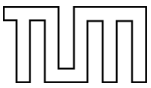


Technische Universität München

Department of Mathematics

Master's Thesis

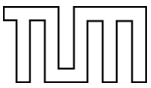
Object Pose Estimation with PointNet

Michael Haberl

Supervisor: Prof. Dr. Ulrich Bauer

Advisor: Benjamin Busam, M.Sc.

Submission Date: 30.09.2018



Technische Universität München

Department of Mathematics

Master's Thesis

Object Pose Estimation with PointNet

Michael Haberl

Supervisor: Prof. Dr. Ulrich Bauer

Advisor: Benjamin Busam, M.Sc.

Submission Date: 30.09.2018

I assure the single handed composition of this master's thesis only supported by declared resources.

Garching, 30.09.2018

Acknowledgement

A sincere thank you to everyone who supported me in writing this thesis.

I am greatly indebted to my supervisors Prof. Ulrich Bauer and Mr. Benjamin Busam for giving me the opportunity to write this thesis on such an interesting topic and for being my constant guide throughout.

I would also like to thank the team at FRAMOS GmbH for providing me with a wonderful working environment and supporting me with the computational resources to train the networks.

Special thanks go to my friends, Michael and Nelson, who helped me out by proofreading my thesis. Finally, thanks to Hridya for writing your thesis alongside me and still having an open ear to talk about my thesis.

Abstract
What does it take for a computer to detect the location and orientation of objects? The aim of this thesis to try to answer this question using neural networks to evaluate point clouds of the objects.
Chapter 1 gives an overview on recent work in pose estimation. Chapter 2 gives theoretical background in pose estimation and machine learning. After that the datasets, used in this thesis are introduced in chapter 3. The challenges in pose estimation are explained in chapter 4. Afterwards, different network architectures and their results are evaluated on $2D$ and $3D$ datasets in chapter 5.

Zusammenfassung
Wie kann ein Computer die Position und Orientierung von Objekten erkennen? Diese Masterarbeit versucht dieses Problem mithilfe von neuronalen Netzwerken, die Punktwolken von den Objekten evaluieren, zu lösen.
Kapitel 1 bietet einen Überblick zu aktuellen Ansätzen in Posenschätzung. In Kapitel 2 ist der theoretische Hintergrund zu Posenschätzung und Maschinellen Lernen. Danach werden die Datensätze, die in dieser Masterarbeit verwendet wurden, in Kapitel 3 vorgestellt. Die Herausforderungen in Posenschätzung sind in Kapitel 4 dargelegt. Anschließend, in Kapitel 5, werden die Ergebnisse von verschiedenen Netzwerkarchitekturen mit $2D$ und $3D$ Datensätzen evaluiert.

Contents

Symbols

Symbol	Explanation
$\mathbf{x}$	Point
$\mathbf{q}$	Quaternion
$\mathbf{R}$	Rotation matrix
$\mathbf{t}$	Translation
$\mathbf{P}$	Pose
$\mathcal{M}$	Object model, source
$\mathcal{P}$	Point cloud
$\hat{.}$	Estimated value. E.g. $\hat{\mathbf{P}}$ estimated pose
$\bar{.}$	Ground truth value. E.g. $\bar{\mathbf{P}}$ ground truth pose
$\|.\|$	L^2-norm
$\langle ., . \rangle$	Scalar product
$\mathbf{1}_n$	$n \times n$ identity matrix

1 Introduction

In pose estimation, the main task is to estimate the position and orientation of objects relative to the camera[1]. This is, for example, relevant in robotics where, for example, a robot needs to pick up an object and therefore, needs to know where that object is located and how it is oriented. Another application of pose estimation can be found in augmented reality. Here, virtual objects are rendered to real scenes and it has to be ensured that these objects have the correct pose.

In this thesis, these objects are described through target point clouds $\bar{\mathcal{M}}$ that are calculated with a depth map taken by an RGBD-camera. The source point cloud $\mathcal{M}$ of the centred objects is also given as reference. The target point cloud describes the rotated and translated object. The goal of this thesis is to modify PointNet [QSMG16], a neural network architecture designed for point cloud inputs, such that the network has as input the source and target point clouds and estimates the rotation and translation of the target with respect to the source.

One approach to pose estimation is using an alternating optimisation of the corresponding points and the transformation between the point clouds. The iterative closest point (ICP) algorithm [BM92] estimates in each iteration the corresponding points (with the nearest neighbour) and then the pose with rotation and translation. Then the point cloud is transformed with the previously estimated transformation and the process is repeated. This method is prone to be stuck in local minima when the point clouds are initially not well aligned [YLCJ16]. To solve this issue Yang et al. proposed Go-ICP [YLCJ16], a method that finds the global optimum using ICP. Go-ICP splits the translation domain and rotation domain into smaller parts and then uses the branch-and-bound method [LD60] combined with ICP to get the globally optimal estimate of the pose. While ICP just uses one-to-one assignment of corresponding points, robust point matching (RPM) [GRL+98], [BEC+15] uses soft assignment where the correspondence probability between two points is a value in $[0, 1]$. The method is then iteratively getting the correspondences and poses similarly to ICP. Coherent point drift (CPD) [MS10] uses also an alternating optimisation, to register non-rigid transformations.

Another approach to estimate the pose of objects is by feature matching. Once the corresponding features are known, the transformation between them can be calculated easily. Since in rigid pose estimation the pose is uniquely determined by 3 non degenerate points RANSAC [FB81], can be used to find such points that belong together. RANSAC samples some points from the input and then fits them together. This is used in [IR96] and [CHC99]. Point pair features with the points and their normals (PPF) are used to find corresponding points in [DBI18], [RBB09], [DUNI10] and lead to state-of-the-art results in [VLM].

The third approach to pose estimating is template matching. [HLI+13] uses RGB and depth images to sample the possible views for different poses and create templates for these poses. These templates can then be compared to the images in the dataset to get

[1] Equivalently pose estimation can be seen as estimating the position of the camera relative to objects.

the poses. Other template-based methods are [RCT13], which allows to learn the templates online and [KTD+16], where Hough Forests are used.

Recently, deep learning approaches are used for pose estimation. Using depth maps, Georgakis et al. [GKW+18] uses convolutional neural networks to find correspondences. Other methods are [ZK15], [WL15], [KMT+17].

Siamese networks, an architecture used in this thesis, were used to detect poses of body parts in [VBVC17] and [DBKK16].

The paper by Hodan et al. [T. 18] gives an overview over the current state of the art in pose estimation.

2 Theoretical Background

This chapter starts with an explanation on how the point clouds were created using RGBD-cameras (section 2.1 to 2.2). Then the parametrisation of the pose with its different representations is introduced in section 2.4 and section 2.5 and finally some background in machine learning and the architecture that this thesis is based on is given in section 2.6 and 2.7.

2.1 Camera Pinhole Model

The camera projects a world point $\mathbf{x}_w = (x, y, z)^T \in \mathbb{R}^3$ through the *pinhole* to the point $\mathbf{x}$ on the *image plane* (see figure 2.1). The distance from the pinhole P or projection centre to the image plane is called *focal length* f. The *principal axis* is the line that is perpendicular to the image plane through the pinhole. The *principal point* $\mathbf{p}$ is the point on the image plane, where the principal axis passes through the image plane [HZ03].

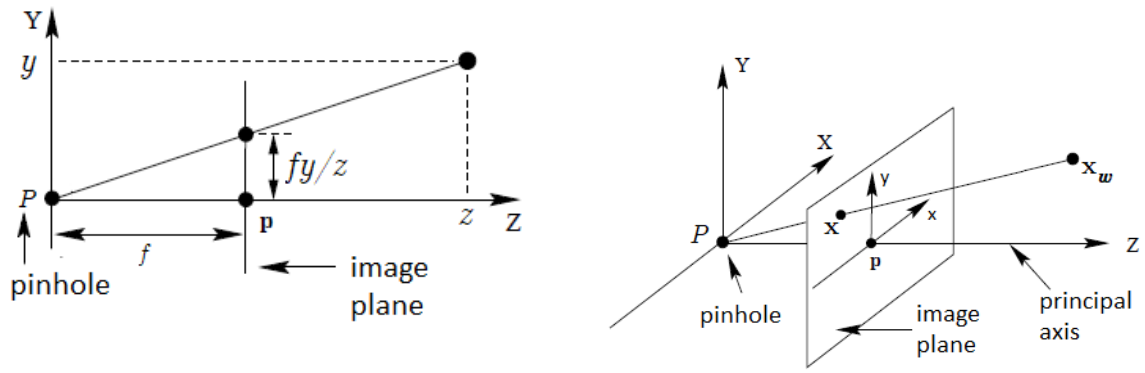


Figure 2.1: Pinhole camera geometry

P is the pinhole of the camera, f the focal length and $\mathbf{p}$ the principal point. Note that the image plane in this figure is before the pinhole, whereas in the camera it is after the pinhole. Images based on [HZ03] p. 154.

Figure 2.2 shows the image coordinate system (x_{img}, y_{img}) and the image plane coordinate system (x, y). The image coordinate system is shifted by the offset x_0, y_0 and scaled with the pixel width s_x and pixel height s_y. The projection of the world point $\mathbf{x}_w = (x, y, z)^T$ to the image plane coordinate system is given by [HZ03]:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} s_x^{-1} f x + z x_0 \\ s_y^{-1} f y + z y_0 \\ z \end{pmatrix} = \begin{pmatrix} f_x & 0 & x_0 & 0 \\ 0 & f_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},$$

where $f_x = s_x^{-1} f$ and $f_y = s_y^{-1} f$ are the focal lengths scaled by the pixel width and height. The matrix $\mathbf{K} = \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$ is called the *intrinsic camera matrix*.

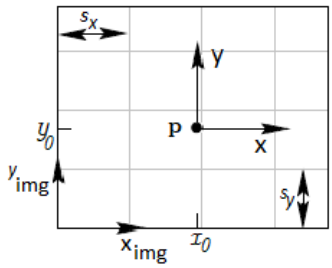
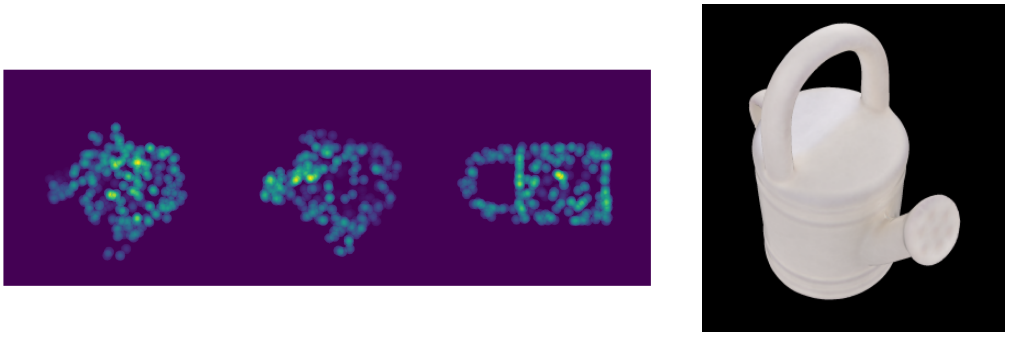


Figure 2.2: Principal offset

Image based on [HZ03] p. 155.

2.2 Point Clouds

A *point cloud* $\mathcal{P}$ is a set of N points $\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with $\mathbf{x}_i \in \mathbb{R}^n$. For the purpose of this thesis, mostly three-dimensional points with coordinates $\mathbf{x}_i = (x_i, y_i, z_i)$ are considered. Apart from coordinates, other possible features of a point $\mathbf{x}$ could be the colour of the point or surface normals. An example for a point cloud of the can in the Linemod dataset [HLI+13] can be seen in figure 2.3.



(a) Point cloud of a can (b) Image of the can

Figure 2.3: Example for a point cloud

The left image shows three views of the point cloud of the can from the Linemod dataset by Hinterstoisser et al. [HLI+13]. Yellow points are close to the camera, blue points are in the background. The image to the right shows an image of the can.

A set of points in a point cloud is without any specific order. That means a neural network that has a point cloud as input has to be invariant to permutations of the points. To achieve that, the network uses pooling layers (discussed in section 5.2.2) inspired by the PointNet from Qi et al. [QSMG16] (introduced in section 2.7).

2.3 From Depth to Point Cloud

Definition 2.1. A *depth image* $I \in \mathbb{N}^{n \times m}$ contains in each pixel p the discretized z_W-coordinate of the point $\mathbf{x}_W = (x_W, y_W, z_W)$ that projects to pixel p.

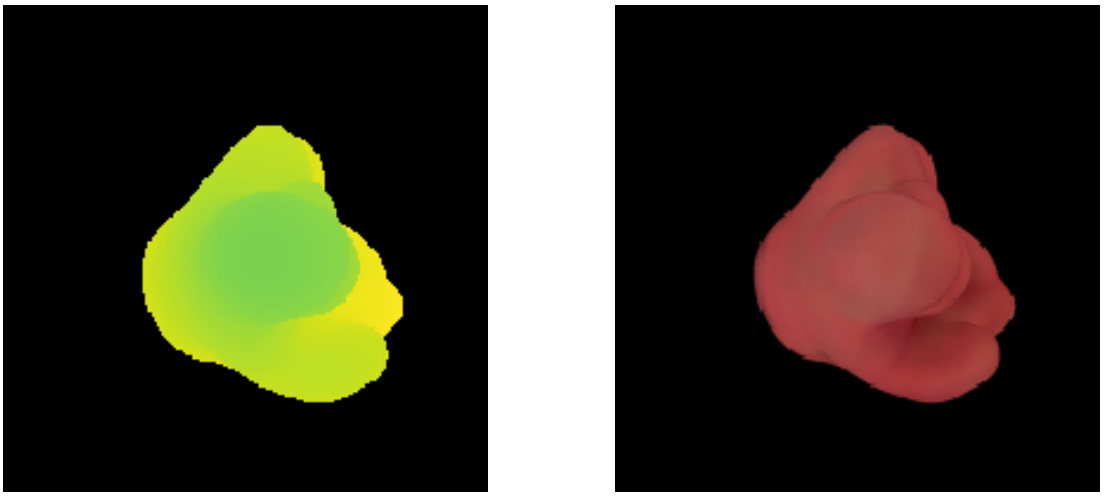
The depth images from the Linemod dataset contain the distance from the camera in millimetres. Depth images can be taken using stereo cameras [KNO11], LIDAR [PGA+16] or active sensors [GAA+12]. For a camera with intrinsic matrix

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

and a depth image I with the values of the distances stored in each pixel, the corresponding 3D point $\mathbf{x}_{i,j}$ of the pixel $I(i,j)$ is computed by:

$$\begin{aligned} z &= I(i,j) \\ x &= (i - x_0)I(i,j)/f_x \\ y &= (j - y_0)I(i,j)/f_y \\ \mathbf{x}_{i,j} &= (x, y, z)^T . \end{aligned}$$

An example for a depth image can be seen in figure 2.4, the corresponding point cloud is in figure 2.5 and the point cloud without partial visibility is in figure 2.6.



(a) Depth image (b) RGB image

Figure 2.4: Depth image and RGB image of the Linemod ape

Example of an RGB image and corresponding depth image from the Linemod dataset by Hinterstoisser et al. [HLI+13]. Green pixels in the depth image are close to the camera, yellow pixels are further away. For pixels in black there is no depth information available.

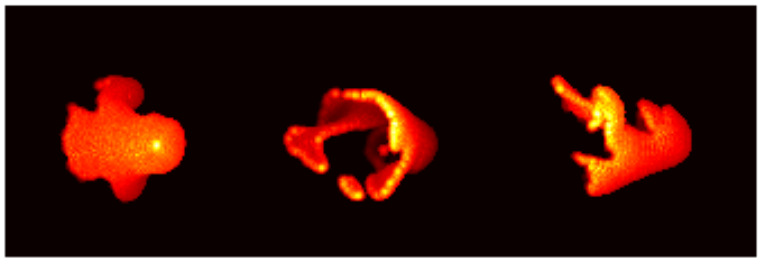


Figure 2.5: Point cloud of the ape

Three views of the point cloud calculated from the depth image in figure 2.4a. Bright points are close to the camera and dark red points are further away. Note that the bottom of the ape is not visible due to self occlusion (see section 4.1.2).

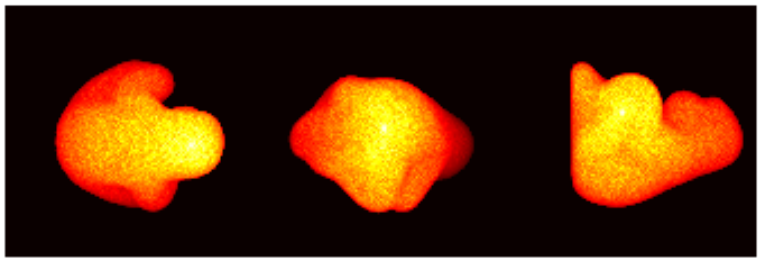


Figure 2.6: Point cloud of the ape with all points

Three views of the point cloud from the ape without occlusions.

2.4 Pose Parametrisation

In this thesis, the pose is a rigid transformation in $3D$ with a rotation $\mathbf{R} \in SO(3)$ and the translation $\mathbf{t} \in \mathbb{R}^3$ [2]. This transformation is an element of the *Special Euclidean Group* $SE(3)$, which has the homogeneous representation $\mathbf{P} = [\mathbf{R}, t; 0, 1] \in \mathbb{R}^{4\times4}$. $SE(3)$ is a Lie group.

Definition 2.2. A *Lie group* is a set $\mathcal{G}$ with an operation $*$ so that $(\mathcal{G}, *)$ it has group properties [SWW73]:

- Closure: $\forall a, b \in \mathcal{G} : \ a * b \in \mathcal{G}$
- Associativity: $\forall a, b, c \in \mathcal{G} : \ (a * b) * c = a * (b * c)$
- Identity: $\exists e \in \mathcal{G}$ s.t. $\forall a \in \mathcal{G} : e * a = a * e$
- Inverse element: $\forall a \in \mathcal{G} : \exists b \in \mathcal{G}$ s.t. $a * b = b * a = e$,

and is also a smooth manifold with smooth group multiplication μ and group inversion ι:

[2] The representation of the rotation can be different in the implementations, for example with quaternions $\mathbf{q}$ or in axis-angle representation. The different representations are discussed in section 2.5.

$$\mu : \mathcal{G} \times \mathcal{G} \to \mathcal{G}, (a, b) \mapsto a * b$$
$$\iota : \mathcal{G} \to \mathcal{G}, a \mapsto a^{-1}$$

Every Lie group has an associated Lie algebra, which is the tangent space around the identity element of the group.

Definition 2.3. A *Lie algebra* is a vector space $\mathfrak{g}$ over some field $\mathbb{F}$ together with an operation $[.,.] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$ that has the following properties [Bak02]:

1. Bilinearity

$$[ax + by, z] = a[x, z] + b[y, z] \tag{2.1}$$
$$[z, ax + by] = a[z, x] + b[z, y] \tag{2.2}$$

   $\forall a, b \in \mathbb{F},\ x, y, z \in \mathfrak{g}$.

2. Skew symmetry

$$[x, y] = -[y, x] \tag{2.3}$$

   $\forall x, y \in \mathfrak{g}$.

3. The Jacobi identity

$$[x, [y, z]] + [z, [x, y]] + [y, [z, x]] = 0 \tag{2.4}$$

   $\forall x, y, z \in \mathfrak{g}$.

With the Lie algebra the tangent space at any element of its Lie group can be constructed via parallel transport [BBN]. This can be used to compare poses (see equation 2.24).

The pose $\mathbf{P}$ has 6 degrees of freedom: 3 in the translation $\mathbf{t}$ and 3 in the rotation $\mathbf{R}$. The point $\mathbf{x} = (x_1, x_2, x_3) \in \mathcal{M}$ is rotated and translated in homogeneous coordinates using $\mathbf{P}$ to the point $\mathbf{x}'$ by:

$$\mathbf{x}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}$$

Hereafter the short form $\mathbf{x}' = \mathbf{P}\mathbf{x}$ is used for the transformation of points and $\mathcal{M}' = \mathbf{P}\mathcal{M}$ for the transformation of point clouds $\mathcal{M}$. The point $\mathbf{x}$ and the transformed point $\mathbf{x}'$ are called *corresponding points*.

