method Act-VH. Five touches were performed during each run of the pipeline. This number was selected because it is high enough to see the trends, but it still takes a reasonable time—about five minutes for one object. For selected objects, experiments with 50 touches were performed and the results are described later. However, before showing the results in detail, the reasons behind some choices in the pipeline will be explained.

An example of how the reconstructions are changing during the iterative refining for an object with poor visual-only reconstruction can be seen in Fig. 6.2. Our method uses random sampling of points for mini-batches (upper and middle rows). However, one can see that the method struggles with fitting the input point cloud to meshes and often extends the input. We think that the problem is in using random mini-batches for the gradient

descent optimization, as the network does not have the complete information about the object at any given time. Yet, the mini-batches are needed because using all points at once is demanding for memory and computational resources. In addition, the mini-batches help to introduce the stochasticity used for uncertainty computation. The answer for this may be using Farthest Point Sampling (FPS) as, for example, in [70].

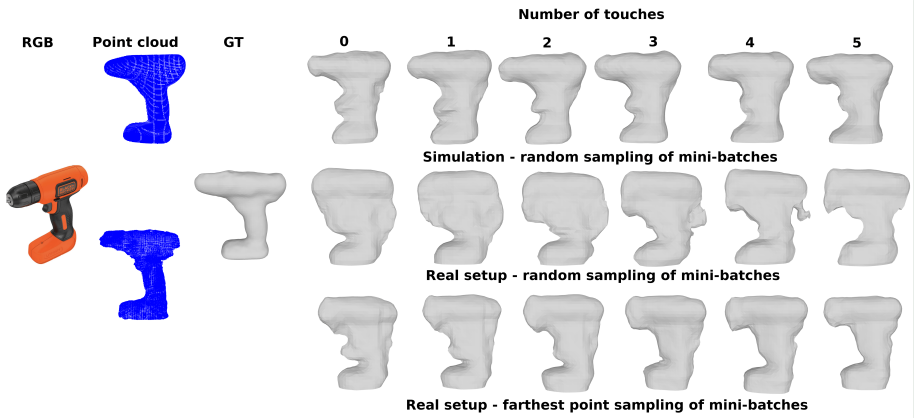
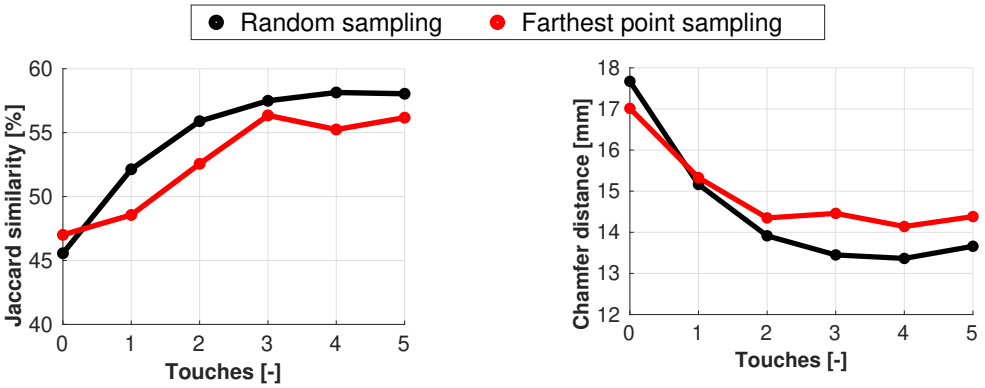


Figure 6.2: An example reconstruction of an object after each touch with Act-VH in the simulation (upper row), in the real world with random sampling of mini-batches (middle row) and in the real world with FPS of mini-batches (bottom-row). From Rustler *et al.* [1].

The mini-batches created with FPS give the network more comprehensive information. Reconstructions in the bottom row of Fig. 6.2 show that using this sampling, the meshes are more tightly fitted to the input point cloud. However, as can be seen in Fig. 6.3 where reconstructions on the real world dataset are evaluated, the overall performance is worse with FPS. We think, that the reason is that FPS is not leveraging the new information added by haptic exploration well enough. The optimal approach is probably to combine random sampling and FPS. However, we plan to explore that possibility in the future and we used only the random one in this work.

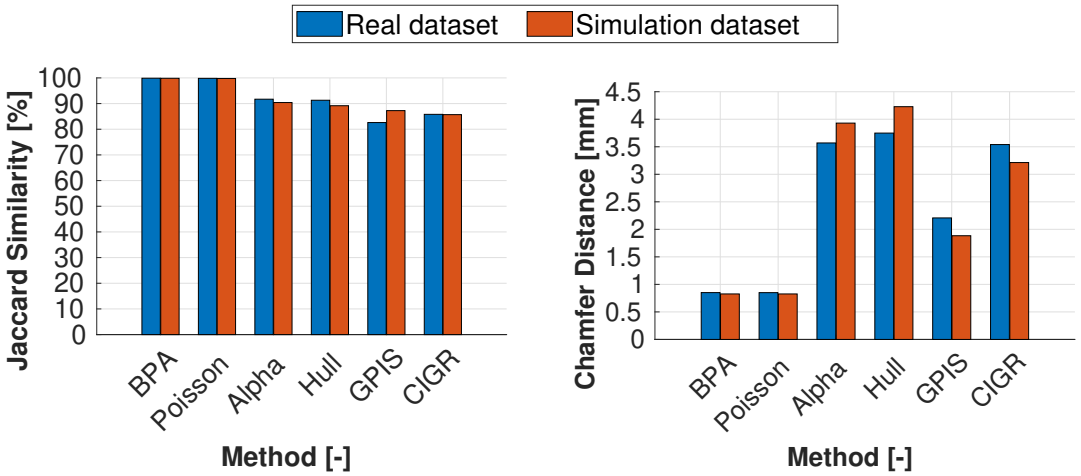


(a) : Jaccard similarity (JS). The higher, the better.

(b) : Chamfer distance (CD). The lower, the better.

Figure 6.3: Reconstructions evaluation for experiments with the 3D printed finger in the real world. Black lines correspond to random sampling of mini-batches and red lines correspond to FPS. Each value is mean over 30 reconstructions (10 objects with three repetitions). From Rustler *et al.* [1].

Even though this work is focused on the shape completion of incomplete point clouds, it is beneficial to know how the particular methods for reconstruction perform on whole ground truth meshes—one can see if the implementation of the method is correct and also it gives a starting point for what values to expect. We evaluated the methods on objects from both real and simulated datasets, and the results can be seen in Fig. 6.4. We can see that the methods from no estimation group (BPA, Poisson) score almost 100% Jaccard similarity (JS) and Chamfer distance (CD) in under 1 mm. It is not surprising because both are state-of-the-art methods for reconstruction on complete point clouds. The methods from the partial estimation group (Alpha, Hull) obtain about 90% of JS and circa 4 mm CD. The drop in performance is caused by the objects with a lot of details (for example power drill) for which the reconstruction is not precise. GPIS and CIGR score almost the same in JS (about 85%), but GPIS is better in terms of CD (circa 1.5 mm difference). However, both have smaller CD than Hull and Alpha methods. It is because the partial estimation methods can not preserve details. The reason why CIGR performs worse than other methods on complete point clouds is that all the objects are from the holdout set. In the case of GPIS, the performance is highly affected by its parameters. We started with values from [30] and ended up with: distance offset $d = 0.01$, noise parameter $s = 0.001$, voxel grid resolution $n = 40$ and the point clouds are down sampled to size $M = 300$. The performance can be boosted by down sampling to a higher number of points, but $M = 300$ was selected for its speed/quality ratio. For example, increasing M from 300 to 500 would increase JS from 87% to 92% (for the simulation dataset), but the time for each object would increase from 30-60 s (depending on object) to 300-600 s.



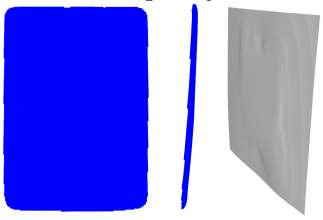
(a) : Jaccard similarity (JS). The higher, the better.

(b) : Chamfer distance (CD). The lower, the better.

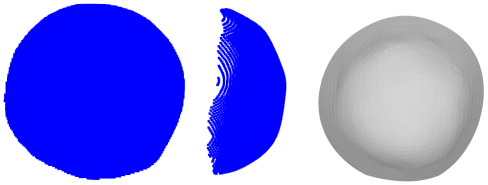
Figure 6.4: The performance of the methods for reconstruction evaluated on ground truth point clouds. Values are mean over all 35 objects for the simulation dataset and 10 objects for the real world dataset.

Two visual-only reconstructions with GPIS can be seen in Fig. 6.5, where you can see the difference when only a front side is visible (Fig. 6.5a) and when the object is small and the camera can see its top (Fig. 6.5b). We can see that GPIS is unable to reconstruct objects with only the front view. The reconstruction is almost flat, just slightly bent around the object. That leads to a situation when the uncertainty would propose to explore the object from the already seen front side. In our case, it was solved by performing the first touch

with a strict heuristic of touching the back of the object. That allowed the reconstruction to wrap around the object and propose touches from different sides. For that reason, values for experiments with policy from GPIS start at touch one in the upcoming graphs.