2.5 Representations of Rotations

Rotations can have many representations like $SO(3)$ matrices, axis-angles [HMK+18], Euler angles [ETW08] or quaternions [BBN]. In this section, the different representations used in this thesis are presented as well as the transformations that were used in this thesis, to switch between different representations[3].

[3] For a complete overview of transformations between the representations see [Die06]

Quaternions

Definition 2.4. A *quaternion* $\mathbf{q}$ is an element of the algebra of quaternions $\mathbb{H}$ and has the form [BBN]:

$$\mathbf{q} = q_1 1 + q_2 i + q_3 j + q_4 k = (q_1, q_2, q_3, q_4)^T$$

with $(q_1, q_2, q_3, q_4)^T \in \mathbb{R}^4$ and $i^2 = j^2 = k^2 = ijk = -1$.

The product of two quaternions $\mathbf{q}, \mathbf{p}$ is defined as [Voi18]:

$$\begin{aligned}\mathbf{q} \cdot \mathbf{p} := {} & q_1 p_1 - q_2 p_2 - q_3 p_3 - q_4 p_4 + (q_1 p_2 + q_2 p_1 + q_3 p_4 - q_4 p_3) i \\ & + (q_1 p_3 - q_2 p_4 + q_3 p_1 + q_4 p_2) j + (q_1 p_4 + q_2 p_3 - q_3 p_2 + q_4 p_1) k\end{aligned} \tag{2.5}$$

and the sum is defined as

$$\mathbf{q} + \mathbf{p} := (q_1 + p_1) + (q_2 + p_2) i + (q_3 + p_3) j + (q_4 + p_4) k.$$

A quaternion can also be written as $\mathbf{q} = [a, \mathbf{v}]$ with the scalar part $a = q_1 \in \mathbb{R}$ and the vector part $\mathbf{v} = (q_2, q_3, q_4)^T \in \mathbb{R}^3$. The conjugate $\bar{\mathbf{q}}$ of a quaternion $\mathbf{q}$ is defined by:

$$\bar{\mathbf{q}} := q_1 - q_2 i - q_3 j - q_4 k.$$

Unit quaternions $\mathbf{q} \in S^3$ with $\left\| (q_1, q_2, q_3, q_4)^T \right\| = 1$ can represent rotations. The sphere of unit quaternions S^3 is a smooth and compact manifold [Lee03]. To rotate a point $\mathbf{x} \in \mathbb{R}^3$ with the rotation given by $\mathbf{q}$ the point is first transformed into a *vector quaternion* $\mathbf{x}'$ with scalar part $q_1 = 0$ s.t. $\mathbf{x}' = (0, \mathbf{x})$. Then, the rotated point is obtained using quaternion multiplication $\mathbf{x}_{rot} = \mathbf{q} \cdot \mathbf{x}' \cdot \bar{\mathbf{q}}$ 2.5. With this formula, it holds that $\mathbf{x}_{rot} = \mathbf{q} \cdot \mathbf{x}' \cdot \bar{\mathbf{q}} = -\mathbf{q} \cdot \mathbf{x}' \cdot -\bar{\mathbf{q}}$ and therefore antipodal quaternions $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation [HZ03]. This ambiguity of the quaternions can create issues when training a neural network, since the network can output only one of two different quaternions $\mathbf{q}$ and $-\mathbf{q}$ and the rotation represented by them is the same (see section 4.1).
A rotation around the unit vector $\mathbf{v}$ by an angle θ (see section 2.5) is represented by the unit quaternion

$$\mathbf{q} = \left(\cos\left(\frac{\theta}{2}\right), \mathbf{v} \sin\left(\frac{\theta}{2}\right) \right). \tag{2.6}$$

Rotation Matrix

Another way to represent rotations is with a matrix $\mathbf{R} \in SO(3)$. The Lie group $SO(3)$ is called the *special orthonormal group* or *rotation group*. A matrix $\mathbf{R} \in \mathbb{R}^{3\times 3}$ is in the special orthonormal group if and only if:

$$\mathbf{R}^T \mathbf{R} = \mathbf{1}_3 \tag{2.7}$$

$$\det \mathbf{R} = 1. \tag{2.8}$$

To rotate the vector $\mathbf{x} = (x_1, x_2, x_3)^T \in \mathbb{R}^3$ by the rotation $\mathbf{R} \in SO(3)$ the matrix is multiplied to the vector:

$$\mathbf{x}' = \mathbf{R}\mathbf{x}.$$

For example the counter-clockwise rotation with the angle ϕ around the x_1 axis is represented by:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}.$$

In $2D$ the point $\mathbf{x} \in \mathbb{R}^2$ is rotated counter-clockwise by the angle ϕ with

$$\mathbf{x}' = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \mathbf{x}.$$

The rotations in 2D can be represented with the circle group $U(1)$ by:

$$\theta \mapsto z = e^{i\theta} = \cos\theta + i\sin\theta.$$

A quaternion $\mathbf{q}$ can be represented by a rotation matrix $\mathbf{R} \in SO(3)$ using the following transformation [Voi18]:

$$\mathbf{R} = \begin{pmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2q_2q_3 - 2q_1q4 & 2q_2q_4 + 2q_1q_3 \\ 2q_2q_3 + 2q_1q_4 & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2q_3q_4 - 2q_1q_2 \\ 2q_2q_4 - 2q_1q_3 & 2q_3q_4 + 2q_1q_2 & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{pmatrix}.$$

Axis-angle

Rotations can also be described with a vector $\mathbf{r}$. The rotation axis is represented as a 3-dimensional unit vector $\mathbf{u} = (u_1, u_2, u_3)^T \in S^2$ and then the vector $\mathbf{u}$ is multiplied by the angle θ to get the rotation vector $\mathbf{r} = \theta\mathbf{u}$. Similarly to the quaternions, this representation is also ambiguous since one can rotate around the vector $\mathbf{u}$ and $-\mathbf{u}$.
To rotate a vector $\mathbf{v}$ counter-clockwise with $\mathbf{r} = \theta\mathbf{u}$, Rodrigues' rotation formula can be used [Voi18]:

$$\mathbf{v}_{rot} = \mathbf{v}\cos\theta + (\mathbf{u} \times \mathbf{v})\sin\theta + \mathbf{u}\langle\mathbf{u}, \mathbf{v}\rangle(1 - \cos\theta),$$

where $\times$ is the cross product of two vectors and $\langle\mathbf{u}, \mathbf{v}\rangle$ is the scalar product.
For quaternions with $q_1 = \pm 1$ (i.e. a rotation of zero degree), the rotation vector is chosen to be $\mathbf{r} = (0, 0, 0)^T$. To get the rotation axis $\mathbf{r}$ from a quaternion $\mathbf{q}$ with $|q_1| < 1$, the following formula can be used [LF05]:

$$\begin{aligned} \theta &= 2\arccos(q_1) \\ \mathbf{u} &= \frac{(q_2, q_3, q_4)^T}{\sin\left(\frac{1}{2}\theta\right)} \\ \mathbf{r} &= \theta\mathbf{u}. \end{aligned}$$

To get the axis and angle from a rotation matrix $\mathbf{R}$, one can use the fact that a vector $\mathbf{v}$ that is parallel to the rotation axis is not affected by the rotation i.e. $\mathbf{R}\mathbf{v} = \mathbf{v}$. To get

the rotation in axis-angle form $\mathbf{r}$, the following formula can be used [ETW08]:

$$\begin{aligned} \theta &= \arccos\left(\frac{\mathrm{Tr}(\mathbf{R})-1}{2}\right) \\ \mathbf{u} &= \frac{1}{2\sin\theta}\begin{pmatrix} \mathbf{R}_{(3,2)}-\mathbf{R}_{(2,3)} \\ \mathbf{R}_{(1,3)}-\mathbf{R}_{(3,1)} \\ \mathbf{R}_{(2,1)}-\mathbf{R}_{(1,2)} \end{pmatrix} \\ \mathbf{r} &= \theta\mathbf{u}. \end{aligned} \tag{2.9}$$

2.6 Supervised Learning with Neural Networks

This section gives first a short overview of machine learning and supervised learning, then the structure of neural networks is explained in sections 2.6.1 and 2.6.2, the optimisation methods in sections 2.6.3 to 2.6.5 and the loss functions in section 2.6.7.

Machine learning has three major branches: *supervised learning*, *unsupervised learning* and *reinforcement learning* [Mur13].
In supervised learning, one is given some dataset $D = \{x_1, x_2, x_3, ..\}$, where x_i can be, for example, images, sound files or point clouds. For each of these data points, there is also a desired output y_i given, which is referred to as *ground truth*. This ground truth can be a discrete label, then the task is *classification*, or it is a continuous variable, then it is referred to as *regression*. Examples for datasets used in classification problems would be the well-studied MNIST-dataset [LC10] to recognize handwritten numbers or the Bach Choral Harmony Data Set [RE10] to recognize the pitch of the cords in Bachs compositions. Some datasets used in regression are the Boston Housing dataset [HR78] to predict the house prices in Boston or the El Nino Dataset [Lab99] used to predict the weather.

In unsupervised learning, just the dataset $D = \{x_1, x_2, x_3, ..\}$ is given and the goal is to find patterns in the data. Since it is not clear which patterns to look for in the data and there is no obvious error metric like in supervised learning, it is much less well-defined than other forms of machine learning [Mur13]. Some examples for unsupervised learning are clustering, for example with the EM-algorithm [DLR77], dimensionality reduction via autoencoders [Bal11] or getting depth information from active stereo sensors [ZKR+18].

Reinforcement learning is used for decision-making problems. There, an agent gets as input the state of an environment (for example, a game) and returns an action to carry out. Reinforcement learning can also be used for non-differentiable loss functions, whereas supervised learning with neural networks relies on differentiable loss functions to update the network weights [HCI+18]. Examples for use cases of reinforcement learning are playing Atari games [MKS+13] or the recent breakthrough at the game of Go by Silver et.al. [SSS+17].

The focus of this thesis will be to use supervised learning to regress the 6D poses of objects, given by point clouds.

In supervised learning with neural networks, a model is trained by feeding a dataset $D = \{x_1, x_2, x_3, ..\}$ as well as the desired outputs y_i for each data point to the network. The network generates for each x_i the output $\hat{y}_i$, which is compared to the ground truth. The network weights $\mathbf{w}$ are updated by a stochastic gradient descent method (introduced in sections 2.6.4 to 2.6.5) such that the difference between the output and the ground truth becomes smaller [Kot07]. The loss functions that quantify the error between network output and ground truth are introduced in section 2.6.7 .
Figure (2.7) is an example of a neural network with a two-dimensional input x, one hidden layer with weights $\mathbf{w}$ and the predicted label $\hat{y}$. The error e is here the difference between the ground truth label y and the predicted label $\hat{y}$.

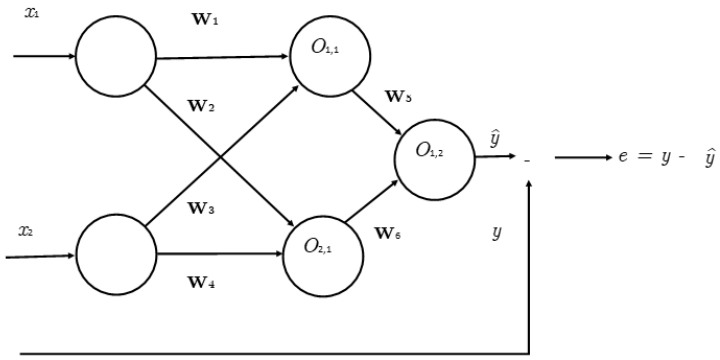


Figure 2.7: Simple neural network

Fully connected network with a two-dimensional input, one hidden layer with two nodes and one output node. The figure is based on [Mur13] p. 998.

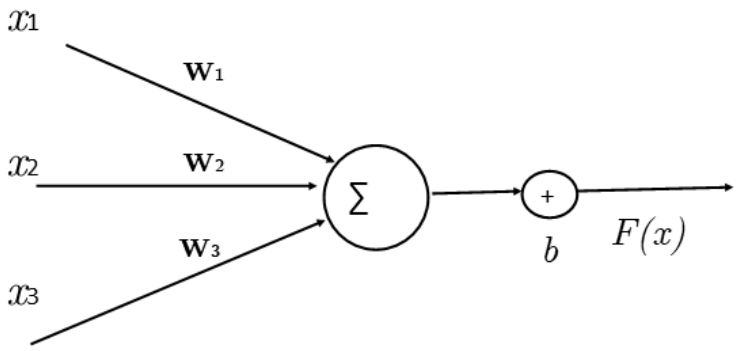


Figure 2.8: Node in neural network

Node with three inputs x_i, the weights $\mathbf{w_i}$, bias b and activation function $F(x)$. Figure based on [Mur13] p. 998.

2.6.1 Structure of Neural Networks

A neural network consists of layers of nodes, (see figure 2.8) which are connected to each other. The jth node of the lth layer can be seen as a function, which generates an output dependent on its input $\mathbf{x} \in \mathbb{R}^n$ with the formula [Bis06]:

$$O_{j,l} = F\left(\sum_{i=1}^{n} x_i \cdot \mathbf{w}_i + b\right),$$

where $F(x)$ is an activation function (see section 2.6.2) and $\mathbf{w}_i$, b are the weights and bias of the node. The number of nodes m in each layer is a hyperparameter of the network. The first layer is called *input layer* and there the data x_i enters the network. The output of the first layer ($O_{1,1}$ and $O_{2,1}$ in figure 2.7) becomes the input to the second layer, which then generates the output $O_{1,2}$. The output $O_l \in \mathbb{R}^m$ of each layer l is computed in the following way:

$$O_l = F^l(\mathbf{w}^l x + \mathbf{b}^l),$$

with $x \in \mathbb{R}^n$ the output of the previous layer O_{l-1} and the trainable weights $\mathbf{w}^l \in \mathbb{R}^{m \times n}$ and biases $\mathbf{b}^l \in \mathbb{R}^m$ of layer l. F^l is a non-linear activation function (see section 2.6.2). The procedure of layers generating output for the next layer is repeated until the last layer, called *output layer*. All layers which are between input and output layers are called *hidden layers*. A layer where all nodes are connected by weights to all nodes in the next layer is said to be *fully connected* or *dense*.
Another layer is the convolution layer that is widely used in computer vision [KSH12a], [TSF17],[HMK+18],[KMT+17],[Tho17]. A convolution layer takes several feature maps or colour channels as input and generates n feature maps as output, where n is the number of filters (also called kernels) in the convolution layer. The filter $\mathcal{F} \in \mathbb{R}^{f_w \times f_h \times d}$ is convolved with the image $I \in \mathbb{R}^{w \times h \times d}$ to generate the output image I'. f_w represents the filters width, f_h the filters height and d is the number of input channels (3 for an RGB image). Each pixel $I'(x, y)$ of the output image is calculated by point-wise multiplication of one filter element with one element of the input image I [Tho17]:

$$I'(x,y) = \sum_{i_x=1-\lceil \frac{f_w}{2} \rceil}^{\lfloor \frac{f_w}{2} \rfloor} \sum_{i_y=1-\lceil \frac{f_h}{2} \rceil}^{\lfloor \frac{f_h}{2} \rfloor} \sum_{i_c=1}^{d} I\left(x + i_x, y + i_y, i_c\right) \cdot \mathcal{F}\left(i_x, i_y, i_c\right).$$

Figure 2.9 shows an example of a 3×3 filter applied to an image with one channel ($d = 1$).

In a convolution layer the output of n different filters is calculated and thus the output of the convolution layer is [Tho17]:

$$o^{(x,y,z)} = b + \sum_{i_x=1-\lceil \frac{f_w}{2} \rceil}^{\lfloor \frac{f_w}{2} \rfloor} \sum_{i_y=1-\lceil \frac{f_h}{2} \rceil}^{\lfloor \frac{f_h}{2} \rfloor} \sum_{i_c=1}^{d} I\left(x + i_x, y + i_y, i_c\right) \cdot \mathcal{F}_z\left(i_x, i_y, i_c\right),$$

with the bias $b \in \mathbb{R}$, $x \in \{1, ..., w\}$, $y \in \{1, ..., h\}$ and $z \in \{1, ..., n\}$. Figure 2.11 shows the results of different filters on an image from the dataset from Hinterstoisser et al. [HLI+13].

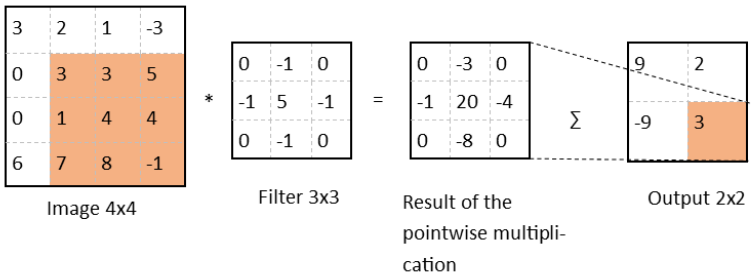


Figure 2.9: Example for a convolution

Visualisation of a 3×3 filter applied to a 4×4 image. The $*$ operator represents point-wise multiplication.

The hyperparameters of the convolution layer are the number of filters n, the filter size $f_w \times f_h$, the stride s and the padding p. The stride specifies how much the filter moves over the picture and the padding adds rows and columns to the image, to ensure that size of the feature maps do not change. An illustration of stride and padding can be found in figure 2.10. Typical choices for the hyperparameters are number of filters $n \in \{32, 64, 128\}$, filter dimensions $f_w = f_h = f \in \{1, 3, 5, 7\}$ and stride $s = 1$. The padding is usually zero-padding. The most commonly used activation function, the rectified linear unit (ReLU), is also used for the models in this thesis [SLJ+15],[KSH12b]. Other possible activation functions are discussed in section 2.6.2.

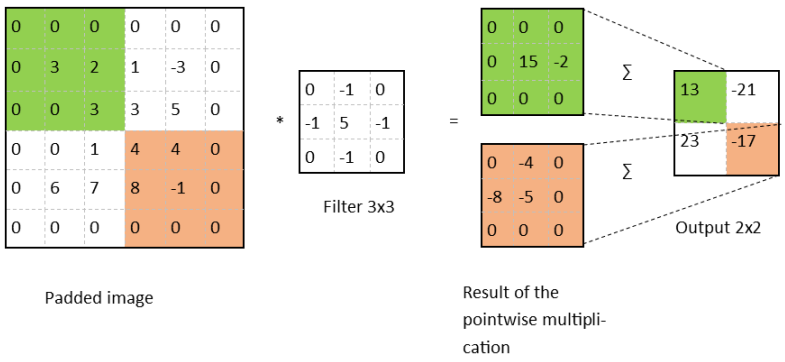


Figure 2.10: Padding and stride

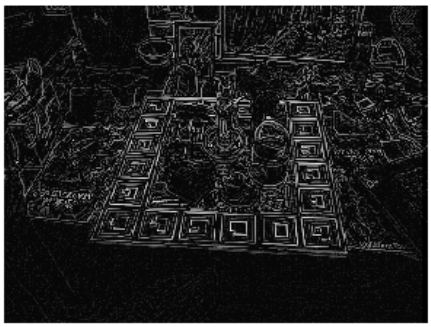
Visualisation of a 3×3 filter applied to an 4×4 image with a zero padding of one and stride $s = 3$.

2.6.2 Activation Functions

The output of a node in a neural network depends on a non-linear activation function. Some common activation functions are the sigmoid function (2.10) in figure 2.12b, the hyperbolic tangent (2.11) in figure 2.12c or the rectified linear unit (ReLU) (2.12) as seen in figure 2.12a.