(a) : Front and side view of the captured point cloud and GPIS reconstruction for a flat point cloud.

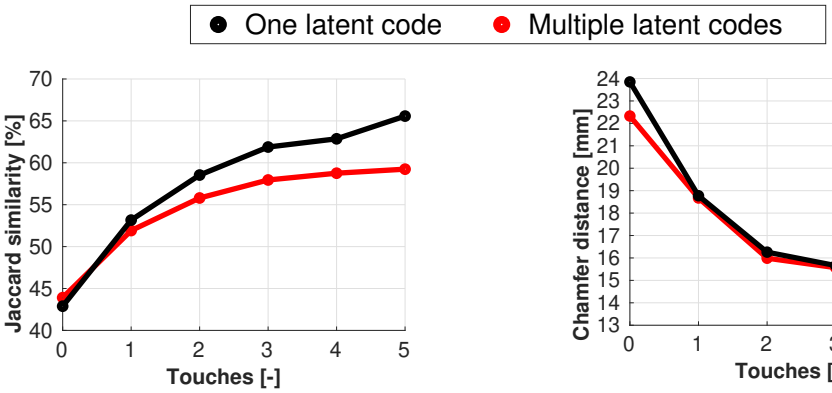


(b) : Front and side view of the captured point cloud and GPIS reconstruction for smaller object with more information from visual input.

Figure 6.5: Example of reconstruction with GPIS from visual only input. Difference between an object which has only a flat point cloud captured from the camera (a) and a smaller object for which the top and side information can be obtained (b).

6.1.1 Simulation Experiments

The initial experiment shows the difference between the sampling methods described in Section 5.1. As can be seen in Fig. 6.6 and have been already said, using only one latent code and sampling intermediate samples from it performs better than sampling multiple latent codes. The difference is visible mainly with increasing number of touches, as the multiple-codes method becomes almost static, without a notable increase in performance. This is mainly the result of the small number of uncertain voxels found. From 105 runs of the pipeline (35 objects with three repetitions each), the method was unable to perform all five touches in 29 cases, as there were no reachable uncertain voxels found. Note that in cases of less than five touches, the last value was propagated, so a consistent mean of the values can be computed. For example, Jaccard similarity for five touches (plus visual-only evaluation) of [36.43, 50.50, 55.90, nan, nan, nan] is considered to be [36.43, 50.50, 55.90, 55.90, 55.90, 55.90].

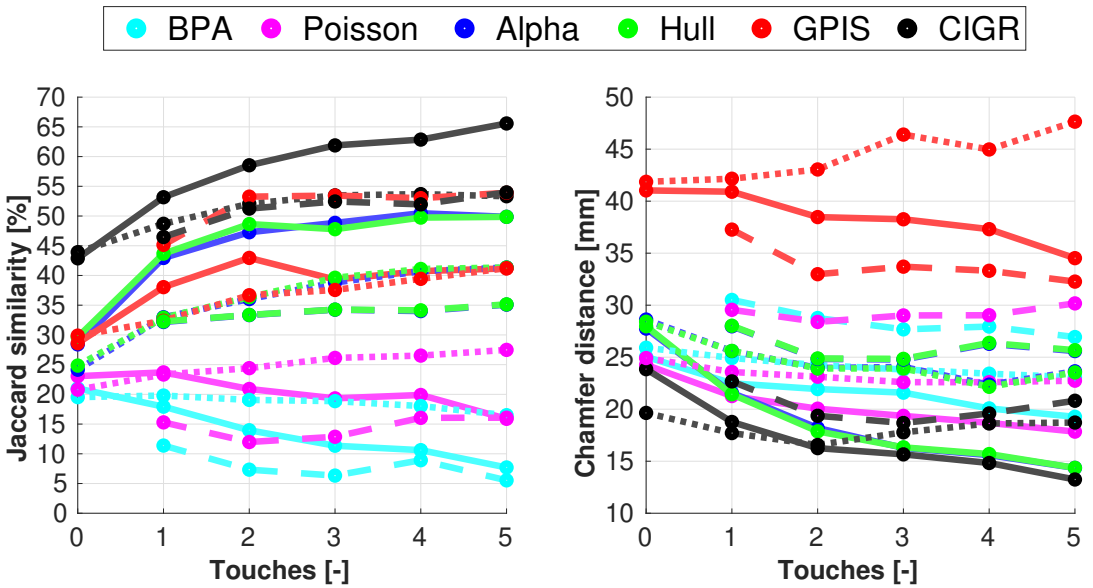


(a) : Jaccard similarity (JS). The higher, the better.

(b) : Chamfer distance (CD). The lower, the better.

Figure 6.6: Reconstructions evaluation for experiments with the 3D printed finger in the simulation. Black lines correspond to uncertainty computation with samples from one latent code (our selected method) and red lines are sampling of multiple latent codes. Each value is mean over 105 reconstructions (35 objects with three repetitions). From Rustler *et al.* [1].

Results of runs with five touches can be seen in Fig. 6.7. The graphs include values for all policies distinguished by the line type—solid line represents Act-VH policy, dotted line is for random touches and dashed line shows GPIS policy results. The colors of the lines mark which method was used to reconstruct the mesh. The Figures provide a lot of insight. However, firstly, one needs to observe both JS (the higher, the better) and CD (the lower, the better) as a whole. Because the methods have their pros and cons and can vary for some approaches, *e.g.*, point clouds from Act-VH reconstructed by Hull (solid green) score almost similar CD as reconstruction from CIGR (solid black), but the difference in JS is huge (about 15%).



(a) : Jaccard similarity (JS). The higher, the better.

(b) : Chamfer distance (CD). The lower, the better.

Figure 6.7: Reconstructions evaluation for experiments with the 3D printed finger in the simulation. The legend says by which method the meshes were reconstructed and the type of line marks which policy was used to guide the haptic exploration: solid is Act-VH, dotted is random and dashed is GPIS. Each value is mean over 105 reconstructions (35 objects with three repetitions). From Rustler *et al.* [1].

Nevertheless, Act-VH with CIGR (black) outperforms the other approaches in both reconstruction accuracy and haptic exploration effectiveness. In terms of reconstruction when Act-VH uncertainty policy is used, the Hull method (green) is the second best. The difference in JS is big (65% for CIGR and 50% for Hull), but in terms of CD the results are similar (13.2 mm for CIGR and 14.3 mm for Hull). The reason may be that Hull creates more sharp objects, which influences the Chamfer distance. However, in the case of using random policy (dotted) or GPIS policy (dashed), CIGR outperforms the other reconstruction baselines by a larger margin. For the random policy, Hull is again the second best. The difference in JS is now 12% (53% versus 41%) and difference in CD is 5 mm (18.7 mm versus 23.6 mm). The difference in CD shows that Hull is not efficient in estimating the shape when the touches are poorly distributed over the surface. The method creates "slopes" from one touch to another and when the touches are not distributed over the whole surface, there may be a large part of the objects missing. Examples of the

"slopes" can be found in Fig. 6.1. When using GPIS policy, the results are not that clear. For JS, reconstruction from CIGR and GPIS (red) score almost the same—approximately 54% both. It shows that GPIS performs better when the touches are made based on its policy. On the other hand, it also shows that CIGR can adjust well even to the situation when haptic exploration is not done based on its uncertainty. In the case of CD, CIGR performs the best with 20.8 mm followed by Hull with 25.7 mm. GPIS is the worst here with 32.3 mm, even when using its own exploration policy. The reason behind that is the behavior of creating unbound meshes—the same reason why we had to add the first touch with a heuristic, as described earlier. The method can fit to the points well when they are distributed evenly over the whole surface (as results on ground truth point clouds show in Fig. 6.4), but when some area of the object is unexplored, it fails. As can be seen in Fig. 6.1 or Fig. 6.5b, the mesh is "opened" to a space. It has again almost negligible influence on JS, but huge impact on CD. The same results can be seen in [30], even though the authors used more than 20 touches. Overall, we can acknowledge that CIGR reconstructs the meshes the best no matter what is the policy used to compute haptic exploration positions.

Comparing the exploration policies, the results are clearly in favor of Act-VH. This policy scored—with the best reconstruction method—65% JS and 13.2 mm with CD. It is a major difference for both random (53%, 18.7 mm) and GPIS (54%, 20.8 mm) policy. A thing to notice is that for random and GPIS policy, the evaluation metrics with all reconstruction methods are getting just slightly better with increasing number of touches, or even getting worse. The reason is that the suggested positions to be explored are not providing enough new information and are not robust enough to prevent errors coming from robotic movement and collision detection, which cause the performance to decrease. On the other hand, with Act-VH policy, the reconstructions are getting better with all reconstruction methods with a visible margin. The only exception is JS for BPA and Hull, when the probable reason is that the methods add surface of the mesh only on places with any points (see Fig. 6.1 for examples). And if the touches are far from the captured point cloud from the camera (which is usually the case for Act-VH, but not for the other policies), the creation of voxel grids is error prone. Eventually, we can say that Act-VH uncertainty policy provides positions for haptic exploration which can add useful information. The positions are also more robust than for the other policies.

Even though we decided to perform only five touches, we also evaluated an experiment with 50 touches. Three objects were randomly selected and evaluated—again with three repetitions for each. The averaged values can be seen in Fig. 6.8. We can see that the performance is getting better for about 15 touches. After that, new touches are not providing enough information to improve the reconstruction. Eventually, both JS and CD start to get worse. We assume that noise and errors—mainly in collision detection—are the main causes here. Even in the simulation, the detection of contact using joint torques is error prone. It sometimes stops too late, *e.g.*, when the impact is not direct and the force is distributed in multiple axes and joints. Or, on the contrary, the movement of the robot may be jerky in some poses and cause false positive detection. With increasing number of touches, these errors have a higher probability to happen and small errors get summed up.

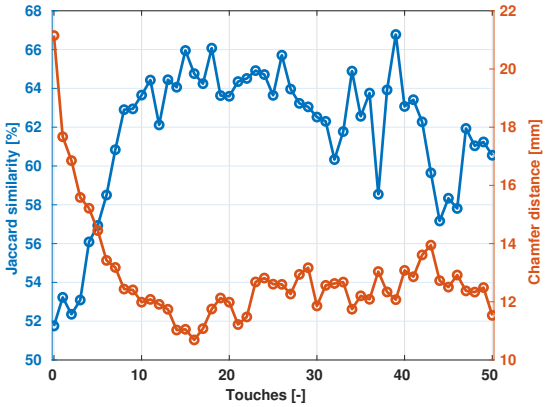
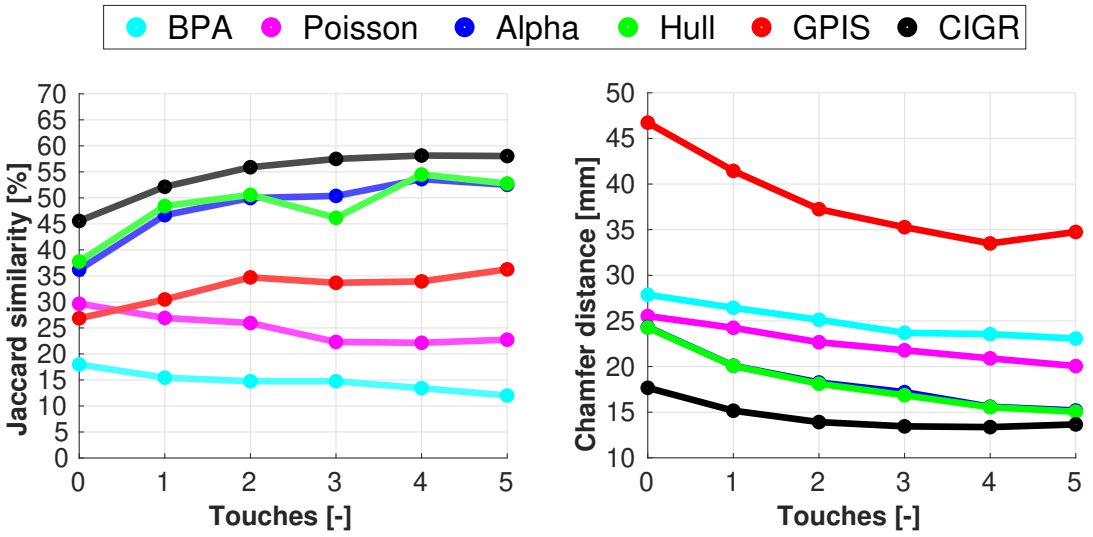


Figure 6.8: Average Jaccard similarity (JS) (blue) and Chamfer distance (CD) (red) for three objects over three repetitions from Act-VH in the simulation. From Rustler *et al.* [1].

6.1.2 Real World Experiments

After the simulation experiments, we switched to the real world setup. The code is the same as for the simulation. Only the collision detection threshold had to be changed. Before the experiments, both intrinsic and extrinsic parameters of the camera were calibrated. The number of touches (five) and the number of repetitions (three) were kept the same. As the previous section showed, Act-VH is the best policy for haptic exploration. Therefore, the other policies were not tested in the real world setup.



(a) : Jaccard similarity (JS). The higher, the better.

(b) : Chamfer distance (CD). The lower, the better.

Figure 6.9: Reconstructions evaluation for experiments with the 3D printed finger in the real world. The legend says by which method the meshes were reconstructed. Each value is mean over 30 reconstructions (10 objects with three repetitions). From Rustler *et al.* [1].

The results are in Fig. 6.9. The relative performances of each reconstruction method are the same as for the simulation. The best is CIGR followed by Hull. The difference is in

the achieved values for the evaluation metric. Our method scored 7% less JS compared with the simulation (58% versus 65%) and 0.4 mm higher CD (13.6 mm versus 13.2 mm). Furthermore, the increase in performance between touches is getting smaller with each new touch and for CD the performance even decreases after the fifth touch. It is probably caused by bigger noise in the real world. The point cloud from the camera is much more noisy, *e.g.*, it can contain holes or points reflected from a shiny surface. Other source of noise is coming from collision detection, even more than in the simulation. There are multiple reasons: the computation of dynamic torques is less precise because a dynamics simulator is used for that; the objects are not as stiff as in the simulation; and the finger is not as stiff as in the simulation, so it can bend a little and absorb some force. Last but not least, the encoders of the real robot are not perfectly precise, so the impact position computed from kinematics is noisy. Overall, the Act-VH works even in the real world, yet it would need more measures to mitigate the effects of noise.

6.2 Haptic Exploration with the Gripper

We decided to test the pipeline also without the 3D printed finger. The upcoming experiments have the same settings as in previous Section 6.1.2. The end-effector is now set to the middle of the closed gripper. The closed gripper is much wider than the printed finger, so there is a bigger probability to hit the object with other parts than the center, or to slide over the edges without detecting collision. From the experiences gained in the previous experiments, we decided to always collide with the gripper rotated horizontally to avoid sliding over the object surface—as can be seen in Fig. 6.10.

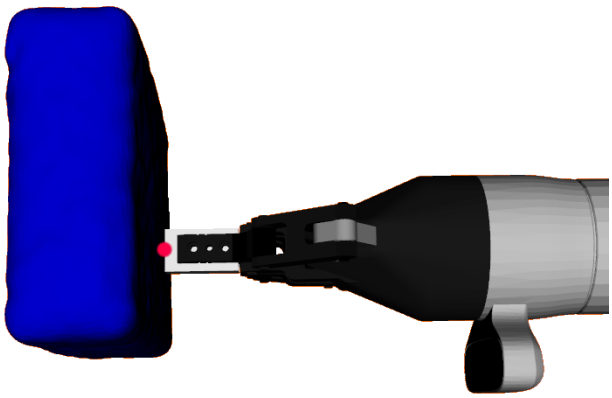
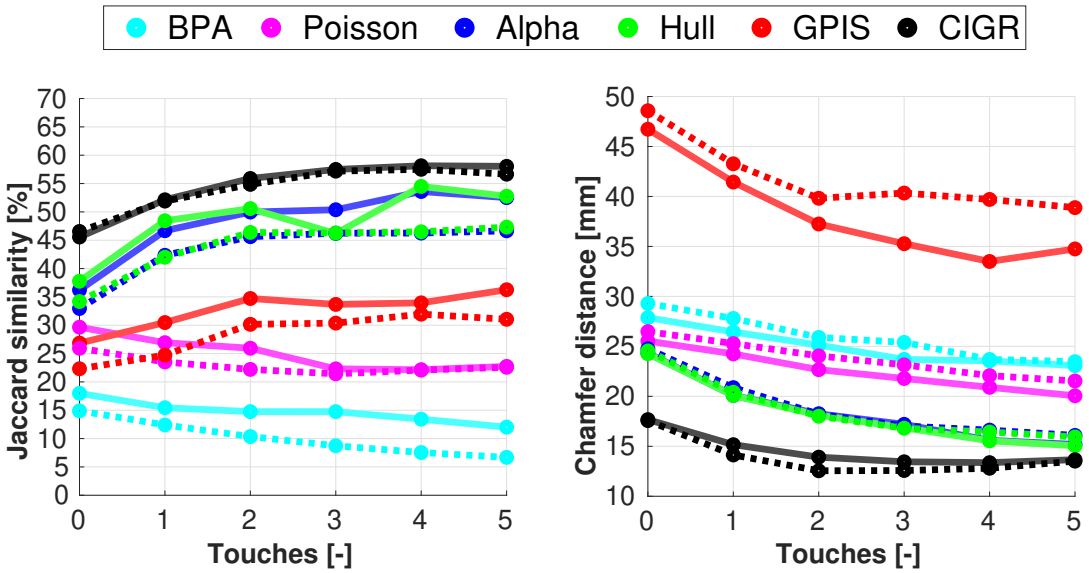