(a) original picture

(b) filter to sharpen



(c) filter to detect edges

(d) blurring filter

Figure 2.11: Effects of different convolutional filters

$$F(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

$$F(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.11}$$

$$F(x) = \max(0, x) \tag{2.12}$$

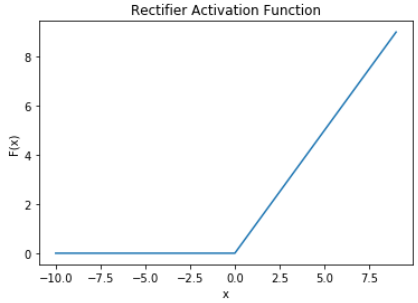
The ReLU activation function is typically used in convolutional neural nets, the sigmoid function and the hyperbolic tangent function are, for example, used in recurrent neural networks (RNN) [GSK+17].

2.6.3 Gradient Descent

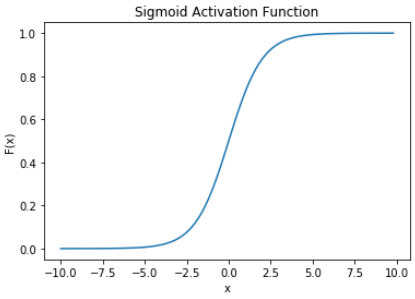
Gradient descent can be used to minimise the loss of the network $L(\mathbf{w}_t)$. $\mathbf{w}_t$ are the weights that are used in the current step, γ is the learning rate, $\nabla_{\mathbf{w}_t}$ is the gradient with respect to the weights and $L(\mathbf{w}_t)$ is the current loss. Then the weights $\mathbf{w}$ for the next step are updated by [Bis06]

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma \nabla_{\mathbf{w}_t} L(\mathbf{w}_t). \tag{2.13}$$

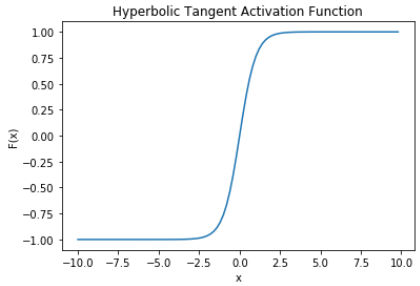
The learning rate γ affects how much the weights are updated. The learning rate is usually decreased during the training to improve the convergence of the network [Bis06].



(a) The rectifier function



(b) The sigmoid function



(c) The hyperbolic tangent function

Figure 2.12: Overview of different activation functions

2.6.4 Mini-batch Stochastic Gradient Descent

For large datasets gradient descent is not practical, since it calculates the gradient for the whole dataset and each update of the weights takes very long. One possible way to deal with this is to use Stochastic Gradient Descent (SGD) and just calculate the gradient of one data point of the whole dataset. This however leads to slower training speed because the gradients are changing very much in each training step [Bot10]. Mini-batch stochastic gradient descent solves this issue by partitioning the whole dataset into batches of several data points and calculating the gradient with respect to one batch. The dataset is shuffled before the partitioning because each batch should give a representation of the whole dataset. Typical batch sizes range between 32 and 4000, dependent on the problem to solve and the hardware available [GDG+17].

2.6.5 Adam Optimizer

To train the networks for this thesis the Adam optimizer [KB14] was used. The update rule of Adam with gradient $g = \nabla_{\mathbf{w}_t} L(\mathbf{w}_t)$ is:

$$\begin{aligned} s_{t+1} &= \beta_1 s_k + (1-\beta_1) g \\ r_{t+1} &= \beta_2 r_k + (1-\beta_2) g \odot g \\ \mathbf{w}_{k+1} &= \mathbf{w}_k - \gamma \frac{\hat{s}}{\delta + \sqrt{\hat{r}_{t+1}}}, \end{aligned}$$

with the unbiased first and second moments $\hat{s}_{t+1} = \frac{s_{t+1}}{1-\rho_1}$ and $\hat{r}_{t+1} = \frac{r_{t+1}}{1-\rho_2}$ and element-wise multiplication $\odot$. Typical parameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\gamma = 0.01$ and

$\delta = 1 \cdot 10^{-8}$. Adam adapts the learning speed according to the moments, which results in faster training.

2.6.6 Backpropagation

Backpropagation is a way to update the weights $\mathbf{w}$ in the neural network during training. A neural network can be viewed as a series of functions, where the output of one function in one layer becomes the input of the function in the next layer. The network in figure 2.7 can be described by:

$$O = F^2\left(\mathbf{w}^2 F^1(\mathbf{w}^1 x + \mathbf{b}^1)\right) + \mathbf{b}^2,$$

where $O \in \mathbb{R}$ is the predicted label, $F^l(x, \mathbf{w}^l, \mathbf{b}^l)$ is the activation function of layer l with weights $\mathbf{w}^l$ and bias $\mathbf{b}^l$. During training the weights are updated by a gradient descent method. To calculate the gradient of the network with respect to the weights $\mathbf{w}^l$ the *chain rule* is used. If a differentiable function $F(w)$ is dependent on a differentiable function $G(w)$ the chain rule states:

$$\frac{\partial F(G(w))}{\partial w} = \frac{\partial F(G(w))}{\partial G(w)} \frac{\partial G(w)}{\partial w}. \tag{2.14}$$

To use backpropagation for updating the weights, it is important that all functions in the network are differentiable, including the loss function (see section 2.6.7).

2.6.7 Loss Functions

The loss function of a neural network compares the output of the network with the ground truth label and returns the *error* or *loss* $e \in \mathbb{R}$. During training the loss function is used to optimize the weights $\mathbf{w}$ of the network using backpropagation 2.6.6. To use backpropagation it is important that the loss function is differentiable. Depending on the task to be solved by the network there are different loss functions. Commonly used in classification tasks is the *cross-entropy loss* [Mur13]:

$$L(\bar{y}, \hat{y}) = -\sum_{c=1}^{N} \bar{y}_c \log(\hat{y}_c),$$

where $\bar{y} \in \{0, 1\}^N$ is a vector indicating the true class of an object and $\hat{y} \in \mathbb{R}^N$ is the output of the network assigning probabilities to each of the N classes. A loss function used in regression tasks is the *mean squared error* (MSE) [Mur13]:

$$L(\bar{y}, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (\bar{y}_i - \hat{y}_i)^2 .$$

The losses used to train networks in this thesis compare rotations in different representations (see page 17), compare translations (see page 17) or compare point clouds $\hat{\mathcal{M}}$ and $\bar{\mathcal{M}}$ transformed by the estimated and ground truth poses $\hat{\mathbf{P}}$ and $\bar{\mathbf{P}}$ (see page 19).

Loss to compare Translations:

To compare the ground truth translation $\bar{\mathbf{t}}$ to the estimated translation $\hat{\mathbf{t}}$ the euclidean distance is used to calculate the difference:

$$L_{transl}(\bar{\mathbf{t}}, \hat{\mathbf{t}}) = \left\| \bar{t} - \hat{t} \right\|_2 . \tag{2.15}$$

Losses to compare Rotations:

To compare the similarity of two unit quaternions $\mathbf{q}_1$, $\mathbf{q}_2$ the function [Ano]

$$\varphi_1(\mathbf{q}_1, \mathbf{q}_2) = \arccos\left(2 \left\langle \mathbf{q}_1, \mathbf{q}_2 \right\rangle^2 - 1\right) \tag{2.16}$$

is used. $\varphi_1(\mathbf{q}_1, \mathbf{q}_2)$ is the angle θ of the rotation that is required to go from the rotation $\mathbf{q}_1$ to the rotation $\mathbf{q}_2$. The square in the loss function ensures that $\varphi_1(\mathbf{q}_1, \mathbf{q}_2)$ returns the same value for antipodal points: $\varphi_1(\mathbf{q}_1, \mathbf{q}_2) = \varphi_1(\pm\mathbf{q}_1, \pm\mathbf{q}_2)$. In the implementation, the input of the arccos is clipped between $[-1 + 10^{-6}, 1 - 10^{-6}]$ to ensure that the gradient of the arccos is not $\pm\infty$.
Another metric that can be used as a loss function for unit quaternions is the euclidean distance between the two quaternions [Huy09]. Since antipodal points represent the same rotation, the minimum of the distances is used:

$$\varphi_2(\mathbf{q}_1, \mathbf{q}_2) = \min\left\{ \left\| \mathbf{q}_1 - \mathbf{q}_2 \right\|, \left\| \mathbf{q}_1 + \mathbf{q}_2 \right\| \right\}. \tag{2.17}$$

Another possible loss function for quaternions used by Wunsch et al. in [WWH97] is $\varphi(\mathbf{q}_1, \mathbf{q}_2) = \min\left\{\arccos\left(\left\langle \mathbf{q}_1, \mathbf{q}_2 \right\rangle\right), \pi - \arccos\left(\left\langle \mathbf{q}_1 \mathbf{q}_2 \right\rangle\right)\right\}$. A computationally more efficient version of that loss function is:

$$\varphi_3(\mathbf{q}_1, \mathbf{q}_2) = \arccos\left(\left|\left\langle \mathbf{q}_1, \mathbf{q}_2 \right\rangle\right|\right). \tag{2.18}$$

With the identity $\cos\theta = 2\cos^2\theta\left(\frac{\theta}{2}\right) - 1$ one can see that $2\varphi_3(\mathbf{q}_1, \mathbf{q}_2) = \varphi_1\left(\mathbf{q}_1, \mathbf{q}_2\right)$.

A trigonometric free measure for $\varphi_1(\mathbf{q}_1, \mathbf{q}_2)$ is [Ano]:

$$\varphi_4(\mathbf{q}_1, \mathbf{q}_2) = 1 - \left\langle \mathbf{q}_1, \mathbf{q}_2 \right\rangle^2 \in [0, 1]. \tag{2.19}$$

To ensure that the network has unit quaternions as output the *regularisation loss*

$$L_{reg}(\mathbf{q}) = L_\delta(\|\mathbf{q}\| - 1) \tag{2.20}$$

is used, where L_δ is the Huber loss 2.26 with $\delta = 1$. For quaternions the output of the network was normalized to be a unit quaternion before the loss was computed.

For rotation matrices $\mathbf{R}$ the output of the network needs to be orthogonalized before the loss can be computed. The following projection onto $SO(3)$ was used to orthogonalize the output $\mathbf{O} \in \mathbb{R}^{3\times3}$ of the neural networks. Let $\mathbf{O} \in \mathbb{R}^{3\times3}$ be of full rank and $\mathbf{M} = \mathbf{O}^T\mathbf{O} = \mathbf{U}\Sigma\mathbf{V}^T$. Then

$$SO(3) \ni \mathbf{R} = \mathbf{O}\mathbf{U} \begin{pmatrix} \frac{1}{\sqrt{\sigma_1}} & & \\ & \frac{1}{\sqrt{\sigma_2}} & \\ & & \frac{s}{\sqrt{\sigma_3}} \end{pmatrix} \mathbf{U}^T, \tag{2.21}$$

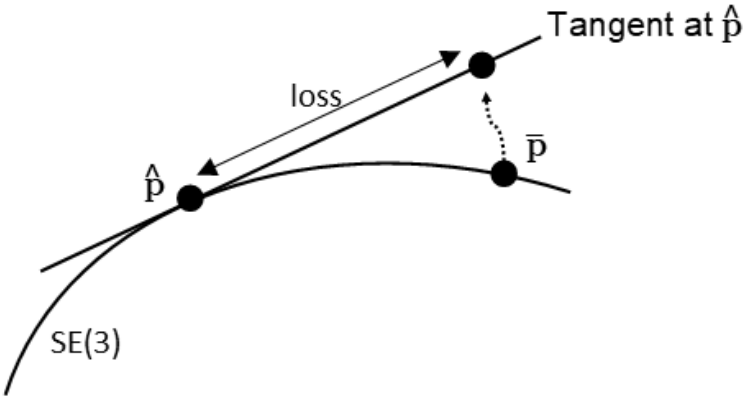


Figure 2.13: Illustration of $\varphi_{SE(3)}$.

where $\mathbf{U}$ is the left singular matrix, the σ_i are the singular values of $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^T$ and $s = \text{sign}(\det \mathbf{O})$. For $\det \mathbf{O} > 0$ the equation 2.21 simplifies to $\mathbf{R} = \mathbf{U}\mathbf{V}^T$.
This orthogonal projection of $\mathbf{O}$ onto $\mathbf{R}$ is optimal with regard to the Frobenius norm, i.e. $\mathbf{R} = \arg\min_{\tilde{\mathbf{R}} \in SO(3)} \left\| \mathbf{O} - \tilde{\mathbf{R}} \right\|_F$ [Moa02].

For rotations represented by $\mathbf{R}_1, \mathbf{R}_2 \in SO(3)$ the following loss discussed by Larochelle et al. [LMA07] was used to train the networks:

$$\varphi_5(\mathbf{R}_1, \mathbf{R}_2) = \left\| \mathbb{I} - \mathbf{R}_1 \mathbf{R}_2^T \right\|_F . \tag{2.22}$$

This loss function measures the deviation of $\mathbf{R}_1\mathbf{R}_2^T$ from the identity matrix. $\|.\|_F$ denotes the Frobenius norm.

φ_6 takes rotation matrices and uses the axis-angle representation (equation 2.9) to get the angle between them

$$\varphi_6 \left(\mathbf{R}_1, \mathbf{R}_2 \right) = \arccos \left(\left(\text{Tr} \left(\mathbf{R}_1 \mathbf{R}_2^{-1} \right) - 1 \right) / 2 \right) . \tag{2.23}$$

This loss is equivalent to φ_1 with $\varphi_1(\mathbf{q}_1, \mathbf{q}_2) = 0.5\varphi_3(\mathbf{q}_1, \mathbf{q}_2) = \varphi_6 \left(\mathbf{R}_1, \mathbf{R}_2 \right)$ where $\mathbf{q}_i$ is a corresponding quaternion to the rotation $\mathbf{R}_i,\ i \in \{1, 2\}$ [Huy09].

To compute the loss for translation and rotation at the same time Hou et al. [HMK+18] introduces a new loss function based on the geodesic distance on $SE(3)$. Hou uses axis-angle combined with the translation to represent the pose $\mathbf{p} = \{r_x, r_y, r_z, t_x, t_y, t_z\}$. For the poses $\hat{\mathbf{p}}, \bar{\mathbf{p}}$ the loss is defined as:

$$\varphi_{SE(3)}(\bar{\mathbf{p}}, \hat{\mathbf{p}}) = \text{dist}^Z_{SE(3)}(\bar{\mathbf{p}}, \hat{\mathbf{p}})^2 = \left|\left| \text{Log}^Z_{\hat{\mathbf{p}}}(\bar{\mathbf{p}}) \right|\right|^2_{Z_{\hat{\mathbf{p}}}} \tag{2.24}$$

with $\text{dist}^Z_{SE(3)}$ as geodesic distance and Log the Riemannian logarithm that maps from the Lie group $SE(3)$ to its Lie algebra $\mathfrak{se}(3)$. $\text{Log}_{\hat{\mathbf{p}}}(\bar{\mathbf{p}})$ uses parallel transport to map $\bar{\mathbf{p}}$ into the tangent space of $SE(3)$ at $\hat{\mathbf{p}}$ (see figure 2.13).[4]

[4]For more details to parallel transport see [BBN].

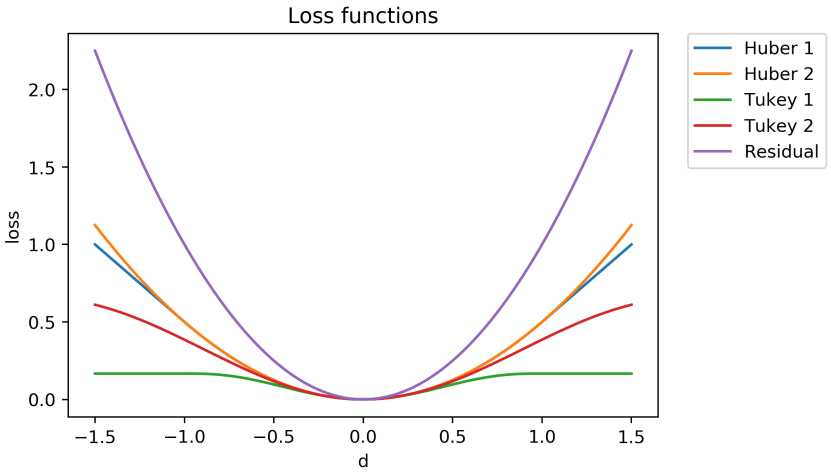


Figure 2.14: Comparison of loss functions

The figure shows the residual loss, the Huber loss for parameters 1 and 2, and the Tukey loss for parameters 1 and 2 for different distances d. The residual loss keeps increasing for larger distances and is not robust against outliers. Huber loss is increasing slower and Tukey loss flattens out for larger distances d and is thus more robust.

Losses to compare Point Clouds

The *residual loss* can be used to compare point clouds $\bar{\mathcal{M}} = \bar{\mathbf{P}}\mathcal{M}$ and $\hat{\mathcal{M}} = \hat{\mathbf{P}}\mathcal{M}$.

$$L_{res}(\bar{\mathcal{M}}, \hat{\mathcal{M}}) = \frac{1}{N} \sum_{\mathbf{x}_i \in \bar{\mathcal{M}}} \min_{\mathbf{x}_j \in \hat{\mathcal{M}}} \|\mathbf{x}_i - \mathbf{x}_j\| . \tag{2.25}$$

This loss calculates the average distance of the points $\mathbf{x}_i \in \bar{\mathcal{M}}$ to the points $\mathbf{x}_j \in \hat{\mathcal{M}}$ that are closest to them. The distance of the closest points is used because the correspondences between points in $\bar{\mathcal{M}}$ and $\hat{\mathcal{M}}$ are unknown. This loss is not robust with regard to outliers or missing points due to partial visibility (see section 4.1.2) [Hub92].

To make the residual loss more robust, the *Huber loss* [Hub92] can be used instead of the residual loss. The Huber loss for the distance $d = \|\mathbf{x}_i - \mathbf{x}_j\|$ is defined as:

$$L_\delta(d) = \begin{cases} \frac{1}{2}d^2 & \text{for } |d| \leq \delta \\ \delta|d| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases} \tag{2.26}$$

A typical value for the parameter δ is 1. With the Huber loss the residual loss is modified to

$$L_{hub}(\bar{\mathcal{M}}, \hat{\mathcal{M}}) = \frac{1}{N} \sum_{\mathbf{x}_i \in \bar{\mathcal{M}}} \min_{\mathbf{x}_j \in \hat{\mathcal{M}}} L_\delta \left(\|\mathbf{x}_i - \mathbf{x}_j\| \right) . \tag{2.27}$$

The *Tukey loss* is another way to make the residual loss more robust. With the distance d Tukey's biweight loss function is defined as:

$$L_c\left(d\right) = \begin{cases} \frac{c^2}{6}\left[\left(1 - \left(\frac{d}{c}\right)^2\right)^3\right] & \text{, if } |d| \leq c \\ \frac{c^2}{6} & \text{, otherwise.} \end{cases} \tag{2.28}$$

The parameter c depends on the range of the distances d that are put in the loss function. With Tukey's biweight function the residual loss becomes

$$L_{tukey}(\bar{\mathcal{M}}, \hat{\mathcal{M}}) = \frac{1}{N} \sum_{\mathbf{x}_i \in \bar{\mathcal{M}}} \min_{\mathbf{x}_j \in \hat{\mathcal{M}}} L_c \left(\|\mathbf{x}_i - \mathbf{x}_j\| \right). \tag{2.29}$$

Figure 2.14 shows the residual loss in comparison with the Tukey and Huber losses.

To estimate the pose, the loss of the network is a weighted sum of the previous losses. These weights are updated during the training to increase performance. A typical loss function that was used in this thesis is

$$\mathcal{L}(\bar{\mathbf{P}}, \hat{\mathbf{P}}, \mathcal{M}) = w_{rot} \cdot \varphi_1(\bar{\mathbf{q}}, \hat{\mathbf{q}}) + w_{transl} \cdot L_{transl}(\bar{\mathbf{t}}, \hat{\mathbf{t}}) + w_{reg} \cdot L_{reg}(\hat{\mathbf{q}}), \tag{2.30}$$

with the weights $w_{rot} = w_{transl} = 1$ and regularisation weight $w_{reg} = 0.1$.
For a comparison of the performance of the different loss functions see chapter 5 with the evaluation.

2.7 PointNet

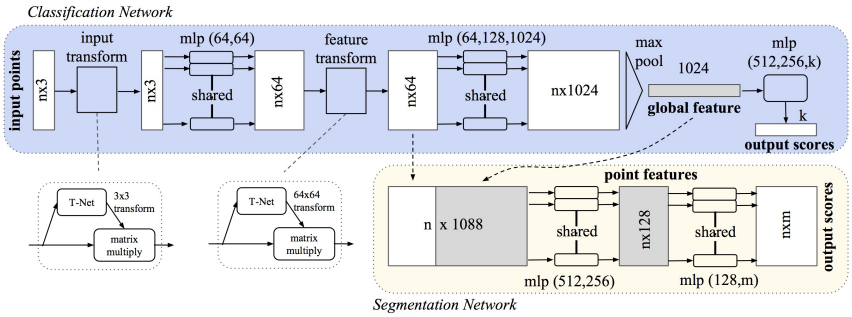


Figure 2.15: Architecture of PointNet

The network has n points as input. To ensure that the network can classify objects independently of their pose the input is then transformed by the T-Net. Then two convolutional layers with 64 filters of size 1×1 are applied to the points separately and these features are then again transformed by a T-Net. Then more 1×1 convolutions are applied with an increasing number of filters. To deal with the permutation invariance of point clouds the symmetric function max pooling returns the global features, for each of the 1024 filters one point with the maximum value. These are then fed into dense layers to get the output score for classification. For the segmentation, local and global features are concatenated and then fed into 1×1 convolutional layers to finally get for each of the n points scores for the m semantic subcategories. The image is taken from [QSMG16], p. 3.

The idea of the network structure used to deal with the permutation invariance of point clouds comes from PointNet by Qi et.al. [QSMG16]. PointNet has as input a set of points and then classifies the object and does part segmentation, where different parts of an object are labelled, and semantic segmentation, where each point is labelled. Figure 2.15 shows the architecture of PointNet.

To deal with the permutation invariance of point clouds PointNet uses a permutation invariant function $g : \mathbb{R}^K \times \cdots \times \mathbb{R}^K \to \mathbb{R}$ with the property:

$$g(x_1, \ldots, x_n) = g(x_{\pi(1)}, \ldots, x_{\pi(n)})$$

for any permutation $\pi : (1, 2, \ldots, n) \to (1, 2, \ldots, n)$. Examples for permutation invariant operators are addition, multiplication, taking the maximum of a set or the L^{10}-norm. The L^p-norm of a vector $\mathbf{x} \in \mathbb{R}^n$ is $(\sum_{i=1}^{n} x_i^p)^{\frac{1}{p}}$. Section 5.2.2 evaluates the performance of networks with different functions g, like the L^{10} norm.

3 Datasets

This chapter introduces the datasets used to train the networks. Section 3.1 shows, how the synthetic data was generated and section 3.2 introduces the Linemod dataset.

3.1 Synthetic Data

In the synthetic datasets, the source point clouds were centred to have zero mean and rescaled so that they were in the unit ball $\mathbf{x}_i \in \mathcal{M} : \|\mathbf{x}_i\| \leq 1$. The point clouds were rotated with uniformly distributed unit quaternions $\mathbf{q}$ (see procedure 3.31) and divided by some rescaling factor like 1.2. After that, the point clouds were translated with a uniformly distributed translation so that the translated point clouds were still in the unit ball. This results, for a scaling factor of 1.2, in a maximal translation of $\|\mathbf{t}\| \approx 0.3$. Then, Gaussian noise was added to the target point clouds. After that, points were left out or random points were added. Figure 3.16 illustrates the augmentation pipeline on a $2D$ point cloud.
To determine the architecture of the network, the Stanford bunny (figure 3.17) was used [Tur]. The performance of the network was evaluated by training with 13 of the objects from the Linemod dataset [HLI+13]. The fish point cloud in figure 3.16 was used to train and evaluate the network on $2D$ data.

The unit quaternionsg are uniformly sampled from the upper hemisphere of the 4-dimensional sphere $S^3_+ = \{\mathbf{q} = (q_1, q_2, q_3, q_4) \in \mathbb{R}^4 : \|\mathbf{q}\| = 1, q_1 \geq 0\}$ as follows:

$$\begin{aligned} &1.\ \mathbf{x} \sim \mathcal{N}(0, \mathbf{1}_4) \\ &2.\ \mathbf{q} = \frac{\mathbf{x}}{\|\mathbf{x}\|} \\ &3.\ \mathbf{q} = \operatorname{sign}(q_1) \cdot \mathbf{q}, \end{aligned} \tag{3.31}$$

where $\mathcal{N}(0, \mathbf{1}_4)$ is a multivariate normal distribution with mean 0 and the unit matrix as covariance matrix. Since $\mathbf{q}$ and $-\mathbf{q}$ represent the same rotation (see section 2.5), only quaternions with $q_1 \geq 0$ are returned. This approach to sample random unit quaternions is very efficient [Pol00].

3.2 Linemod-Dataset

The Linemod dataset by Hinterstoisser et al. [HLI+13] contains 15 household objects (for examples, see figure 3.18). Provided are the object meshes with surface normal and colour. For each object more than 1100 RGB and depth images, taken with a Kinect, are provided together with the ground truth poses and the bounding boxes. The objects are texture-less with diameters ranging between 9cm and 30cm, and the distance of the objects from the camera is between 60cm and 110cm. The azimuth range of the rotation is between 0° and 360°, and the elevation is between 3° and 90°. The objects in the pictures are surrounded by clutter and in some test cases the objects are occluded by other objects. Examples of the Linemod images can be seen in figure 3.19.

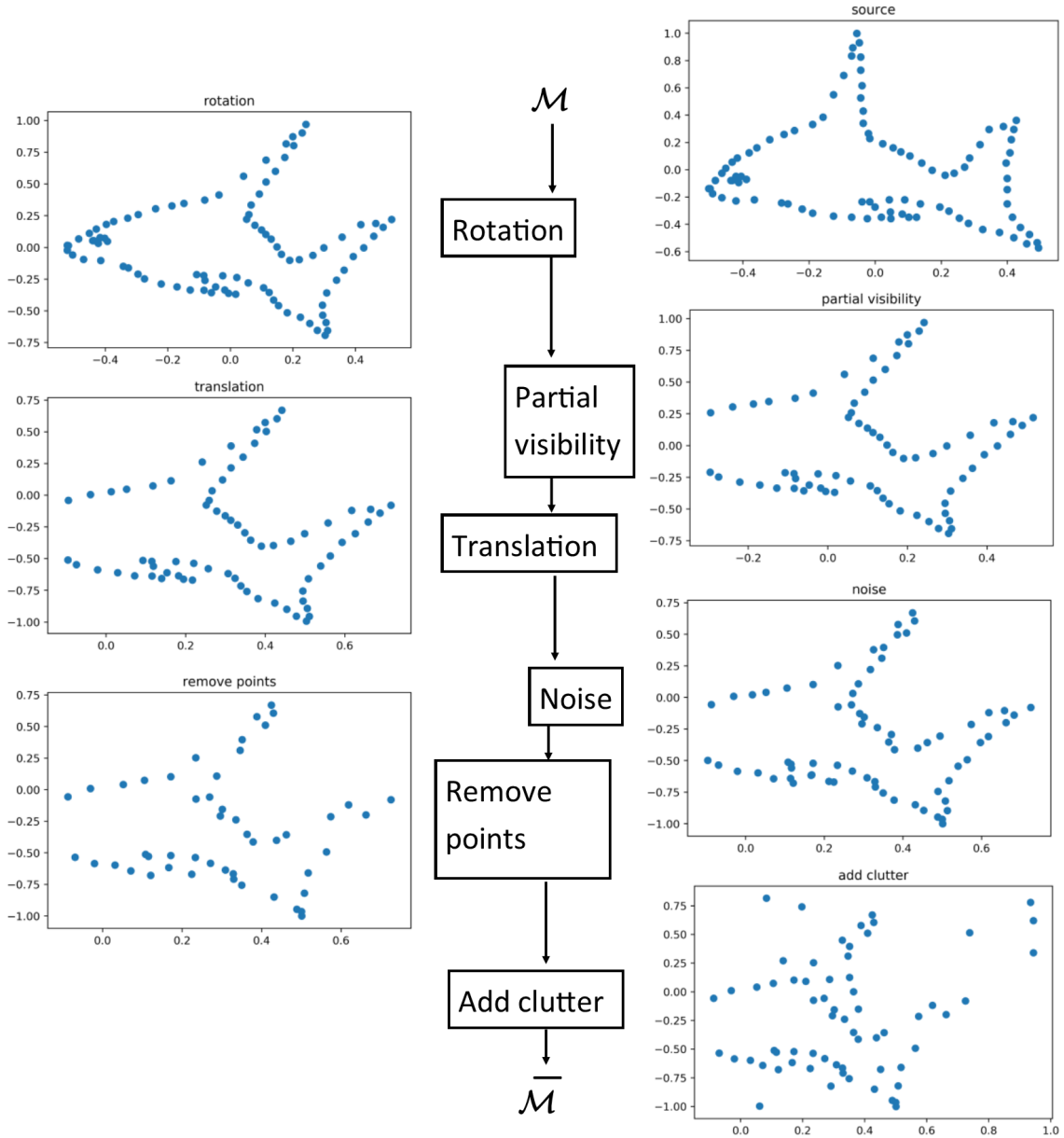


Figure 3.16: Augmentation process for synthetic data

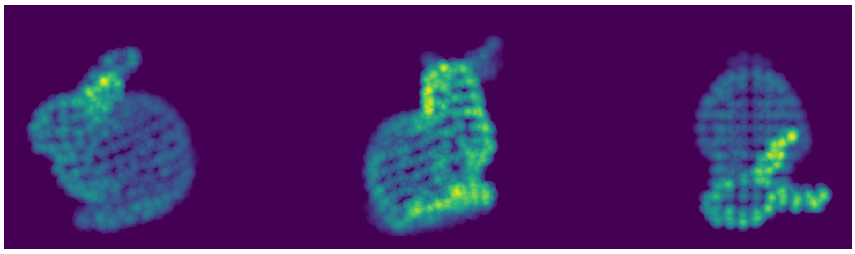


Figure 3.17: Point cloud of the bunny

Figure 3.18: Some objects of the Linemod dataset

Selection of objects in the Linemod dataset [HLI+13]. From left to right: can, cat, driller, glue, phone, ape, bench vise and egg box.

The Linemod dataset was taken from the SIXD Challenge 2017[5] organized at the 3rd International Workshop on Recovering 6D Object Pose at ICCV 2017. The challenge also features other datasets in a unified format.

During training the depth image was cropped around the bounding box of the object. This is a two-stage approach, where it is assumed that another method provides the position of the object in the image. The bounding box provided by the dataset was increased by 10 pixels in each direction, to account for errors of the method that provides the bounding box. Examples for two-stage approaches are given in[RHGS15], [LPY+17]. An example for the provided bounding box and the resulting point cloud can be seen in figure 3.20.

[5] http://cmp.felk.cvut.cz/sixd/challenge˙2017/

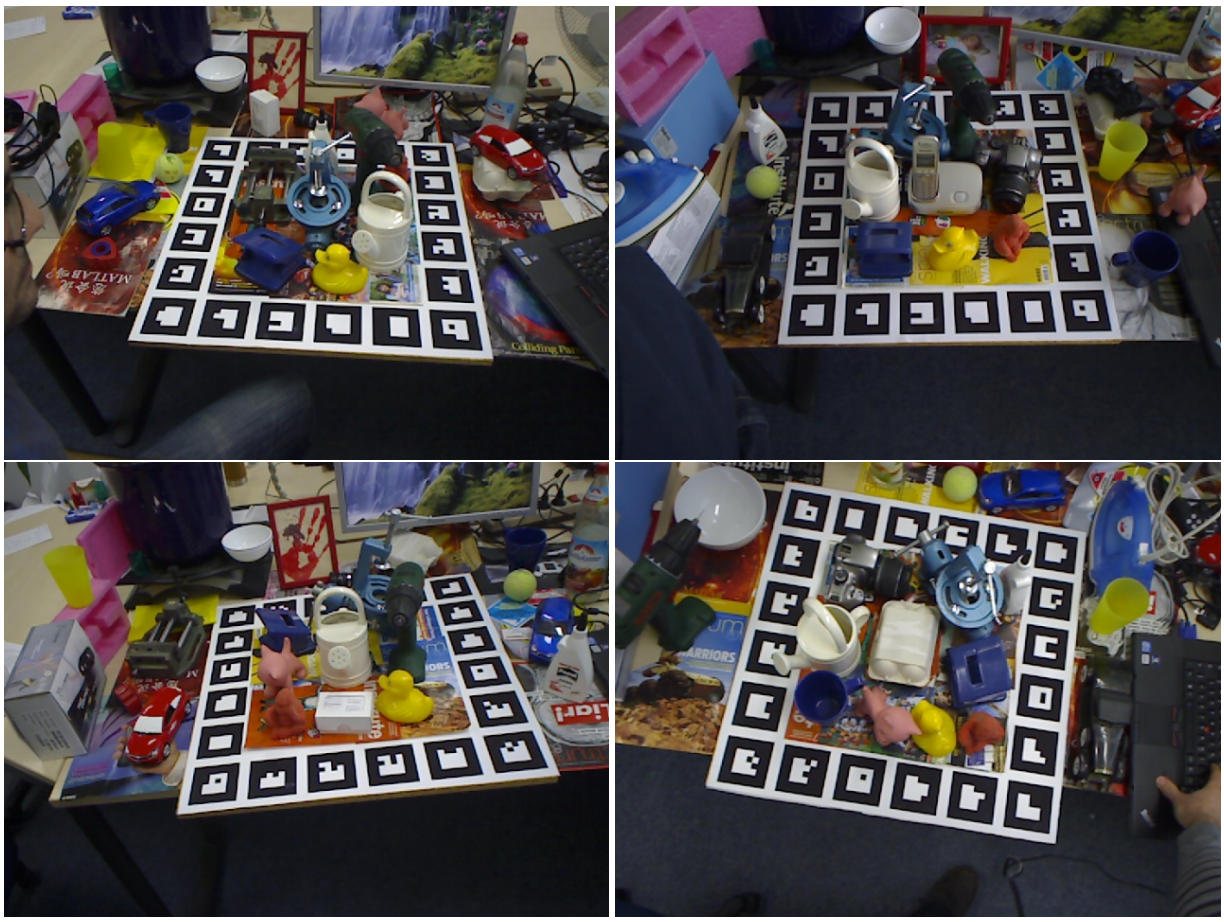
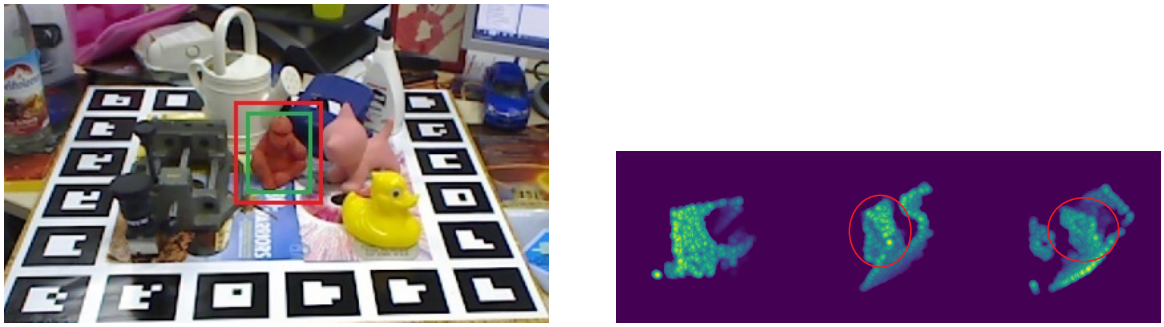


Figure 3.19: Some Linemod images

Selection of RGB images of the Linemod dataset by Hinterstoisser et al. [HLI+13].



(a) Bounding box

(b) Point cloud of the ape in figure 3.20a

Figure 3.20: Bounding box around ape and corresponding point cloud

The figure to the left shows an image from the Linemod dataset [HLI+13] with provided bounding box in green and the bounding box that was used to create the target point cloud. In figure 3.20b the ape is seen from the front in the right point cloud and from the back in the middle one.

4 Problem Description

Given are two point clouds: the source $\mathcal{M} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and target or ground truth $\bar{\mathcal{M}} = \{\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_N\}$. The transformation $\bar{\mathbf{P}} = \left[\bar{\mathbf{R}}, \bar{\mathbf{t}}; 0, 1\right]$ between them is given by $\bar{\mathcal{M}} = \bar{\mathbf{P}}\mathcal{M}$[6]. The goal of this thesis is to find a function that has source $\mathcal{M}$ and target $\bar{\mathcal{M}}$ as inputs and returns an estimate $\hat{\mathbf{P}}$ close to $\bar{\mathbf{P}}$. This function should be bi-lipschitz to account for noise (see section 4.1.1), occlusions and clutter (see section 4.1.2) in the point clouds.

Definition 4.1. Given two metric spaces (X, d_Y), (Y, d_Y) a function $f : X \to Y$ is called *bi-lipschitz* if there is a $K > 1$, so that for all $x_1, x_2 \in X$ it holds [LP01]:

$$\frac{1}{K} d_X(x_1, x_2) \leq d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2). \tag{4.32}$$

In this thesis the function is a neural network that returns several values for the pose: three values for the translation and, dependent on the representation that is chosen for the rotation, four values for a quaternion $\hat{\mathbf{q}}$, nine values for a rotation matrix $\hat{\mathbf{R}}$ or three values for a rotation vector $\hat{\mathbf{r}}$ (see section 2.5).

4.1 Challenges in Pose Estimation

This section gives some insights in the challenges of pose estimation and some possible solutions for them. Some challenges in pose estimation are due to the limitations of the hardware (see section 4.1.1), due to the difficulties in the dataset (see sections 4.1.2, 4.1.5) or due to the pose parametrisation (see section 4.1.4).

4.1.1 Noise in the Point Cloud

Because of the precision of the sensor that takes the depth image, the target point cloud $\bar{\mathcal{M}}$ generated with this depth image has some noise on its points. This means that the points in the source point cloud are transformed with $\bar{\mathbf{x}}_i = \mathbf{P}\mathbf{x}_i + \varepsilon_i$, where $\varepsilon_i \in \mathbb{R}^3$ is some random vector. This noise is assumed to be Gaussian and it is accounted for in the training of the network by adding Gaussian noise to the input points of the target point cloud.

4.1.2 Occlusions and Clutter

Not every point $\mathbf{x} \in \mathcal{M}$ of the source point cloud has a corresponding point $\bar{\mathbf{x}} \in \bar{\mathcal{M}}$ such that $\bar{\mathbf{x}} = \mathbf{P}\mathbf{x}$ because the view to the point $\mathbf{x}$ can be occluded by other objects or the object itself. On the other hand there can also be points $\bar{\mathbf{x}} = \mathbf{P}\mathbf{x}$ without corresponding $\mathbf{x} \in \mathcal{M}$ due to surrounding objects that are also captured in the depth map by the camera. Figure 4.21 shows an RGB image with occlusions and clutter. The problem of occlusion and clutter can be minimized by robust losses like Huber loss and Tukey loss (see section 2.6.7) that reduce the impact of outliers and by training on point clouds with occlusions and clutter.

[6] This equality only holds exactly in very ideal and unrealistic cases. See section 4.1.

Figure 4.21: RGB image with occlusion and clutter

Picture from the Linemod dataset [HLI+13]. The blue hole punch is occluded by the cat and the ape. The backside of the hole punch is covered by itself. The image contains apart from the hole punch various other objects that are clutter.

4.1.3 Ambiguous Objects

Some objects have poses that are not distinguishable with the available data (see figure 5.23). This can lead to issues in estimating the pose during the training of a neural network, since all possible correct poses need to be taken into account. This thesis does not focus on the detection of ambiguous objects and, thus the trained networks performed worse for ambiguous objects (see figure 5.30). For approaches how to deal with ambiguous objects see for example [Ano], [LE06] and [KCJ+].