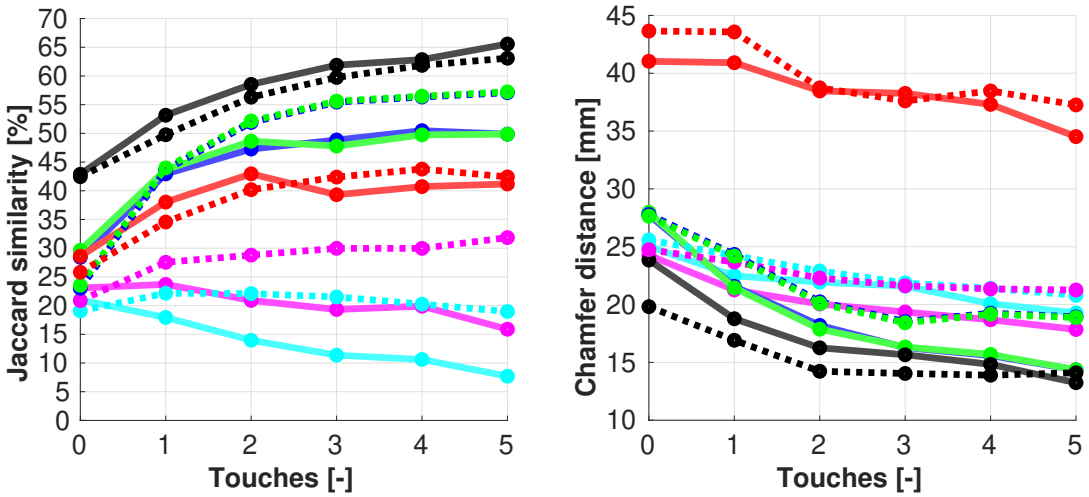
Figure 6.10: Top view of the pose of the gripper used for touching. Pose rotated 180° about the last joint was also allowed. The position of the end-effector is marked with the red circle.

The results are shown in Fig. 6.11. We can see that the experiments with the gripper (dashed) almost copy the experiments with the finger (solid) when reconstructing using CIGR. For the other methods, exploring with the gripper in the simulation performs even better than with the finger. It is not the case in the real world, but the decrease in

performance is slight. Again, we assume that it is caused by the collision detection. Overall, when comparing results for CIGR, which is again the best, exploring with the gripper is slightly worse, but the difference is almost negligible (see Table 6.1 for exact values).



(a) : Real world setup.



(b) : Simulation.

Figure 6.11: Reconstruction evaluation for experiments with the gripper (dashed) and with the finger (solid) for Jaccard similarity (JS) (left) and Chamfer distance (CD) (right) for both real world setup (a) and simulation (b). The legend says by which method the meshes were reconstructed. Each value is mean over 105 reconstructions (35 objects with three repetitions) for the simulation experiments and mean over 30 reconstructions (10 objects with three repetitions) for the real-world experiments.

Even after using a fixed rotation, exploration with the gripper is challenging because of its width. However, there is lower noise from collision detection, as the gripper is closer to the joints and is more stiff, so the force propagation to the joints is much more effective

than in the case of the 3D printed finger. In conclusion, we can say that the results for poking with the finger and with the gripper are almost the same and using the gripper is the preferred possibility as it is much more compact and self-contained solution.

	JS sim [%]	JS real [%]	CD sim [mm]	CD real [mm]
Finger	65.57	58.04	13.24	13.66
Gripper	63.09	56.65	14.10	13.51

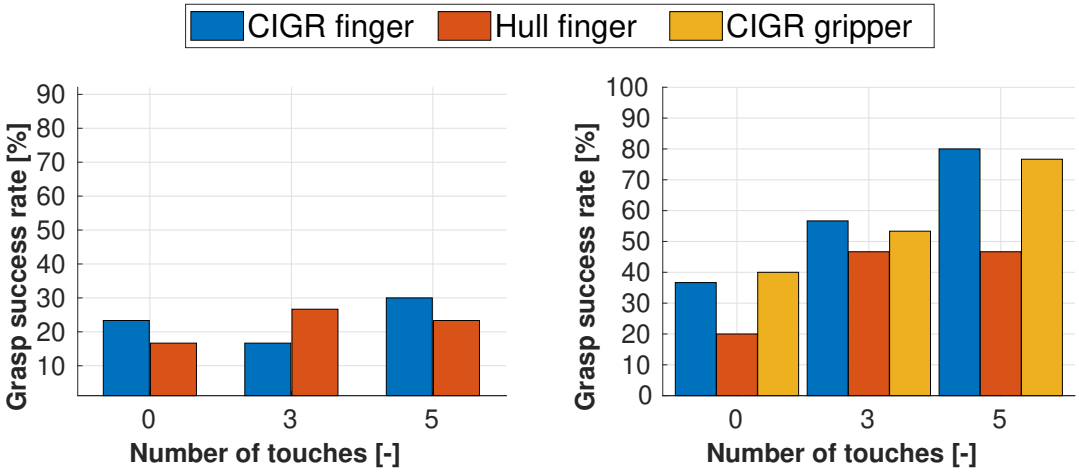
Table 6.1: Jaccard similarity (JS) and Chamfer distance (CD) after five touches for the experiments when the 3D printed finger and the gripper were used as an end-effector. Each value is mean over 105 reconstructions (35 objects with three repetitions) for the simulation experiments and mean over 30 reconstructions (10 objects with three repetitions) for the real-world experiments.

6.3 Grasping

In the previous sections, we discussed the results based on the evaluation metrics. These metrics are helpful to discover whether the selected approach has the potential to be successful in shape completion. However, one of the main possible uses of shape completion is grasping. Grasp success rate was evaluated on 10 objects. The grasping was done accordingly to Algorithm 4. For each object, the reconstruction pipeline was run once with five touches and then the grasping was done three times. We wanted to use the same 10 objects as for the reconstruction experiments, but we realized that two of them have no parts which could fit to the gripper, so we had to replace these. We marked the positions of all objects in the workspace, so we were able to return them to the original position after an unsuccessful grasp. The grasping experiments were done only in the real world. Grasp success rates after zero, three, and five touches are provided.

Because Hull method achieved the second best result in the reconstruction accuracy in the previous experiments, it was selected as a baseline in the grasping experiments. We decided to use just one baseline, because the experiments were time consuming and we also wanted to reduce the damage of paper items caused by grasping. Grasp success rate can be seen in Fig. 6.12. The first grasps were done before the reconstruction experiments with the gripper, so Hull is evaluated on point clouds collected with the finger. The intention was to run the pipeline to gather point clouds and run the grasping while the finger was still mounted on the robot. However, it appeared to be impossible with GraspIt!. When the finger was not included in GraspIt! when sampling grasps, almost no grasp was feasible for the robot because the finger would collide with the objects. However, when the finger was added to the GraspIt! model, the planner considered it as part of the gripper and tried to minimize the distance to it. That resulted in unusable grasps, *e.g.*, the finger was close the the object, but the gripper itself was rotated away from the object and closed empty. Results from this experiments can be found in Fig. 6.12a. We can see that there is almost no difference between CIGR and Hull as both performed poorly. The maximal success rate was for CIGR after five touches scoring 30%, *i.e.*, only 10 out of 30 grasps were successful. To overcome this issue, the finger was unmounted from the robot and new grasps were computed, still using the reconstructions from exploration with the finger. Success of the experiments is depicted in Fig. 6.12b. We can see that now CIGR achieved success rate of

36% with zero touches, which is more than after five touches in the previous experiment. On reconstructions after five touches CIGR got to 80%, which is more than double the improvement over the visual-only reconstruction. Hull was able to get only 46% after five touches, but it also improved from reconstruction without haptic exploration (20%). The same graph includes results for grasping from exploration with the gripper. Note that this experiments were done at different time with different light conditions, which could result in different initial point clouds. However, the trend is similar to the finger experiments. The grasp success rate increased from 40% after zero touches to 76% after five touches. In absolute numbers, it is an increase from 12 to 23 successful grasps, which is analogous to the increase from 11 to 24 for exploration with the finger.



(a) : Mean grasp success rate over 30 grasps (10 objects with three repetitions) with finger mounted during the grasp phase.

(b) : Mean grasp success rate over 30 grasps (10 objects with three repetitions) with finger unmounted during the grasp phase.

Figure 6.12: Grasp success rates for Hull (point clouds collected with Act-VH with 3D printed finger) and CIGR (point clouds collected with both possibilities of Act-VH—with 3D printed finger and with gripper). From Rustler *et al.* [1].

Fig. 6.13 provides evaluation over individual objects. We can see that CIGR for both exploration possibilities grasped 9 out of 10 objects at least 2 times from 3. Exploration with the gripper failed on a pitcher, where it is crucial to identify the handle to be able to find a grasp. Exploration with the finger failed with a long and oval chip can. From what we saw, the gripper was too far from the center of the can, so the can was "slipping out" while grasping. On the other hand, Hull grasped for zero or one time on the majority of objects. Rectangular objects seem to be troublesome for this method. It is probably coming from the already mentioned "slopes" created by the method. Example of one rectangular object can be seen in Fig. 6.1. Because of these slopes, the fingers of the gripper in the estimated grasp are not parallel to the real surface of the objects, and thus the grasps are unstable.