4.1.4 Representation of the Rotation

With rotation matrices $\mathbf{R} \in SO(3)$, the network $f_{SO(3)}$ that returns the pose maps for the translations to $\mathbb{R}^3$ and for the rotation to $SO(3)$:

$$f_{SO(3)} : \mathbb{R}^{N\times 3} \times \mathbb{R}^{N\times 3} \to \mathbb{R}^3 \times SO(3).$$

$\mathbf{R} \in SO(3)$ has 6 redundant parameters for a rotation with 3 degrees of freedom. This leads to problems during training and the network does not converge easily.
Quaternions solve the problem of over-parametrisation and it holds:

$$f_{\mathbf{q}} : \mathbb{R}^{N\times 3} \times \mathbb{R}^{N\times 3} \to \mathbb{R}^3 \times S^3$$

However, because antipodal points represent the same rotation, this can cause problems around the equator of the smooth and compact manifold S^3 with $q_1 = 0$. At the equator the two hemispheres that cover the group of rotations $S^3_+ = \{\mathbf{q} \in S^3 : q_1 \geq 0\}$ and $S^3_- = \{\mathbf{q} \in S^3 : q_1 \leq 0\}$ meet. These quaternions on the equator represent rotations by 180° around some unit vector $\mathbf{u}$. The performance of the network around the equator is evaluated in chapter 5 (see figure 5.33).
In 2D the rotation can be represented with the circle group $U(1)$ without the need to

identify antipodal points. The performance of the network in $2D$ is evaluated in section 5.3.2 (see figure 5.45).
Just like quaternions, the axis-angle approach with only 3 parameters also does not have the issue of over-parametrisation. But this representation is also ambiguous, since one can rotate around the vector $\mathbf{u}$ and $-\mathbf{u}$.

4.1.5 Small Object Diameter compared to Translation

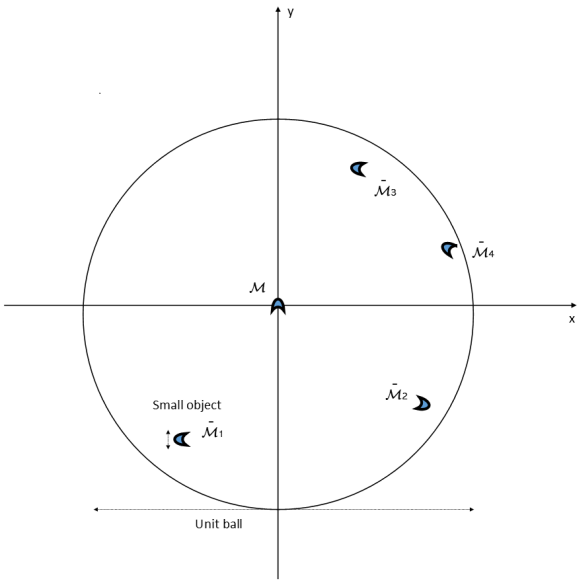


Figure 4.22: Source and multiple targets in the unit ball

For the Linemod dataset, the object diameters range from 9cm to 30cm, while the maximal translation is 110cm. Since the network takes point clouds that are located in the unit ball, the object and the translations are scaled down. To detect the rotation of an object, the network has to distinguish between points in the target point cloud that are (compared to the unit ball) close together. Since the target point cloud can be translated to any location in the unit ball, the network has to distinguish close points that can be anywhere in the unit ball (see figure 4.22). A network with just 1024 filters in the last convolutional layer is not able to do that and converges to just one rotation. Since the network does not need to distinguish between single points for the translation, the translation converges without problems to the ground truth.
One possible solution to the problem is training two networks. One network can focus on the translation of the object and centers the target. Another network then has as input the centred target to estimate the rotation and improve the estimate for the translation from the first network.
Another possible solution is to add the surface normals to the point clouds. Since the surface normals are invariant to translation, the neural network can estimate the translation with the points and the rotation with the normals. However, the normals are not given directly by the sensor and are approximated using a plane fit on a subset of surrounding points. This is computationally expensive and can lead to error accumulation [MN03].

5 Training and Evaluation

This chapter introduces first some metrics used to evaluate pose estimation methods in section 5.1 and shows how the architecture of the network was determined in section 5.2. Finally, the results on the synthetic and Linemod datasets are presented in section 5.3 and section 5.4.

5.1 Evaluation Metrics

This section first introduces the concept of indistinguishable views and ambiguity invariance of error metrics. Then the evaluation metrics ADD, ADI and the rotational and translational error that were used in this thesis are presented[7]. The implementation of the error metrices are provided by [HMO16][8].

Definition 5.1. An object model $\mathcal{M}$ has *indistinguishable views* if there exist poses $\mathbf{P}, \mathbf{P}'$ and a camera C such that [HMO16]:

$$d\left(v_C\left[\mathbf{P}\mathcal{M}\right],\ v_C\left[\mathbf{P}'\mathcal{M}\right]\right) \leq \varepsilon\ \wedge\ f\left(\mathbf{P},\mathbf{P}'\right) \geq \rho.$$

$v_C\left[\mathcal{M}\right] \subseteq \mathcal{M}$ is the part of the model surface that is visible from camera C, d measures the distance between the two surfaces (for example the Hausdorff distance) and ρ is the minimum distance f between the poses. The inequality $f\left(\mathbf{P},\mathbf{P}'\right) \geq \rho$ is necessary to exclude nearly identical poses where the surface distance is below ε .

An example for an object with indistinguishable views is the cup from the Linemod dataset [HLI+13] in figure 5.23.

Definition 5.2. The *ε-indistinguishable set* of poses of model $\mathcal{M}$ from pose $\mathbf{P}$ in image I is:

$$\left[\mathbf{P}\right]_{\mathcal{M},I,\varepsilon} = \left\{\mathbf{P}' : d\left(v_I\left[\mathbf{P}\mathcal{M}\right], v_I\left[\mathbf{P}'\mathcal{M}\right]\right) \leq \varepsilon\right\},$$

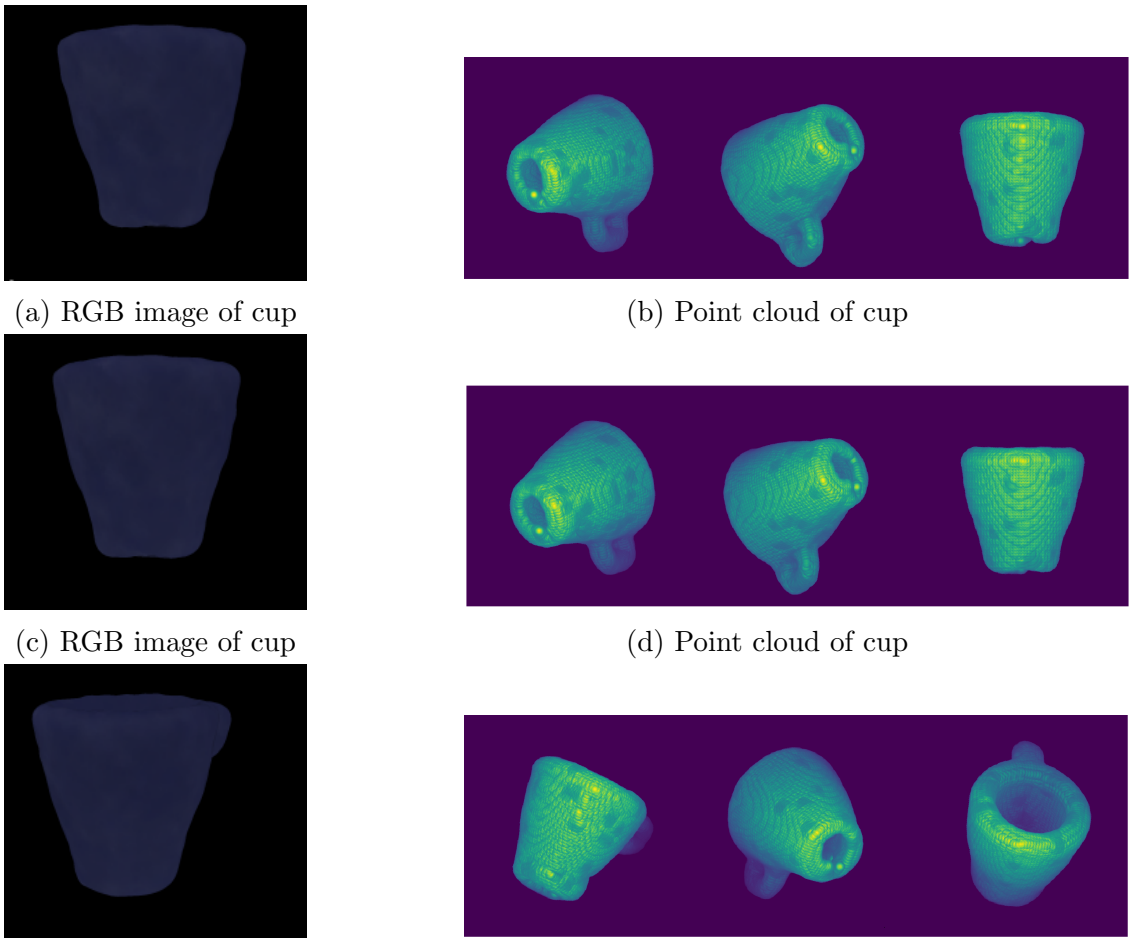
where $v_I\left[\mathcal{M}\right] \subseteq \mathcal{M}$ is the part of the model that is visible in I and d is a distance between surfaces. The tolerance ε controls how detailed the poses are distinguished.

Definition 5.3. A pose error function $e(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}, I) \in \mathbb{R}_0^+$ for an estimated pose $\hat{\mathbf{P}}$ w.r.t. the ground truth pose $\bar{\mathbf{P}}$ of an object model $\mathcal{M}$ in image I is *ambiguity-invariant* if it returns for indistinguishable poses $\hat{\mathbf{P}}$ and $\bar{\mathbf{P}}$ a similar value for the error [HMO16]:

$$\forall \hat{\mathbf{P}}' \in \left[\hat{\mathbf{P}}\right]_{\mathcal{M},I,\varepsilon},\ \forall \bar{\mathbf{P}}' \in \left[\bar{\mathbf{P}}\right]_{\mathcal{M},I,\varepsilon} : e(\hat{\mathbf{P}}', \bar{\mathbf{P}}') \approx e(\hat{\mathbf{P}}, \bar{\mathbf{P}}).$$

[7] Further evaluation metrics like the Visible Surface Discrepancy or the Complement over Union can be found in [HMO16].

[8] https://github.com/thodan/sixd_toolkit, accessed 01.08.2018



(a) RGB image of cup

(b) Point cloud of cup

(c) RGB image of cup

(d) Point cloud of cup

(e) RGB image of cup

(f) Point cloud of cup

Figure 5.23: Indistinguishable views of a cup

The left images show an RGB image of the cap from the Linemod dataset by Hinterstoisser et al. [HLI+13]. The right images show three views of the corresponding point cloud. The poses in figure 5.23a and 5.23c are not distinguishable by just the images alone. In 5.23e the handle of the cup is visible to the right and that pose is distinguishable.

Average Distance of Model Points (ADD)

ADD was proposed by Hinterstoisser et al. [HLI+13] and is now a very common method to evaluate estimated poses. It is for example used in [HZL+15],[KTD+16],[KMT+17] and [KMT+16]. For an object model $\mathcal{M}$ with no indistinguishable views the ADD error is calculated by [HLI+13]:

$$e_{ADD}(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}) = \underset{\mathbf{x} \in \mathcal{M}}{\text{avg}} \left\| \bar{\mathbf{P}}\mathbf{x} - \hat{\mathbf{P}}\mathbf{x} \right\|. \tag{5.33}$$

Like in [HLI+13], an estimated pose $\hat{\mathbf{P}}$ is considered as correct, if $e_{ADD}(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}) \leq 0.1 d_{\mathcal{M}}$ where $d_{\mathcal{M}}$ is the diameter of $\mathcal{M}$.
For objects with indistinguishable views, the ADD error might be high even though the estimated and true poses result in the same view and the ADI error has to be used [HLI+13]:

$$e_{ADI}(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}) = \underset{\mathbf{x}_1 \in \mathcal{M}}{\text{avg}} \min_{\mathbf{x}_2 \in \mathcal{M}} \left\| \bar{\mathbf{P}}\mathbf{x}_1 - \hat{\mathbf{P}}\mathbf{x}_2 \right\|. \tag{5.34}$$

Rotational Error and Translational Error

Rotational e_{RE} and translational e_{TE} errors of the estimated pose $\hat{\mathbf{P}} = \left[\hat{\mathbf{R}}, \hat{\mathbf{t}}; 0, 1\right]$ w.r.t. to the ground truth pose $\bar{\mathbf{P}} = \left[\bar{\mathbf{R}}, \bar{\mathbf{t}}; 0, 1\right]$ are independent of the model object $\mathcal{M}$ and are measured as [HMO16]:

$$e_{TE}\left(\hat{\mathbf{t}}, \bar{\mathbf{t}}\right) = \left\| \bar{\mathbf{t}} - \hat{\mathbf{t}} \right\|_2, \tag{5.35}$$

$$e_{RE}\left(\hat{\mathbf{R}}, \bar{\mathbf{R}}\right) = \arccos\left(\left(\text{Tr}\left(\hat{\mathbf{R}}\bar{\mathbf{R}}^{-1}\right) - 1\right)/2\right) \cdot 180/\pi. \tag{5.36}$$

e_{RE} is the smallest angle in degrees that is necessary to rotate from $\hat{\mathbf{R}}$ to $\bar{\mathbf{R}}$ calculated using the axis-angle representation in equation 2.9. e_{RE} and e_{TE} are not ambiguity-invariant and therefore, are not useful for ambiguous object models $\mathcal{M}$ [HMO16].

5.2 Determining the Architecture

In this section, networks of different architecture were trained on the bunny point cloud with 452 points in figure 3.17. The training data and training epochs were identical for all the trainings. Different loss functions for the rotation are evaluated in section 5.2.1 and different pooling layers in section 5.2.2.
The networks were trained using Python 3.6.2[9] and TensorFlow 1.8.0[10], with wrapper functions for TensorFlow layers from PointNet[11]. The utility functions from PointNet[12] were used to load and draw the point clouds. For quaternions, the pyquaternion package[13] was used and for the axis-angle representation geomstats[14]. The scripts to evaluate the

[9] www.python.org, accessed 20.09.2018
[10] www.tensorflow.org, accessed 11.04.2018
[11] https://github.com/charlesq34/pointnet, accessed 11.04.2018
[12] https://github.com/charlesq34/pointnet/blob/master/utils/pc_util.py, accessed 11.04.2018
[13] kieranwynn.github.io/pyquaternion/, accessed 20.04.2018
[14] https://github.com/geomstats/geomstats, accessed 04.07.2018

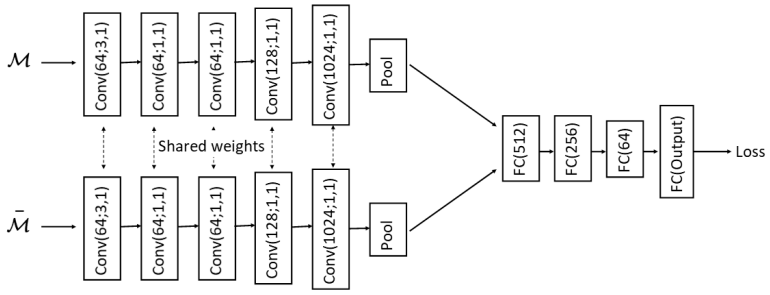


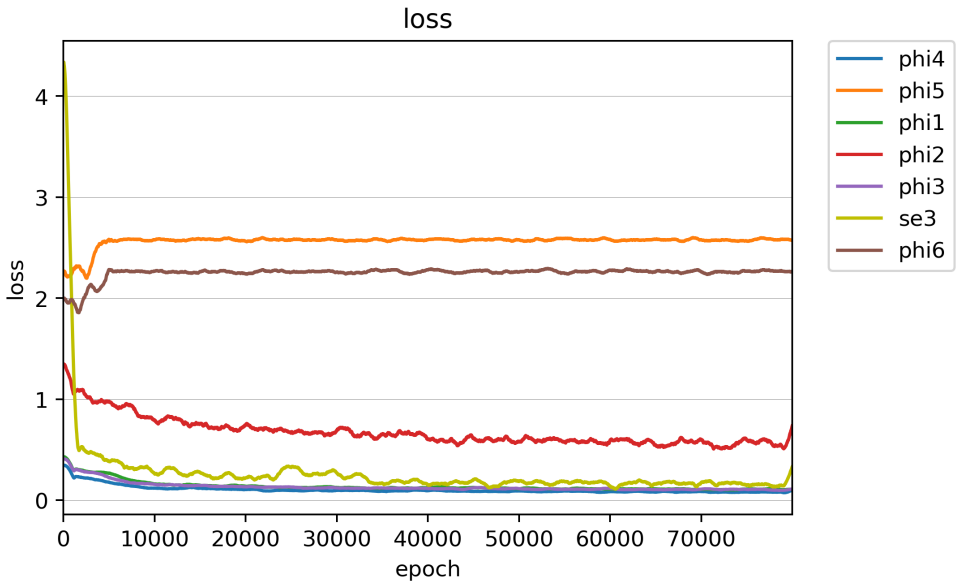
Figure 5.24: Neural network architecture used in this thesis

performance of the networks were taken from the SIXD toolkit[15].
The architecture used in this thesis (see figure 5.24) is a Siamese network [CHL05] with shared weights in the convolutions and has the source and target point clouds $\mathcal{M}, \bar{\mathcal{M}}$ as inputs. Then convolutional filters with ReLU activation (see equation 2.12) are applied to the points with an increasing amount of filters. The first convolutional layer 64 filters of size 3×1 for the $3D$ input points, the following convolutional layers have all a filter size of 1×1. With these filter sizes, each of the points in the source and target point cloud are dealt with identically and independently of each other. The pooling layer calculates for each of the 1024 filters one output and it is the first layer where each point is handled differently. In case of max pooling, the output of the pooling layer is in each filter the highest value of the points. In the case of L^1 pooling the output is in each filter the average over the values[16]. Then, the outputs of the pooling layers are concatenated and fed into fully connected layers (see section 2.6.1) with ReLU activation functions. The number of outputs of the network depends on the chosen representation for the rotation: 12 for rotation matrices $\mathbf{R}$, 7 for quaternions $\mathbf{q}$ and 6 for the rotation vector $\mathbf{r}$. Where 3 of the outputs are to estimate the translation and the others estimate the rotation.

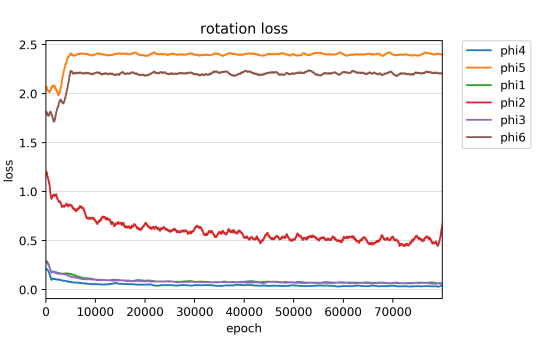
5.2.1 Evaluating Different Rotation Losses

In this section, the performances of the different rotation losses in section 2.6.7 are evaluated. Each network was fed with identical data for an identical amount of epochs and only the output layer and the loss functions were changed. Figure 5.25 shows how the different loss functions perform during training. It can be seen that the networks with an output $\mathbf{R} \in SO(3)$ for the rotation do not learn well, due to the over representation of the rotation. The networks that are using quaternions and the $SE(3)$ loss all train well, except φ_2 (see equation 2.17) where the loss reduces slower than the others.
In figure 5.26, one can see the performance of the different loss functions on the test set. The networks trained with the $SE(3)$ loss gets 90.9% of the poses correct. φ_1 and φ_3 get around 72% of the poses correct- φ_4, a trigonometric approximation of φ_1, gets around 37% correct estimated poses and the other loss functions have only very little correct poses, because the networks did not converge to the ground truth (see figure 5.25).

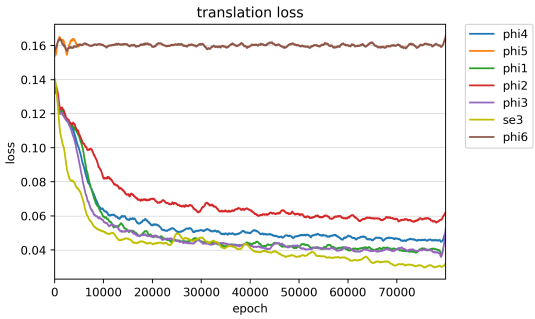
[15] https://github.com/thodan/sixd_toolkit, accessed 01.08.2018
[16] For other pooling layers and their performance see section 5.2.2.



(a) Change of the loss during training

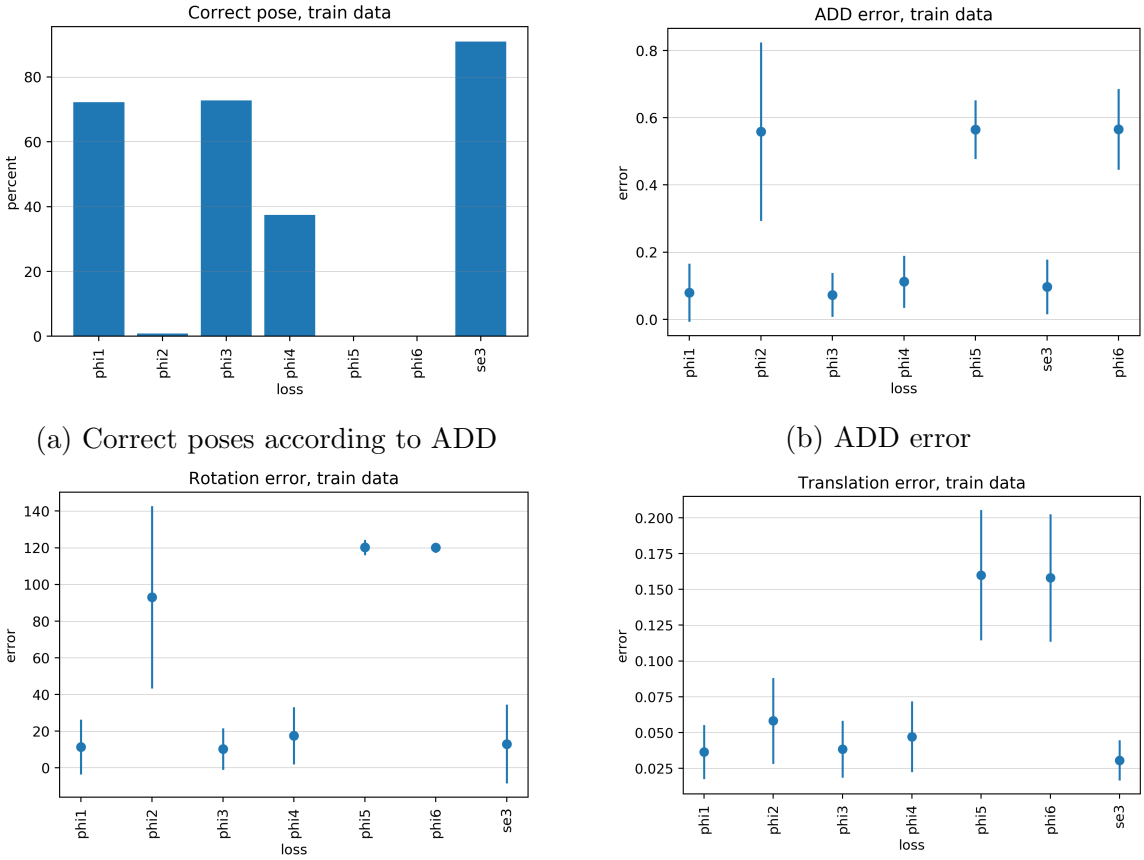


(b) Change of the rotation loss during training



(c) Change of the translation loss during training

Figure 5.25: Comparison of loss for different rotation-losses



(a) Correct poses according to ADD

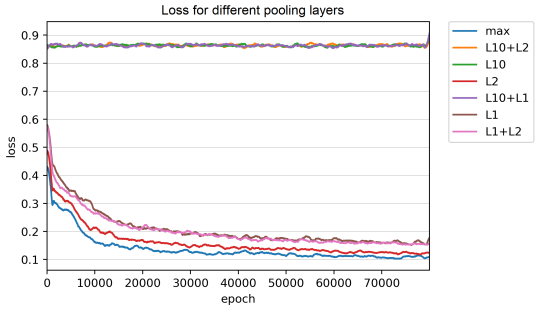
(b) ADD error

(c) Rotation error during evaluation

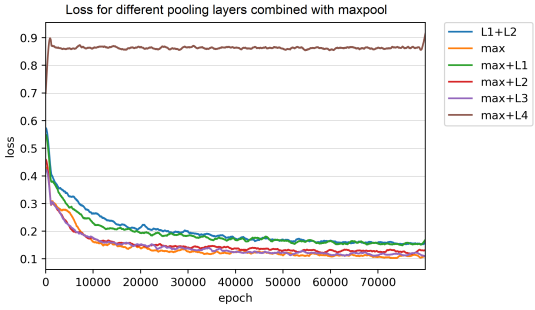
(d) Translation error during evaluation

Figure 5.26: Evaluation of the different rotation-losses

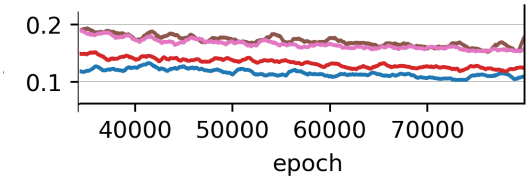
Evaluation metrics for different rotation-losses on networks trained with the bunny (see 3.17). The parameters used to sample the test data were the same as the parameters for the training data.



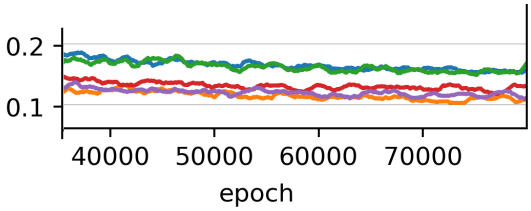
(a) Training of networks with different pooling layers



(b) Training of networks with max pooling combined with other pooling



(c) Close-up of the figure above



(d) Close-up of the figure above

Figure 5.27: Comparison of loss with different pooling layers

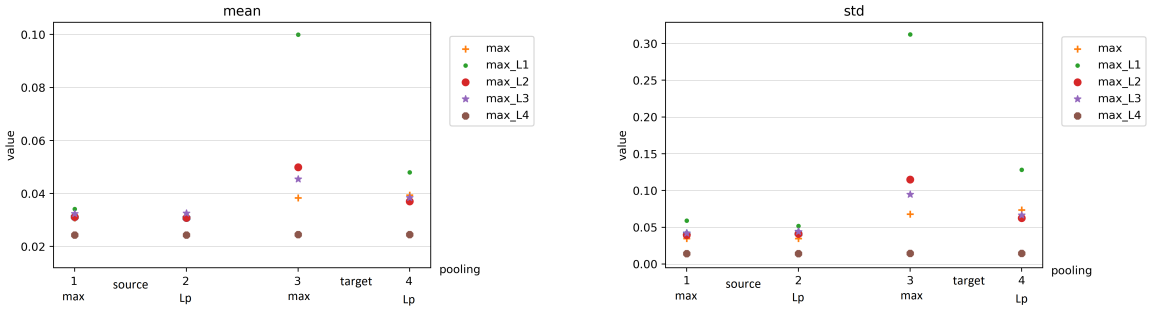
The figure shows the performance of different pooling layers. The networks were trained on the bunny point cloud 3.17 with the loss function described in equation 2.30. In figure 5.27a the networks with $L^1$0+L^2, $L^1$0+L^2 and $L^1$0+L^1 pooling did not converge and the network with only max pooling performed the best.

5.2.2 Evaluating Different Poolings

PointNet uses as permutation-invariant function the max pooling operator (see section 2.7). In this section, several permutation-invariant functions are benchmarked for their performance. The functions are max pooling, different norms like the L^1, L^2, L^{10}-norm and combinations of them. Max pooling only returns the highest value in the filter, while the other permutation-invariant functions return the norm over all the values in the filter. For the combination of the permutation-invariant functions, the number of filters in the last convolutional layer was adjusted, so that the output of the permutation-invariant layer is always 1024. That means the number of filters was reduced to 512 and the outputs of the two different permutation-invariant functions were concatenated. A combination of max pooling and the L^2-norm returns then for each of the 512 filters the highest value and also the L^2-norm for each of the filters, a total of 1024 values, which are then fed to the dense layer.

To see how much the different permutation-invariant functions contribute to the output of the network, the distributions of the absolute weights in the first fully connected layer after the pooling are plotted in figure 5.28. Since the source was the same, the weights in 1 and 2 are similar. In figure 5.28a, we see that the max pooling layer in 3 generally has higher weights than the norm in 4, i.e. the max pooling affects the decision of the network more than the pooling with the norm. The only exception there is when max pooling is combined with the L^4 norm, a combination that did not converge (see figure

5.27b). The means of the combined pooling layer with L^3 and max pooling are the closest together (apart from maxpool+L^4) which means that L^3-pooling contributes the most to the output of the network, compared to the other norm poolings.
The standard deviation seen in figure 5.28b shows that the weights for the filters differ more in max pooling than for the other pooling functions. This makes sense, since max pooling returns only the value of one point, whereas the other pooling layers take the norm over all values in one filter.
Since the network with max pooling had the lowest loss in the evaluation, max pooling was chosen as pooling layer for training the networks.



(a) Mean of the weights

(b) Standard deviation of the weights

Figure 5.28: Statistics on the weights of the first dense layer for different pooling layers

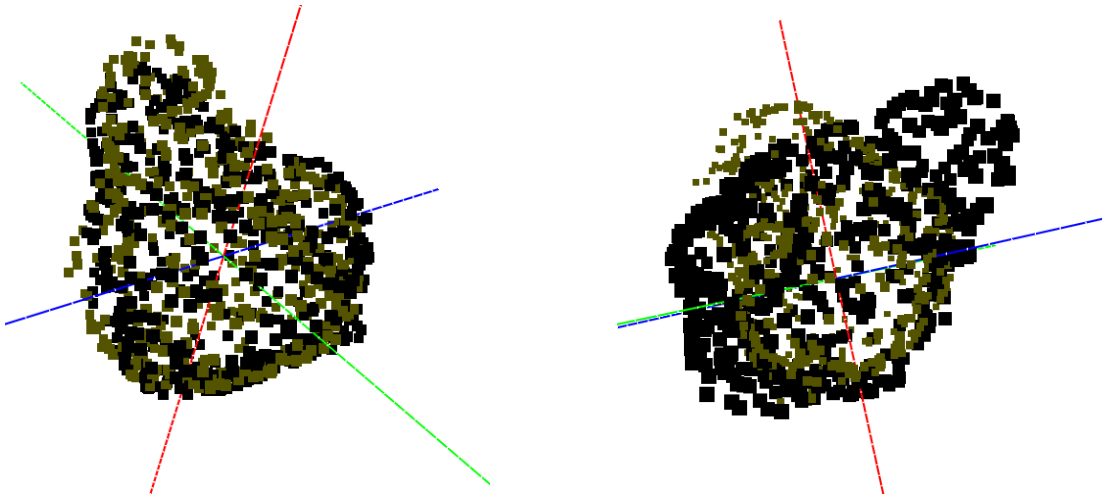
Statistics on the absolute value of the weights of the first fully connected layer after the pooling layer. 1 and 2 are the statistics of the weights for the source point cloud, where 1 is for the max pooling, 2 is for the L^p-norm (except for the + where it is max pooling). 3 and 4 are similarly for the target point cloud, with 3 for max pool and 4 for the L^p-norm.

5.3 Evaluation on Synthetic Data

In this section the network is first trained and evaluated on $3D$ point clouds and then in section 5.3.2 the networks are trained with $2D$ data.

5.3.1 $3D$ Data

The architecture used here is with max pooling and with the loss in equation 2.30. The network was trained for 80000 epochs using Adam (section 2.6.5) with learning rate 0.005. Each epoch consisted of 10 batches of size 32. The translation weight was gradually decreased during training from $w_{transl} = 1$ to $w_{transl} = 0.25$ to focus on improving the rotation error. The networks were trained on 13 of the Linemod objects with 512 points per object. Two Linemod objects (bowl and cup) were left out to see if the network can generalize to objects that it has not been trained with. During training, the maximal translation of the target point cloud was $\|\mathbf{t}\| \approx 0.3$, the rotations were sampled as described in 3.31 and the maximal movement due to noise was set to 0.05.
The network was first tested with data that was sampled like the training data (see figures 5.30, 5.31). In the second test, the rotations were changed to be around the equator (see figure in 5.33) to see if the rotation parametrisation leads to higher errors here. The last



(a) Correct estimated pose (b) Incorrectly estimated pose

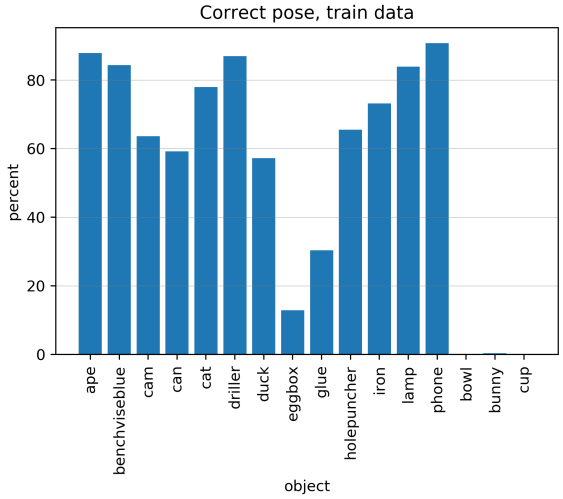
Figure 5.29: Predicted poses on the ape point cloud

The figures show the estimated pose in yellow and the ground truth pose in black.

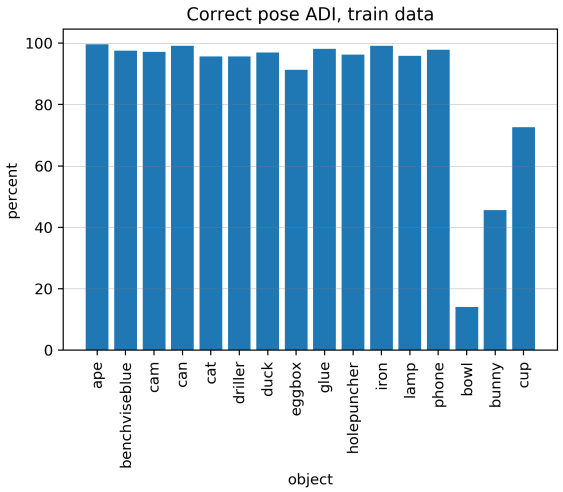
four tests evaluate how stable the network is against removing random points (figures in 5.34), removing points below a certain value to simulate partial visibility (figures in 5.36), an increase in noise (figures in 5.38) and added clutter (figures in 5.40).

Evaluation with Training Parameters

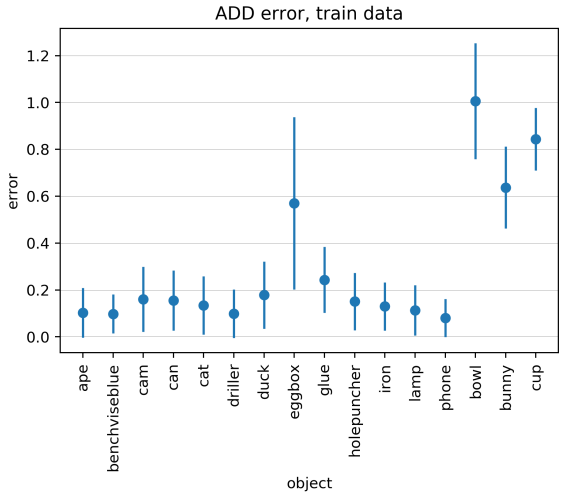
The evaluation with the ADD and ADI metric are in figure 5.30 and the rotation error and translation error in figure 5.31. In figures 5.30a and 5.31a, one can see that the non-ambiguity-invariant error metrics ADD and rotation error are high for ambiguous objects like the egg box and the glue. However, the ambiguity-invariant metric ADI indicates that 91.3% of the egg box poses and 98.1% of the glue poses are correct. The rotation error for the not ambiguous objects ranges between 10° and 20°, and the translation error is around 0.02 (see table 1).
The network does not generalize well to the objects bunny, bowl and cup that were not in the training set. The correct poses according to the ADD are very small. The correct poses according to the ADI are higher, with up to 72.6% for the cup, compared to the above 90% of the other objects (see figure 5.30b).
In figure 5.30a, a pose is considered correct if $e_{ADD}(\hat{\mathbf{R}}, \hat{\mathbf{t}}, \bar{\mathbf{R}}, \bar{\mathbf{t}}) \leq a d_{\mathcal{M}}$ where $d_{\mathcal{M}}$ is the diameter of $\mathcal{M}$ and $a = 0.1$. Figure 5.32 shows correct poses with different values of the parameter a to see how much this parameter affects the evaluation. With the ape point cloud, the percentage of correct poses increases quickly for increasing a to almost 100% for $a = 0.23$, while it increases slower for the duck point cloud.
Figure 5.29 shows an example for a correct pose and an incorrect pose on the ape. Further examples of estimated poses are in the appendix A.



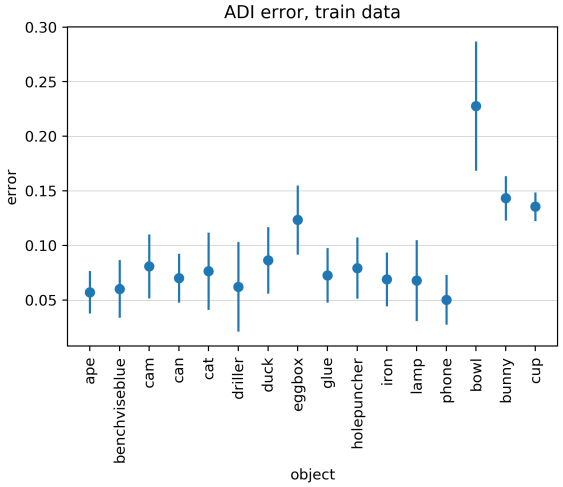
(a) Correct poses according to ADD



(b) Correct poses according to ADI



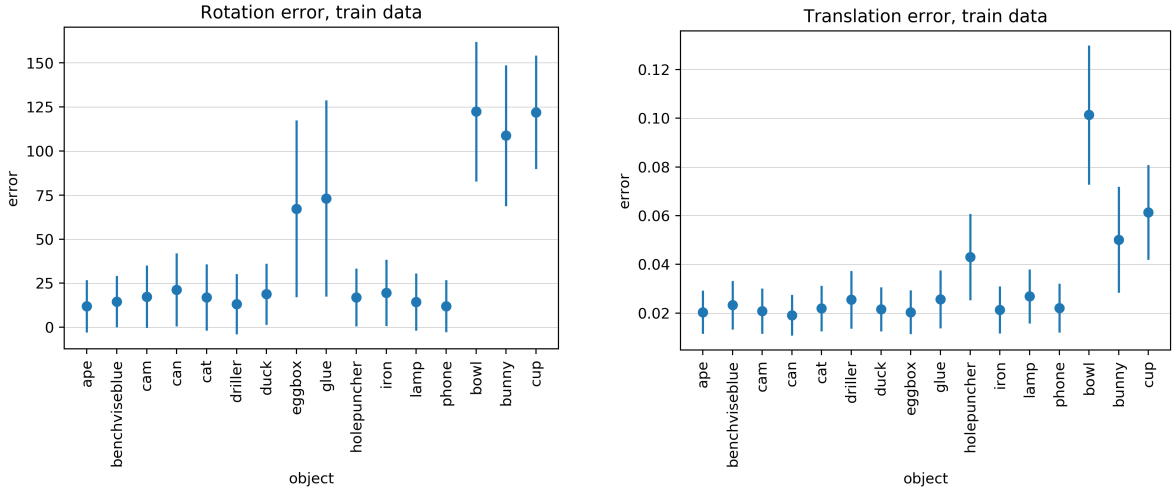
(c) ADD for different objects



(d) ADI for different objects

Figure 5.30: ADD and ADI of the network with inputs created using training parameters

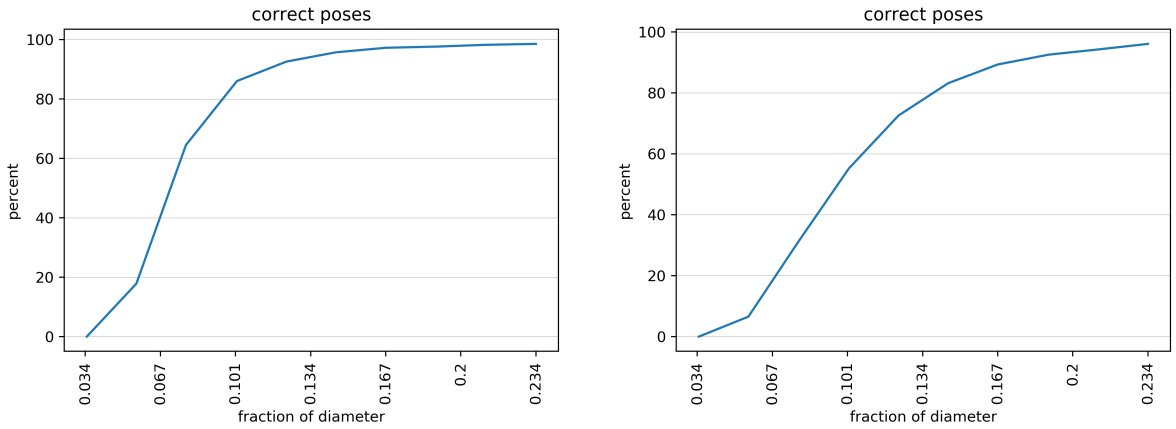
Evaluation of the network with test data sampled with the same parameters as the training data.