Overall, the policy based on uncertainty from Act-VH helps even the baseline to improve with additional touches. However, reconstructing with CIGR is superior over the baseline with a notable difference. It is an interesting revelation, because the methods performed almost the same in mesh evaluation metrics.

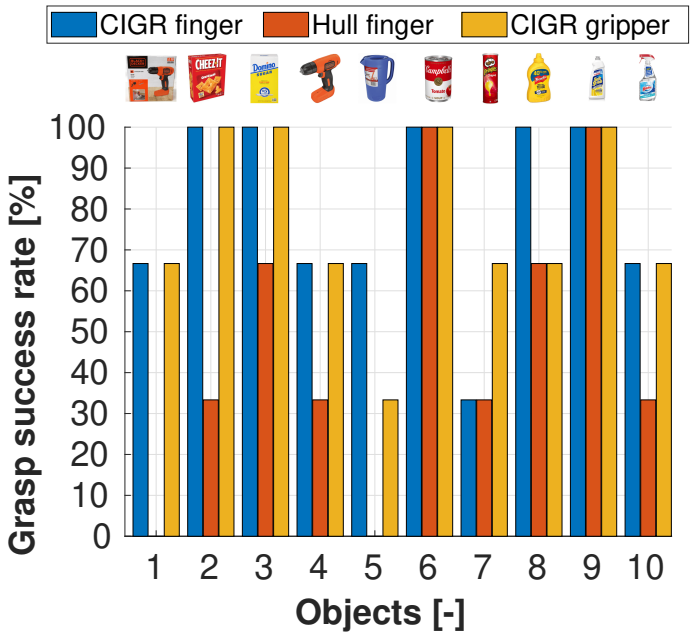
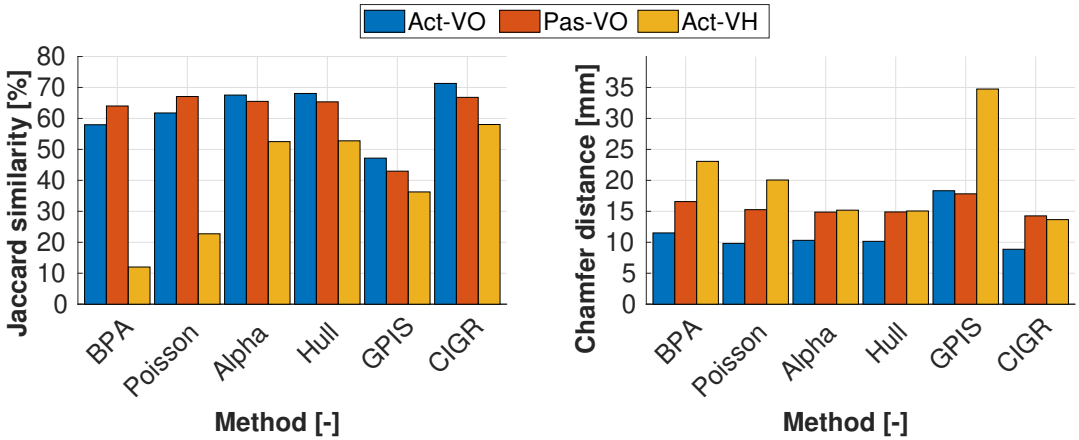


Figure 6.13: Mean grasp success rate over three grasps for each object for Hull (point clouds collected with Act-VH with 3D printed finger) and CIGR (point clouds collected with both possibilities of Act-VH—with 3D printed finger and with gripper). From Rustler *et al.* [1].

6.4 Visual-only Approaches Experiments

The last set of experiments are visual-only approaches (only camera is used; described in Section 5.3). Again, three repetitions were made for each object. In the case of uncertainty-driven approach, five views were used. Firstly, we compare the visual-only approaches and then compare them with Act-VH with the gripper. Visual-only experiments were performed only in the real world.

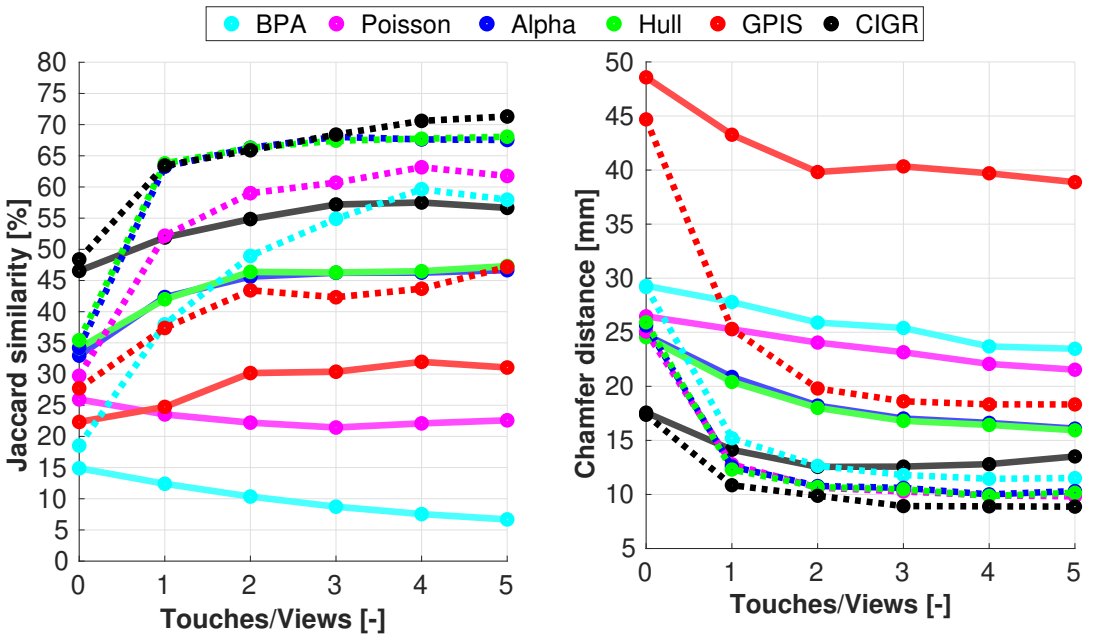


(a) : Jaccard similarity (JS). The higher, the better.

(b) : Chamfer distance (CD). The lower, the better.

Figure 6.14: Reconstruction evaluation for Act-VO (blue), Pas-VO (red) and Act-VH with gripper (orange) in the real world. Each value is mean over 30 reconstructions (10 objects with three repetitions). Method names on x-axis stand for the mesh reconstruction methods.

In Fig. 6.14 one can see JS and CD values for Passive visual-only shape completion (Pas-VO) approach, fifth touch of Act-VH with gripper and fifth view of Active visual-only shape completion (Act-VO). When comparing only the visual-only approaches, we can see that in case of JS, Pas-VO performs slightly better, except for BPA and Poisson. The best results are obtained by CIGR reconstruction, which scored 71% for Act-VO and 67% for Pas-VO. For CD, the differences are much more notable. CIGR reconstruction achieved CD 8.9 mm with Act-VO and 14.3 mm for the other one. This comparison clearly shows that Act-VO is better. However, it also takes more time. Pas-VO can be done in less than a minute and the uncertainty-driven one needs about three minutes for five views. Both of the methods are better than Act-VH in case of JS for all reconstruction baselines. However, interestingly, visuo-haptic approach for some reconstruction methods performs the same or even better than the Pas-VO. We assume that may be caused by the high uncertainty of Pas-VO. The method does not have any information about the object except the initial point cloud from the table camera and so it may not see the whole object. Or, on the contrary, it can see a lot of background (for smaller objects), which is more error prone to depth image errors caused by, for example, reflections from the table.



(a) : Jaccard similarity (JS). The higher, the better.

(b) : Chamfer distance (CD). The lower, the better.

Figure 6.15: Reconstruction evaluation for Act-VH with the gripper (solid) and Act-VO (dashed). The legend says by which method the meshes were reconstructed. Each value is mean over 30 reconstructions (10 objects with three repetitions).

Values for all five touches/views of the visuo-haptic approach and uncertainty-driven visual-only completion can be seen in Fig. 6.15. It confirms the results from the previous graph. Yet, it offers interesting information about the influence of additional views. We can see that the biggest improvement is gained with the first two views and then the slopes of the lines are almost flat. Overall, the visual-only approach is better in both JS (58% vs 71%) and CD (8.9 mm vs 13.5 mm). However, we have to keep in mind that the visual-only

approach can gain several times more information than the visuo-haptic one.

However, all objects tested were nontransparent and "friendly" for the camera. Fig. 6.16 shows an example of reconstruction of adversarial transparent object from Act-VH (reconstructed with CIGR; reconstructions with other baselines can be seen in Fig. 6.1). We can see that the visual-only methods fail to even coarsely estimate the object shape. The main problem is with the thin "neck" of the bottle. The depth sensor sees through it and estimates the depth when the sticker is seen. That creates a problem with inconsistent normals, which are essential for most mesh reconstruction methods. On the other hand, the haptic exploration can add new information properly and the resulting reconstruction is better, even though still not perfect. Note that this object was not part of the analytical evaluation, because no ground truth is available.