(a) Rotation error (b) Translation error

Figure 5.31: Rotation and translation error of inputs generated with parameters used to train the network

Here one can see that the network has problems generalizing to the unseen objects bowl, bunny and cup. The rotation errors are higher for the objects egg box and glue due to their ambiguity.



(a) Correct poses with different fractions of the diameter for ape (b) Correct poses with different fractions of the diameter for the duck

Figure 5.32: Correct pose for different fractions of the object diameter

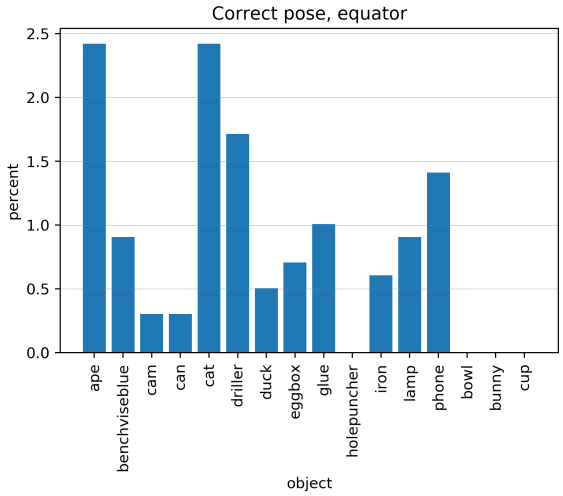
Object	ape	benchviseblue	cam	can	cat	driller	duck	eggbox
Correct	87.9	84.4	63.6	59.2	77.9	87.0	57.3	12.9
ADD	0.1	0.1	0.16	0.16	0.13	0.1	0.18	0.57
Rotation error	11.97	14.57	17.31	21.23	16.96	13.17	18.75	67.27
Translation error	0.02	0.023	0.021	0.019	0.022	0.025	0.022	0.02

Object	glue	hole punch	iron	lamp	phone	bowl	bunny	cup
Correct	30.3	65.5	73.2	83.9	90.7	0.0	0.4	0.1
ADD	0.24	0.15	0.13	0.11	0.08	1.01	0.64	0.84
Rotation error	73.14	16.86	19.54	14.37	12.0	122.39	108.77	122.02
Translation error	0.026	0.043	0.021	0.027	0.022	0.101	0.05	0.061

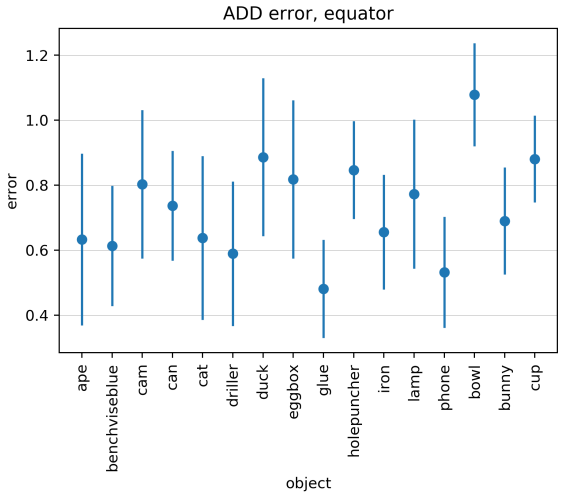
Table 1: Evaluation of the network with data sampled with training parameters

Evaluation around the Equator

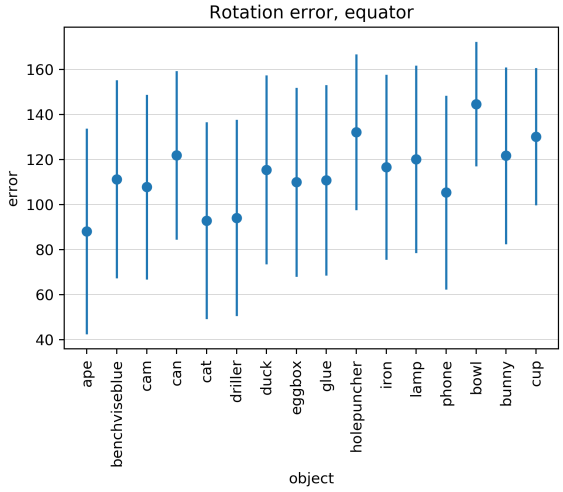
The evaluation for rotations with quaternions around the equator ($\mathbf{q}$ with $q_1 \approx 0$) shows that the error here is bigger and only around 2% of the poses are estimated correctly (see figure 5.33). This is due to the representation of the rotations with quaternions (see section 4.1.4). In section 5.3.2, the network is trained with $2D$ data to see if there are similar issues in $2D$.



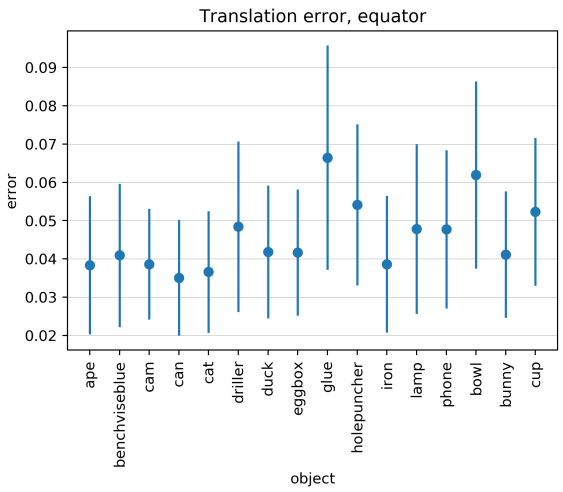
(a) Correct poses according to ADD



(b) ADD for different objects

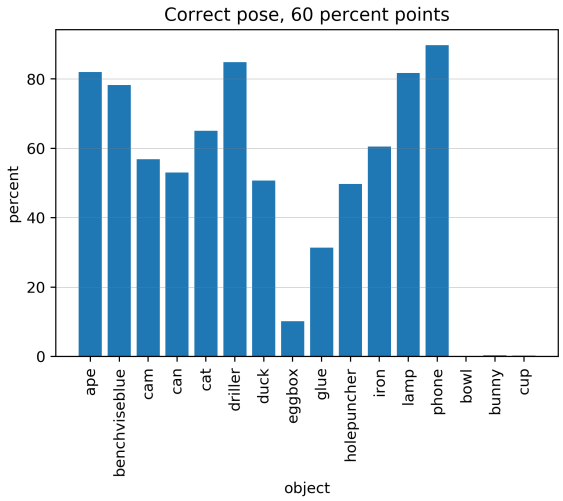


(c) Rotation error

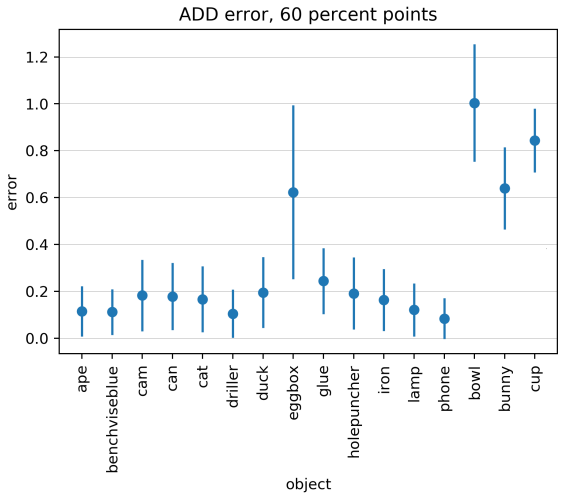


(d) Translation error

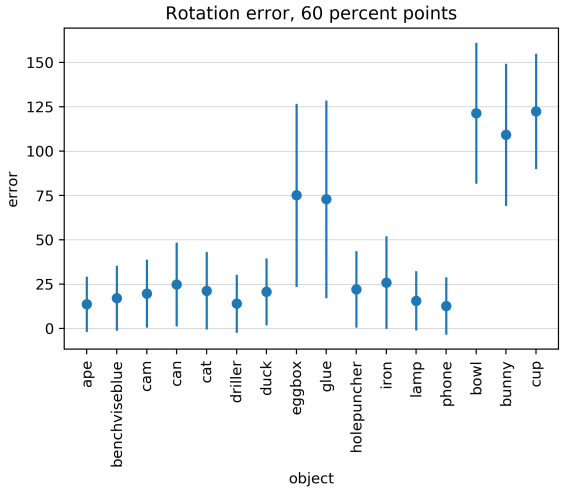
Figure 5.33: Evaluation with rotations around the equator



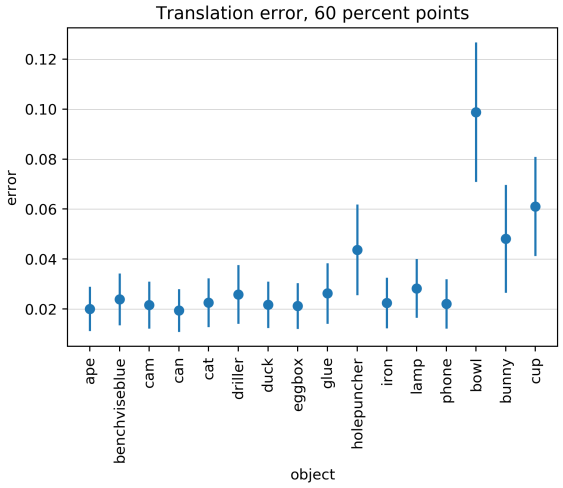
(a) Correct poses according to ADD



(b) ADD for different objects



(c) Rotation error

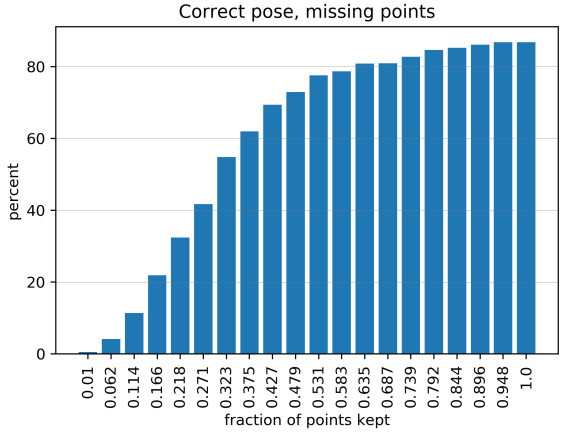


(d) Translation error

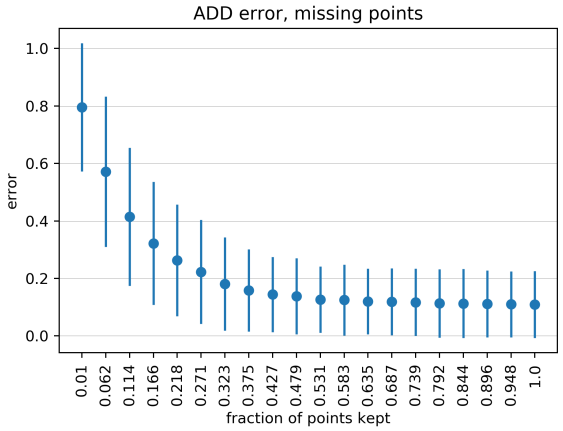
Figure 5.34: Evaluation with 40% points in the target point cloud missing

Evaluation with Random Points Removed

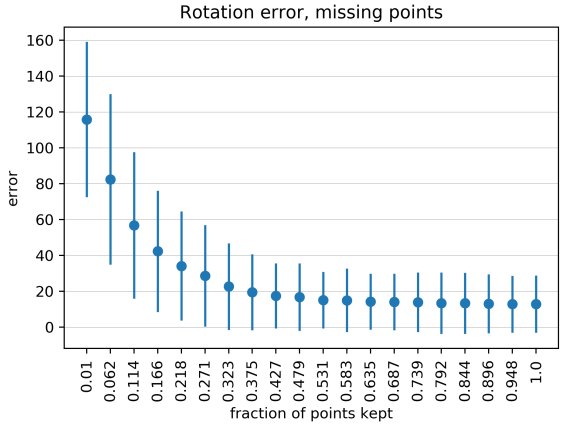
For this test, 40% of the points were removed from the target point cloud $\bar{\mathcal{M}}$. In figure 5.34, one can see that the network is stable against missing points and the error increases only slightly compared to the error with the full target point cloud in figure 5.30. This behaviour was expected, since the PointNet architecture already showed stability against missing points in [QSMG16]. In figure 5.35 is a plot with an increasing number of missing points for the ape point cloud. There one can see that the performance of the network decreases only when more than 50% of the points are removed.



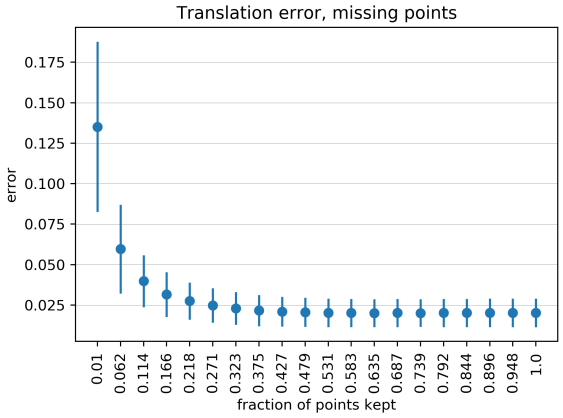
(a) Correct poses according to ADD



(b) ADD for decreasing number of points



(c) Rotation error for decreasing number of points



(d) Translation error for decreasing number of points

Figure 5.35: Evaluation with decreasing points in the target point cloud

The figure shows how removing an increasing number of points from the ape point cloud affects the performance of the network. The translation error increases significantly when more than 80% of the points are removed, while the rotation error already increases when more than 50% of the points are removed.

Evaluation with Partial Visibility

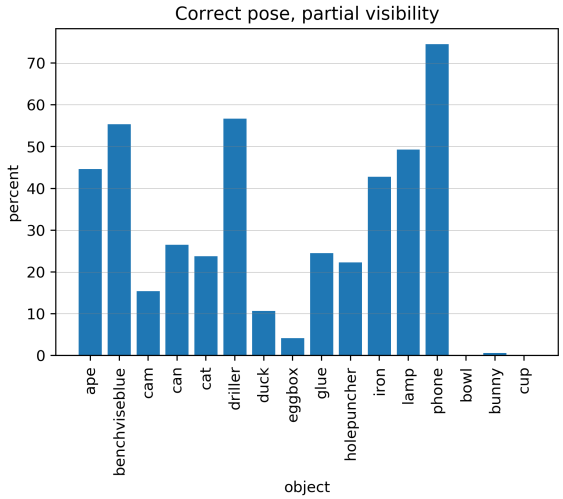
For this test, all points $\mathbf{x} \in \bar{\mathcal{M}}$ with $x_1 < -0.5$ were removed to simulate partial visibility. Here the performance of the network decreases significantly compared to the performance in the evaluation with the training parameters (figure 5.30).
This network was not trained on partially visible data, but its inability to predict poses for only partially visible objects might be a problem on the real datasets. There the back side of the objects are hidden and large parts of the objects might be hidden behind other objects. To address this weakness of the network, a loss that is more stable against outliers can be used (see section 2.6.7). Additionally, the network can be trained with partial visible data to improve the performance here.
The objects bunny, cup and bowl that were not in the training set were in all the previous tests incorrectly estimated and are from now on omitted from the tests.

Evaluation with More Noise

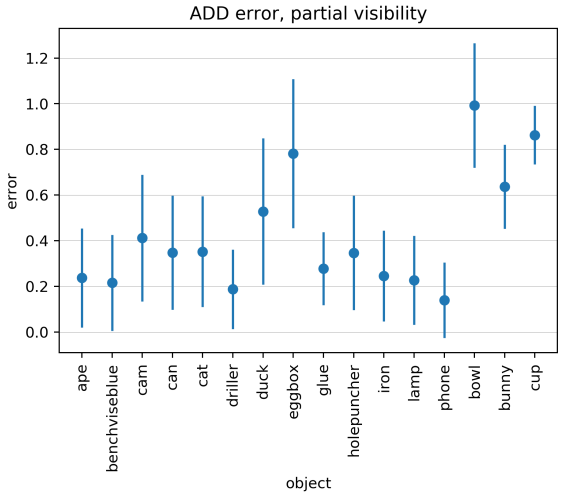
For the training of the network, the maximal movement of a point due to noise was set to 0.05. To see how stable the network is against noise, the maximal movement for this test was set to 0.1. In figure 5.38, one can see that this doubling of the noise decreases the performance of the network significantly. To show how much different levels of noise affect the network in estimating the performance of the ape point cloud see figure 5.39.

Evaluation with Clutter

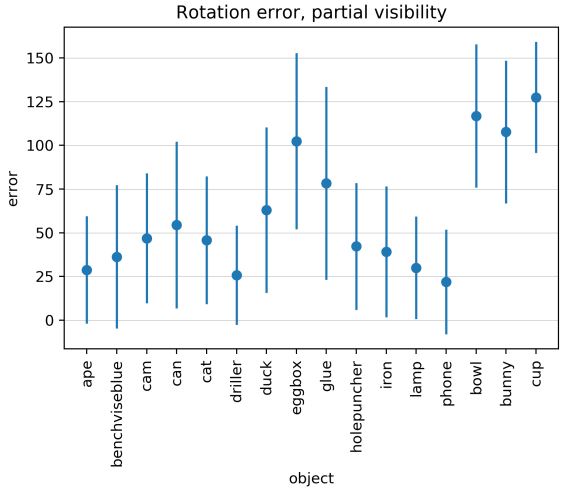
For this evaluation, 5% of the points were replaced by random points on a plane to simulate clutter around the objects. The network is not robust against clutter and only 2% to 12% of the poses are estimated correctly as seen in figure 5.40. Figure 5.41 shows that even a small amount of clutter affects the performance a lot. This network was not trained with clutter. To improve the performance with clutter, the network can be trained with a robust loss function, like Tukey or Huber (see section 2.6.7), and clutter in the trainings data.