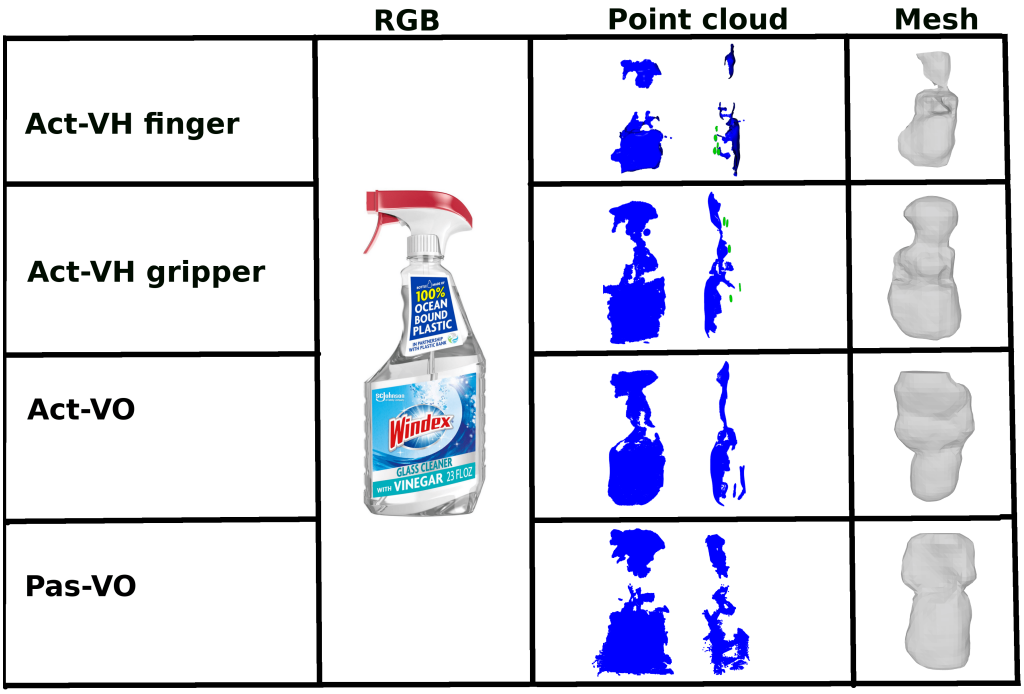


Figure 6.16: Reconstructions of an adversarial transparent object. All the meshes are from the real setup and reconstructed with CIGR after five touches/views. Touches are highlighted in the point clouds with green color.

6.5 Summary

To summarize the experiments, Table 6.2 and Table 6.3 with exact values are provided. The Tables include absolute values after five touches/views and relative improvement from zero to five touches—Pas-VO is not iterative, so no improvement can be computed. Columns are divided by different reconstruction methods (CIGR and baselines described in Section 4.8). Rows contain experiments with different policies for uncertainty computation (Act-VH, random, GPIS), different worlds (simulation, real world) and different shape completion methods (visual-only, visuo-haptic).

	Jaccard similarity [%]					
	BPA	**Poisson**	**Hull**	**Alpha**	**GPIS**	**CIGR**
Act-VH finger (real)	12.01	22.74	52.51	52.79	36.26	58.04
Act-VH finger (sim)	7.69	15.88	49.86	49.86	41.18	65.57
Act-VH gripper (real)	6.69	22.58	46.65	47.31	31.05	56.65
Act-VH gripper (sim)	18.97	31.84	57.08	57.25	42.42	63.09
GPIS (sim)	5.53	16.07	35.09	35.14	53.90	53.97
Random (sim)	16.49	27.48	41.33	41.34	41.29	53.32
Act-VO (real)	57.96	61.76	67.54	68.06	47.17	71.31
Pas-VO (real)	64.02	67.07	65.52	65.35	42.96	66.81

(a) : Absolute values after five touches/views.

	Relative improvement [%]					
	BPA	**Poisson**	**Hull**	**Alpha**	**GPIS**	**CIGR**
Act-VH finger (real)	-33.19	-23.32	45.01	39.74	35.04	27.38
Act-VH finger (sim)	-63.33	-31.13	75.52	68.19	43.97	52.90
Act-VH gripper (real)	-55.10	-12.93	41.58	38.41	39.10	21.70
Act-VH gripper (sim)	-0.54	52.62	147.78	143.15	64.36	48.64
GPIS (sim)	-51.39	5.15	9.07	8.92	19.40	16.16
Random (sim)	-15.56	32.17	71.19	66.43	37.98	21.42
Act-VO (real)	212.95	107.85	97.43	92.10	70.18	47.37
Pas-VO (real)	0.00	0.00	0.00	0.00	0.00	0.00

(b) : Relative improvement from zero to five touches. Pas-VO is not iterative, so no improvement can be computed.

Table 6.2: Jaccard similarity (JS) (the higher, the better). Rows are different policies for visuo-haptic or visual-only exploration (the last two rows). Columns are different reconstruction methods. Each value is mean over 105 reconstructions (35 objects with three repetitions) for the simulation experiments and mean over 30 reconstructions (10 objects with three repetitions) for the real-world experiments. Visual-only approaches (Act-VO, Pas-VO) were performed only in the real world.

The absolute values recapitulate what we have already seen. In terms of reconstruction methods, CIGR achieves the best results on all point clouds. When comparing the efficiency of visuo-haptic strategies, Act-VH is superior over both random and GPIS policies. Act-VH scores over 10% higher JS and more than 5 mm lower CD. The benefits of Act-VH are also visible from the relative improvements. For CD, the improvements using Act-VH (in both worlds and with both poking end-effectors) are at least twice better than the other two approaches. In case of JS, the improvements are 4-5 times better than improvements with GPIS. The random policy have similar improvements as Act-VH. However, together with improvements of CD, the Act-VH is preferable. As has been said, the Act-VH fails in the improvements of BPA and Poisson for all visuo-haptic experiments. However, it is not important as these are not preferable shape completion methods. The visual-only methods confirm the previous statements in both absolute and relative cases. The uncertainty-driven visual-only method is the overall best. It obtained about 15% higher JS and 5 mm lower CD than Act-VH.

	Chamfer distance [mm]					
	BPA	**Poisson**	**Hull**	**Alpha**	**GPIS**	**CIGR**
Act-VH finger (real)	23.07	20.25	15.20	15.05	34.75	13.66
Act-VH finger (sim)	19.27	17.85	14.35	14.36	34.52	13.25
Act-VH gripper (real)	23.47	21.53	16.09	15.94	38.88	13.51
Act-VH gripper (sim)	20.80	21.26	18.94	18.87	37.26	14.10
GPIS (sim)	26.94	30.17	25.59	25.67	32.27	20.81
Random (sim)	23.07	22.74	23.64	23.53	47.64	18.72
Act-VO (real)	11.51	9.82	10.31	10.15	18.32	8.87
Pas-VO (real)	16.57	15.26	14.87	14.90	17,82	14.26

(a) : Absolute values after five touches/views.

	Relative improvement [%]					
	BPA	**Poisson**	**Hull**	**Alpha**	**GPIS**	**CIGR**
Act-VH finger (real)	17.24	21.46	37.56	37.97	25.64	22.70
Act-VH finger (sim)	23.02	26.45	48.27	48.66	15.87	44.46
Act-VH gripper (real)	19.87	15.86	34.99	35.07	19.96	23.08
Act-VH gripper (sim)	18.67	14.08	31.85	31.71	14.63	28.83
GPIS (sim)	11.70	-2.08	8.51	8.39	13.41	8.21
Random (sim)	10.96	8.73	17.34	17.15	-13.81	4.71
Act-VO (real)	60.55	60.79	59.73	60.83	58.99	48.91
Pas-VO (real)	0.00	0.00	0.00	0.00	0.00	0.00

(b) : Relative improvement from zero to five touches. Pas-VO is not iterative, so no improvement can be computed.

Table 6.3: Chamfer distance (CD) (the lower, the better). Rows are different policies for visuo-haptic or visual-only exploration (the last two rows). Columns are different reconstruction methods. Each value is mean over 105 reconstructions (35 objects with three repetitions) for the simulation experiments and mean over 30 reconstructions (10 objects with three repetitions) for the real-world experiments. Visual-only approaches (Act-VO, Pas-VO) were performed only in the real world.