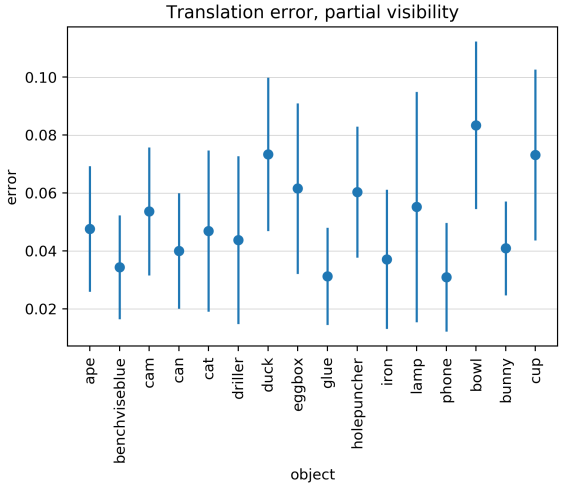
(a) Correct poses according to ADD



(b) ADD for different objects



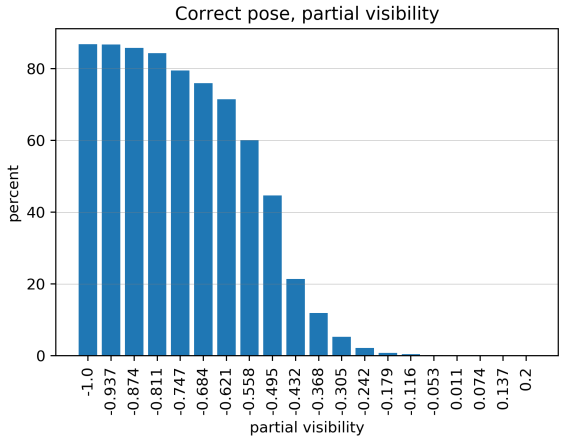
(c) Rotation error



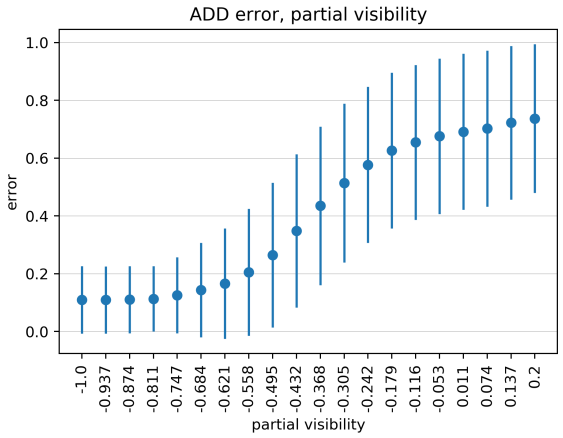
(d) Translation error

Figure 5.36: Evaluation with partially visible target point cloud

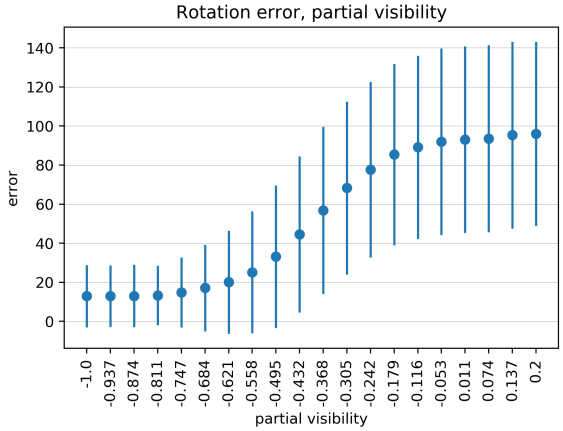
For this evaluation the source point cloud was first rotated and then all points $\mathbf{x} = (x_1, x_2, x_3)$ with $x_1 < c_{pvis}$ were removed. c_{pvis} is the partial visibility value. After that, the point clouds were translated. The error increases especially on compact objects like the ape and cam. Objects that have something standing out, like the phone, are less affected by it. This is because the objects were centred to zero mean and scaled to be in the unit ball. For compact objects, there are more points close to the unit ball and thus more points get removed.



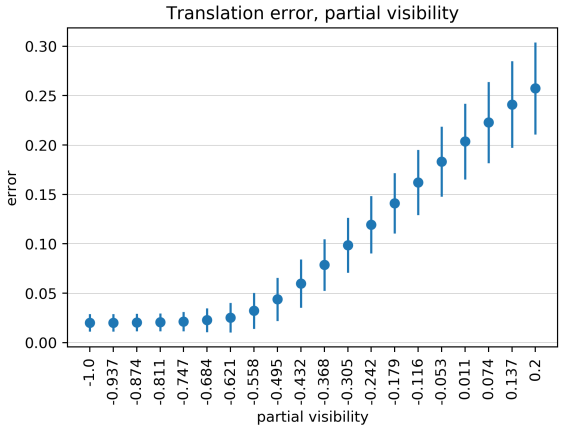
(a) Correct poses according to ADD



(b) ADD for decreasing visibility



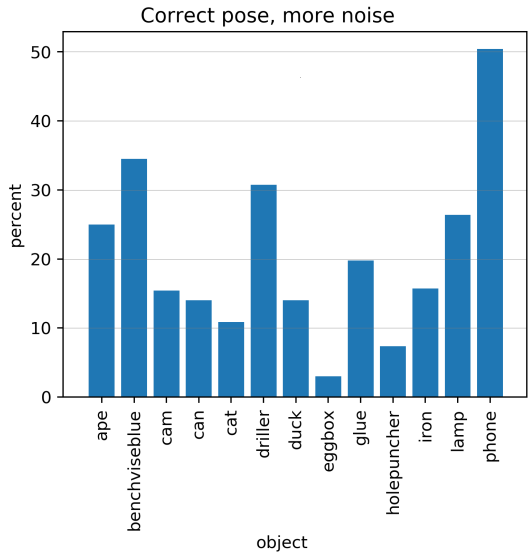
(c) Rotation error for decreasing visibility



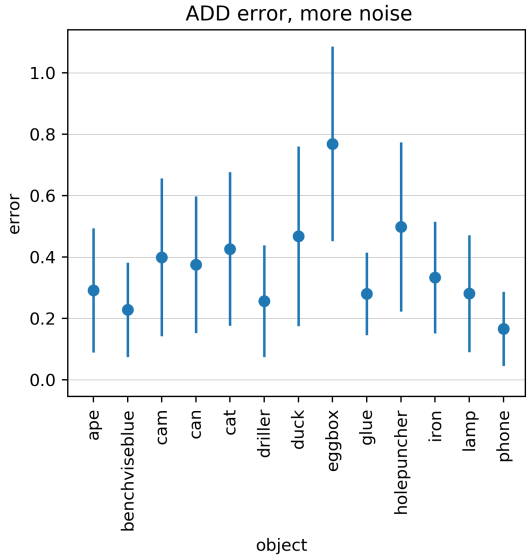
(d) Translation error for decreasing visibility

Figure 5.37: Evaluation with decreasing visibility in the target point cloud

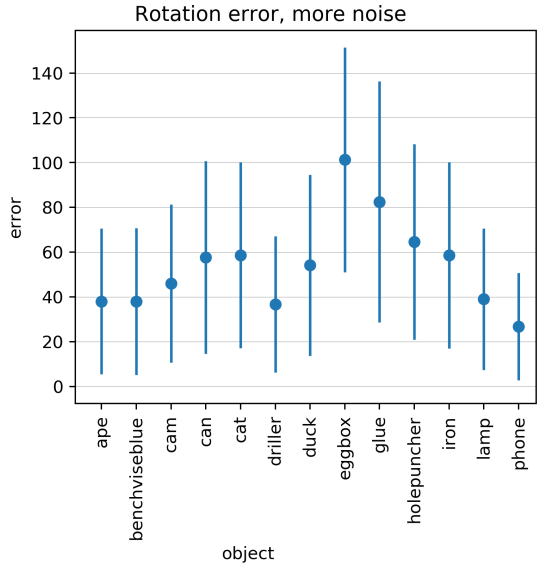
For this evaluation the source point cloud was first rotated and then all points $\mathbf{x} = (x_1, x_2, x_3)$ with $x_1 < c_{pvis}$ were removed. c_{pvis} is the partial visibility value. After that, the point clouds were translated. Here only the ape point cloud is evaluated. The translation error increases at the partial visibility value -0.5, the rotation error already increases at -0.7.



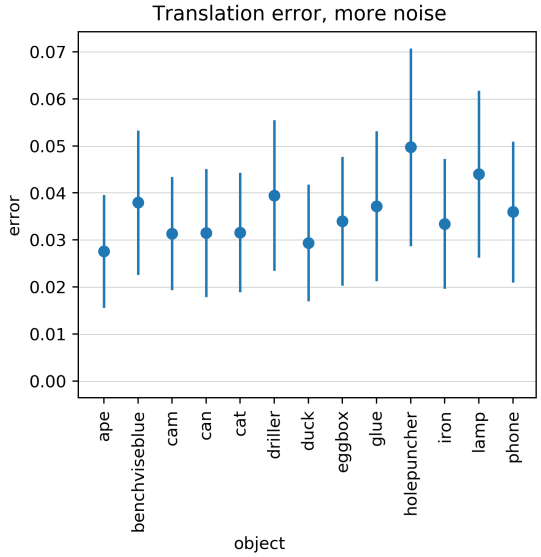
(a) Correct poses according to ADD



(b) ADD for different objects



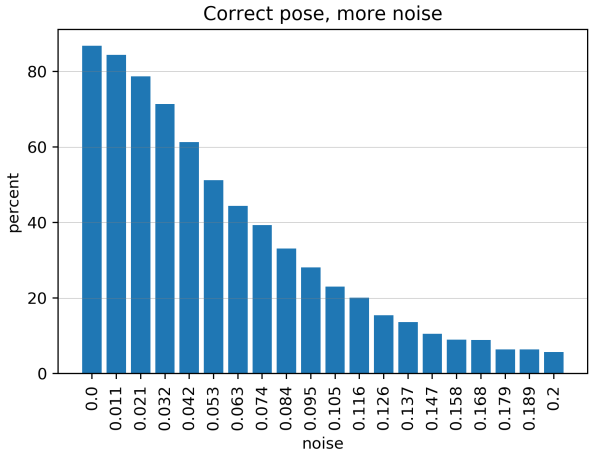
(c) Rotation error



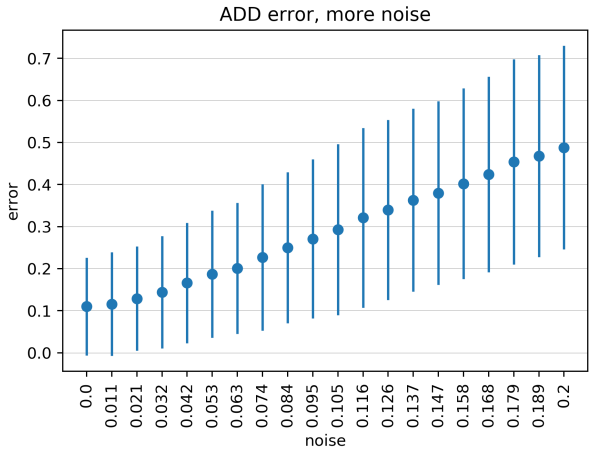
(d) Translation error

Figure 5.38: Evaluation with more noise in the target point cloud

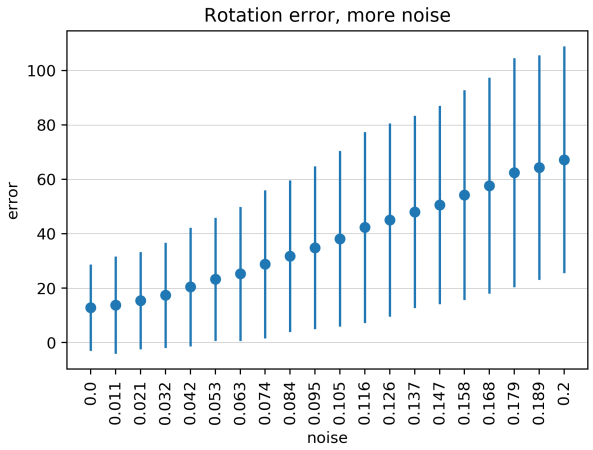
In this evaluation the maximal movement of the point in the target point cloud was 0.1, compared to a maximal noise of 0.05 during training. Compact objects like the cam and the egg box are more affected by noise, while long objects or objects that have something standing out, like the phone, driller and bench vise are less affected.



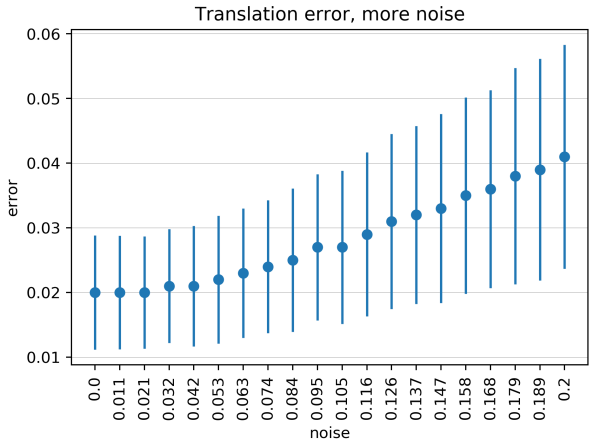
(a) Correct poses for increasing noise



(b) ADD for increasing noise



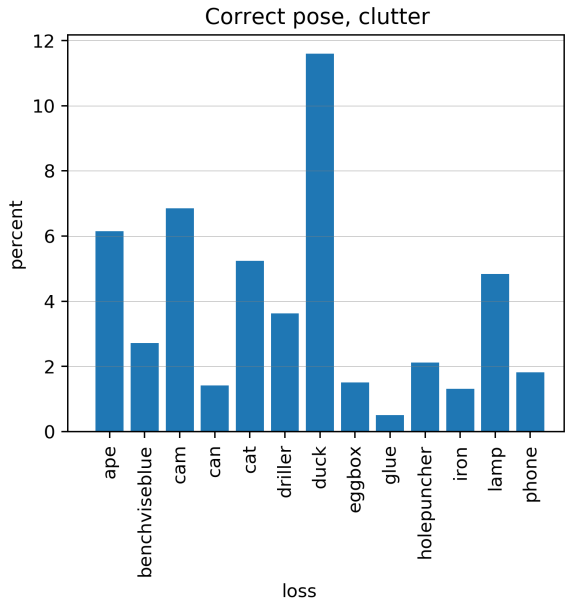
(c) Rotation error for increasing noise



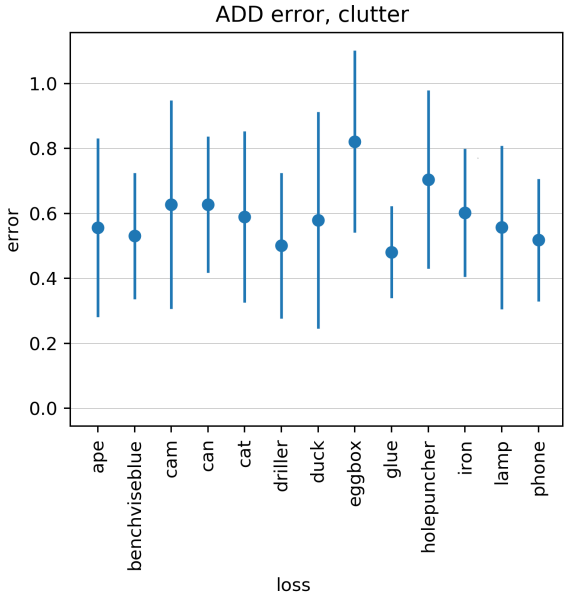
(d) Translation error for increasing noise

Figure 5.39: Evaluation with increasing noise in the target point cloud

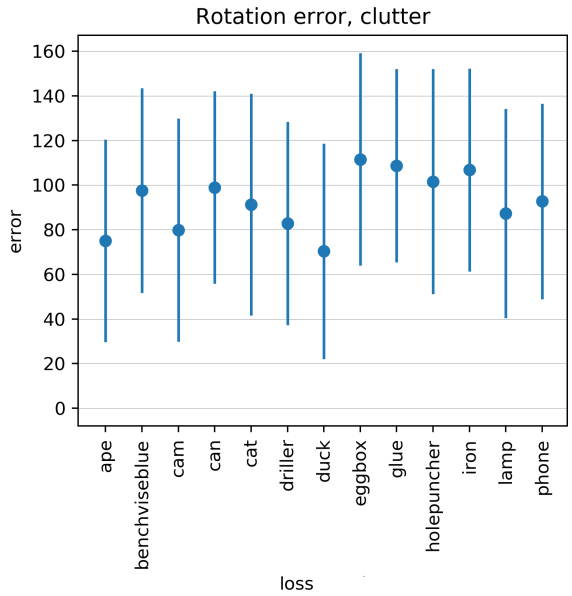
The network is not very stable against an increase in noise. The rotation error gets higher even for small noise of 0.01 (see figure 5.38c), while the translation error increases only after a maximal noise of 0.03 (see figure 5.38d).



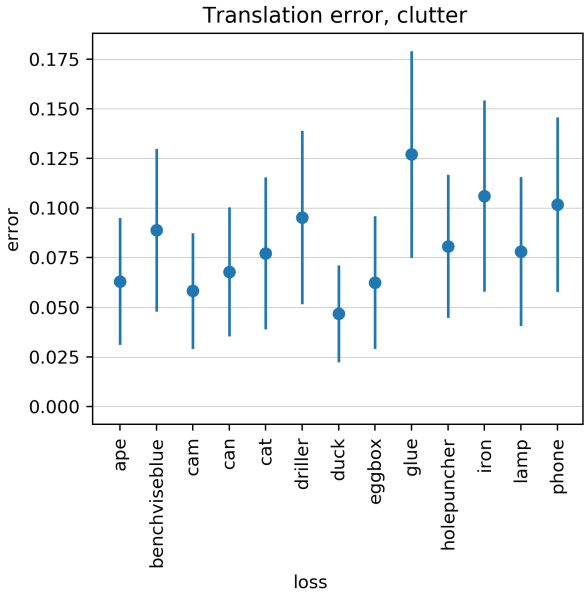
(a) Correct poses according to ADD



(b) ADD for different objects

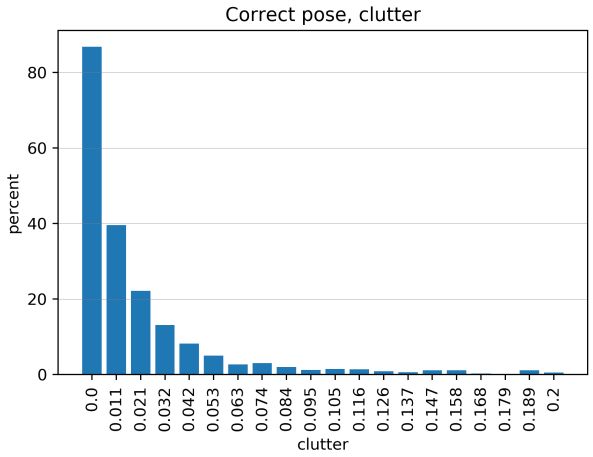


(c) Rotation error

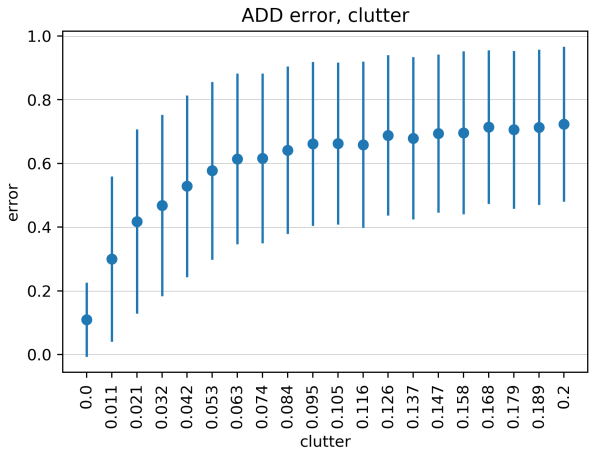


(d) Translation error