Chapter 7

Discussion, Conclusion and Future Work

We presented Act-VH as a solution for uncertainty-driven visuo-haptic shape completion. The haptic exploration itself was tested in simulation and real world with a robotic manipulator using torque sensors to detect a collision. The method is based on state-of-the-art implicit surface Deep Neural Network (DNN). From various possibilities [5, 14, 15], the Implicit Geometric Regularization for Learning Shapes (IGR) was selected after a series of experiments. We trained the network from a dataset we created. The dataset consists of complete point clouds created from *YCB* [36] and *Grasp Database* [37]. For each training object, a latent vector is created. Each vector is a parametrization of an object space and the final object can be obtained from isosurface at level zero. The advantage is that only partial point clouds can be supplied during interference phase and the network is able to find a most suitable vector which parametrizes the current object space and obtains a complete mesh from it. The other advantage is that the older methods needed to be trained from a enormous number of views [2–4, 12, 13], which is not the case here. Also, classical Convolutional Neural Network (CNN) needed to be trained with simulated haptic data to be able to use it [30]. For IGR, one can just add the haptic data to the point cloud without any retraining. However, as showed in Fig. 6.4, the method is outperformed with state-of-the-art mesh reconstruction methods when the full point cloud is supplied. In future work, a more diverse dataset should be created to combat this issue. A more diverse dataset could fix also problems with visual-only reconstructions. The Fig. 6.2 shows that without any haptic exploration, our method fails to give a reasonable estimate for unknown objects when only one view is available. We showed that replacing random sampling of mini-batches used in gradient descent with Farthest Point Sampling (FPS) can help to better fit the point cloud. However, using only FPS is worse in overall and in the future we would like to introduced some mixed sampling approach. In addition, we would also like to take advantage of moving through free space. For voxel grid-based CNN [13], one could easily set a voxel to unoccupied if there was nothing during the haptic exploration. It is not possible in our method and we think it may improve the reconstruction a lot.

The main contribution of this work is a novel uncertainty-driven exploration policy. Most of the state-of-the-art methods use random exploration or heuristic [2, 30, 31]. We provided a solution based on variance computed from probabilistic voxel grids of multiple samples. The uncertainty finds the most promising places to be explored to maximize the information gained. To effectively implement and test the algorithm, we used a simulation environment based on a physics-based simulator *MuJoCo* [44] and improved it for our case. The simulation contains the robotic arm and also a virtual camera providing point clouds. In addition, multiple benchmarking features were added to help the user with evaluating the

algorithm. The features can be also used in the real world, providing a real-time overview of the progress at any given time. In addition, we designed a custom 3D printed finger, which helped with more precise touches in the initial experiments. In addition to the finger, the gripper of the robot was also used for exploration. We were able to obtain similar results with the gripper even when it is much wider than the finger. We would like to improve the reconstruction with the gripper even more, because the 3D printed finger suffers from insufficient stiffness and placement errors. Moreover, using the gripper already installed on the robot is a more compact and self-contained solution. Thus, in the future, we want to focus on improving the gripper manipulation.

We compared our approach with a random exploration and with the uncertainty gained from Gaussian Process Implicit Surface (GPIS) [25]. The results on 10 objects (60 different reconstructions for each baseline reconstruction method) in the real world and 35 objects (over 400 different reconstructions for each baseline reconstruction method) in the simulation were described in Section 6.1 and Section 6.2. Act-VH outperforms all other exploration policies for visuo-haptic exploration with a notable margin. The same applies for visual-only methods, where the uncertainty-driven approach is superior to the approach with fixed positions. Additionally, reconstructions from point clouds collected with Act-VH are getting better with each touch even for the other reconstruction baselines. However, the visual-only approach is still better then Act-VH. Yet, one has to take into consideration that it is less common for robots to have a built-in camera. Furthermore, haptic exploration gains much less information every touch than the camera with a single view. Still, we were able to get the results with only 5 touches, while state-of-the-art methods usually use tens of them. However, our experiments also showed that with our method, the results may even get worse with increasing number of touches because of noise. In the future, we would like to try a different method for collision detection. Even though using torque sensors is a self-contained solution, it is also very noisy and imprecise. We are sure that if we use more sensitive contact detection sensors (tactile, for example), we would achieve even better results. A more sensitive sensor would also help with the necessity to tape the objects to the table. Moreover, as showed in Fig. 6.16, visual-only reconstruction can fail for some objects (transparent, for example). And that is where haptic exploration can help a lot.

Finally, we evaluated reconstructions from Act-VH in grasping experiments. We compared performance of our modified implementation of IGR (Shape Completing IGR (CIGR)) and Hull reconstruction method on point clouds collected with Act-VH with both the finger and the gripper on 30 grasps for each method—we performed 450 grasps in total. Both variants of Act-VH showed superior performance over the baseline, when CIGR reconstructions from Act-VH with the finger were grasped successfully in 80% of grasps (24 from 30) and Hull reconstructions were grasped only in 46% (14 from 30). CIGR from Act-VH with the gripper performed very similar with 76% (23 from 30) grasp success rate. In addition, the success rate after zero and five touches was more than doubled for all methods. However, when using the finger, it must have been unmounted for grasping. That is inconvenient and using only the gripper is the preferred approach for the following research.

We proposed a visuo-haptic approach. However, the visual input is taken only once, and new information is added with haptic exploration. As could be seen, the camera can add more information, but it can fail in some cases, *e.g.*, for transparent objects. In the

future, an improvement could be to enable the pipeline to decide whether it is better to add new information from touch or from the camera, *e.g.*, based on the object's material. Additionally, we suppose there is only one object in the scene, which is not always the case in real scenarios. Thus, other possible improvement would be to find all objects in the scene and shape complete them separately.

An accompanying video is here: `https://youtu.be/Ft1PUYRNFHw`. The simulation environment and the data from experiments pertaining to [1] are available at `https://github.com/ctu-vras/visuo-haptic-shape-completion`. Additional internal code is available in the attachment of this work. Data collected during the experiments (point clouds and ROSbags) can be downloaded from `https://drive.google.com/drive/folders/1Du8hVDbsFYqEvb-hwS6ISpnaE4RGJW8c?usp=sharing`.

Bibliography

[1] L. Rustler, J. Lundell, J. Behrens, V. Kyrki, and M. Hoffmann, "Active Visuo-Haptic Object Shape Completion," 2022, submitted.

[2] J. Varley, C. DeChant, A. Richardson, J. Ruales, and P. Allen, "Shape Completion Enabled Robotic Grasping," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 2442–2447.

[3] S. Wang, J. Wu, X. Sun, W. Yuan, W. T. Freeman, J. B. Tenenbaum, and E. H. Adelson, "3D Shape Perception from Monocular Vision, Touch, and Shape Priors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 1606–1613.

[4] M. Björkman, Y. Bekiroglu, V. Högman, and D. Kragic, "Enhancing Visual Perception of Shape through Tactile Glances," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 3180–3186.

[5] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman, "Implicit Geometric Regularization for Learning Shapes," in *International Conference on Machine Learning*. PMLR, Nov. 2020, pp. 3789–3799.

[6] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, "Revisiting active perception," *Autonomous Robots*, vol. 42, no. 2, pp. 177–196, 2018.

[7] J. Bohg, M. Johnson-Roberson, B. León, J. Felip, X. Gratal, N. Bergström, D. Kragic, and A. Morales, "Mind the gap-robotic grasping under incomplete observation," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 686–693.

[8] R. Schnabel, P. Degener, and R. Klein, "Completion and Reconstruction with Primitive Shapes," in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 503–512.

[9] M. Pauly, N. J. Mitra, J. Giesen, M. H. Gross, and L. J. Guibas, "Example-Based 3D Scan Completion," in *Symposium on Geometry Processing*, no. CONF, 2005, pp. 23–32.

[10] M. Li, K. Hang, D. Kragic, and A. Billard, "Dexterous Grasping Under Shape Uncertainty," *Robotics and Autonomous Systems*, vol. 75, pp. 352–364, 2016.

[11] A. Dai, C. R. Qi, and M. NieBner, "Shape Completion Using 3D-Encoder-Predictor CNNs and Shape Synthesis," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 6545–6554.

[12] X. Han, Z. Li, H. Huang, E. Kalogerakis, and Y. Yu, "High-Resolution Shape Completion Using Deep Neural Networks for Global Structure and Local Geometry Inference," in *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 85–93.

[13] J. Lundell, F. Verdoja, and V. Kyrki, "Robust Grasp Planning Over Uncertain Shape Completions," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 1526–1532.

[14] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, Jun. 2019, pp. 165–174.

[15] M. Atzmon and Y. Lipman, "SAL: Sign Agnostic Learning of Shapes From Raw Data," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, Jun. 2020, pp. 2562–2571.

[16] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[17] S. Kriegel, C. Rink, T. Bodenmüller, and M. Suppa, "Efficient Next-Best-Scan Planning for Autonomous 3D Surface Reconstruction of Unknown Objects," *Journal of Real-Time Image Processing*, vol. 10, 12 2013.

[18] R. Monica and J. Aleotti, "A Probabilistic Next Best View Planner for Depth Cameras Based on Deep Learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3529–3536, 2021.

[19] Z. Fei, X. Zhou, X. Gao, and G. Zhang, "A flexible 3d laser scanning system using a robotic arm," 06 2017, p. 103294U.

[20] D. Banerjee, K. Yu, and G. Aggarwal, "Robotic Arm Based 3D Reconstruction Test Automation," *IEEE Access*, vol. 6, pp. 7206 – 7213, 03 2018.

[21] S. Larsson and J. Kjellander, "Path Planning for Laser Scanning with an Industrial Robot," *Robotics and Autonomous Systems*, vol. 56, pp. 615–624, 07 2008.

[22] R. Pito and R. K. Bajcsy, "Solution to the next best view problem for automated CAD model acquisiton of free-form objects using range cameras," in *Modeling, Simulation, and Control Technologies for Manufacturing*, R. Lumia, Ed., vol. 2596, International Society for Optics and Photonics. SPIE, 1995, pp. 78 – 89.

[23] V. H. Chan and M. Samaan, "Spherical/cylindrical laser scanner for geometric reverse engineering," in *Proceedings of the Conference on Three-Dimensional Image Capture and Applications VI, San Jose, CA, USA, January 18, 2004*, ser. SPIE Proceedings, vol. 5302. SPIE, 2004, pp. 33–40.

[24] S. Ottenhaus, M. Miller, D. Schiebener, N. Vahrenkamp, and T. Asfour, "Local Implicit Surface Estimation for Haptic Exploration," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, Nov. 2016, pp. 850–856.

[25] Z. Yi, R. Calandra, F. Veiga, H. van Hoof, T. Hermans, Y. Zhang, and J. Peters, "Active Tactile Object Exploration with Gaussian Processes," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 4925–4930.

[26] D. Driess, P. Englert, and M. Toussaint, "Active Learning with Query Paths for Tactile Object Shape Exploration," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 65–72.

[27] S. Dragiev, M. Toussaint, and M. Gienger, "Uncertainty aware grasping and tactile exploration," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 113–119.

[28] G. Z. Gandler, C. H. Ek, M. Björkman, R. Stolkin, and Y. Bekiroglu, "Object Shape Estimation and Modeling, Based on Sparse Gaussian Process Implicit Surfaces, Combining Visual Data and Tactile Exploration," *Robotics and Autonomous Systems*, vol. 126, p. 103433, Apr. 2020.

[29] S. Ottenhaus, D. Renninghoff, R. Grimm, F. Ferreira, and T. Asfour, "Visuo-Haptic Grasping of Unknown Objects based on Gaussian Process Implicit Surfaces and Deep Learning," in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, Oct. 2019, pp. 402–409.

[30] D. Watkins-Valls, J. Varley, and P. Allen, "Multi-Modal Geometric Learning for Grasping and Manipulation," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 7339–7345.

[31] E. Smith, R. Calandra, A. Romero, G. Gkioxari, D. Meger, J. Malik, and M. Drozdzal, "3D Shape Reconstruction from Vision and Touch," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 14 193–14 206.

[32] E. J. Smith, D. Meger, L. Pineda, R. Calandra, J. Malik, A. Romero, and M. Drozdzal, "Active 3D Shape Reconstruction from Vision and Touch," *arXiv:2107.09584 [cs]*, Jul. 2021.

[33] W. Lorensen and H. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 163–, 08 1987.

[34] M. G. Crandall and P.-L. Lions, "Viscosity solutions of Hamilton-Jacobi equations," *Transactions of the American mathematical society*, vol. 277, no. 1, pp. 1–42, 1983.

[35] A. Gropp, "IGR: Implicit Geometric Regualrization for Learning Shapes," 2020, accessed: 2021-10-14. [Online]. Available: https://github.com/amosgropp/IGR

[36] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and Model set: Towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics (ICAR)*, 2015, pp. 510–517.

[37] D. Kappler, J. Bohg, and S. Schaal, "Leveraging big data for grasp planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4304–4311.

[38] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," vol. 3, 01 2009.

[39] K. Robotics, "ros_kortex," 2021, accessed: 2021-10-14. [Online]. Available: https://github.com/Kinovarobotics/ros_kortex

[40] M. Marghany, M. Tahar, and M. Hashim, "3d stereo reconstruction using sum square of difference matching algorithm," *Scientific Research and Essays*, vol. 6, pp. 6404–6423, 12 2011.

[41] Intel RealSense, "ROS Wrapper for Intel RealSense Devices," 2021, accessed: 2021-10-14. [Online]. Available: https://github.com/IntelRealSense/realsense-ros

[42] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, May 9-13 2011.

[43] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[44] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[45] J. K. Behrens, "MuJoCo-ROS," 2020, accessed: 2021-10-14. [Online]. Available: https://github.com/JKBehrens/mujoco-ros

[46] K. Mammou, "Volumetric-Hierarchical Approximate Convex Decomposition," 2015, accessed: 2021-10-14. [Online]. Available: https://github.com/kmammou/v-hacd

[47] Interdisciplinary Research Laboratory at Computer Aided Medical Procedures, "easy_handeye," 2021, accessed: 2021-10-14. [Online]. Available: https://github.com/IFL-CAMP/easy_handeye

[48] J. Lundell and F. Verdoja, "Point-Cloud Segmentation," 2019. [Online]. Available: https://github.com/aalto-intelligent-robotics/point_cloud_segmentation/

[49] J. Bimbo, C. Pacchierotti, N. G. Tsagarakis, and D. Prattichizzo, "Collision Detection and Isolation on a Robot using Joint Torque Sensing," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 7604–7609.

[50] D. Popov, A. Klimchik, and N. Mavridis, "Collision detection, localization & classification for industrial robots with joint torque sensors," in *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2017, pp. 838–843.

[51] M. Basseville, I. V. Nikiforov *et al.*, *Detection of abrupt changes: theory and application*. prentice Hall Englewood Cliffs, 1993, vol. 104.

[52] R. Diankov and J. Kuffner, "Openrave: A Planning Architecture for Autonomous Robotics," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, vol. 79, 2008.

[53] H. Fan, H. Su, and L. J. Guibas, "A Point Set Generation Network for 3D Object Reconstruction from a Single Image," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2463–2471, 2017.

[54] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson Surface Reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[55] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The Ball-Pivoting Algorithm for Surface Reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, Oct. 1999.

[56] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull Algorithm for Convex Hulls," *ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE*, vol. 22, no. 4, pp. 469–483, 1996.

[57] O. Williams and A. Fitzgibbon, "Gaussian Process Implicit Surfaces," *Gaussian Proc. in Practice*, 2007.

[58] M. P. Gerardo-Castro, "GPIS," 2014, accessed: 2021-10-14. [Online]. Available: https://github.com/marcospaul/GPIS

[59] A. Miller and P. Allen, "Graspit! A Versatile Simulator for Robotic Grasping," *IEEE Robotics Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.

[60] ——, "Examples of 3d grasp quality computations," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, 1999, pp. 1240–1246 vol.2.

[61] M. Ciocarlie, C. Goldfeder, and P. Allen, "Dimensionality reduction for hand-independent dexterous robotic grasping," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 3270–3275.

[62] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 04 1970.

[63] P. Min, "binvox," http://www.patrickmin.com/binvox, 2004 - 2019, accessed: 2021-10-14.

[64] D. Maturana, "binvox_rw," 2012, accessed: 2021-10-14. [Online]. Available: https://github.com/dimatura/binvox-rw-py

[65] D. Comaniciu and P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[66] P. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[67] S. Rusinkiewicz and M. Levoy, "Efficient Variants of the ICP Algorithm," in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 2001, pp. 145–152.

[68] J. Park, Q.-Y. Zhou, and V. Koltun, "Colored Point Cloud Registration Revisited," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 143–152.

[69] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust Reconstruction of Indoor Scenes," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 5556–5565.

[70] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," *arXiv preprint arXiv:1706.02413*, 2017.

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Rustler Lukáš** Personal ID number: **465972**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Visuo-Haptic Uncertainty-Driven Object Shape Completion

Master's thesis title in Czech:

Visuo-haptické modelování tvaru objektu řízené nejistotou

Guidelines:

1) Research and evaluate state of the art object shape completion methods.
2) Develop a simulation environment (e.g., Gazebo, Mujoco).
3) Develop a visuo-haptic approach for shape completion.
a) Start from a depth image of the object under investigation and complete the object shape. Focus on methods that keep track of the uncertainty of the estimated shape.
b) Use a robot manipulator to touch the object where the uncertainty would be reduced the most.
4) Experimentally evaluate the approach in both simulation and real setup (Kinova Gen3 manipulator).
5) Develop a visual only approach (use the RGB-D sensor in the robot wrist to look from different viewpoints) and compare with visuo-haptic approach.
6) Discuss the results.

Bibliography / sources:

[1] Lundell, J.; Verdoja, F. & Kyrki, V. (2019), Robust Grasp Planning Over Uncertain Shape Completions, in '2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', pp. 1526-1532.
[2] Park, J. J., Florence, P., Straub, J., Newcombe, R., & Lovegrove, S. (2019). DeepSDF: Learning continuous signed distance functions for shape representation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 165-174).
[3] Watkins-Valls, D., Varley, J., & Allen, P. (2019). Multi-modal geometric learning for grasping and manipulation. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 7339-7345). IEEE.
[4] Dai, A., Ruizhongtai Qi, C., & Nießner, M. (2017). Shape completion using 3d-encoder-predictor cnns and shape synthesis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5868-5877).

Name and workplace of master's thesis supervisor:

Mgr. Matěj Hoffmann, Ph.D., Vision for Robotics and Autonomous Systems, FEE

Name and workplace of second master's thesis supervisor or consultant:

Jens Lundell, Aalto University, Helsinki, Finland

Date of master's thesis assignment: **07.09.2021** Deadline for master's thesis submission: **04.01.2022**

Assignment valid until: **19.02.2023**

Mgr. Matěj Hoffmann, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature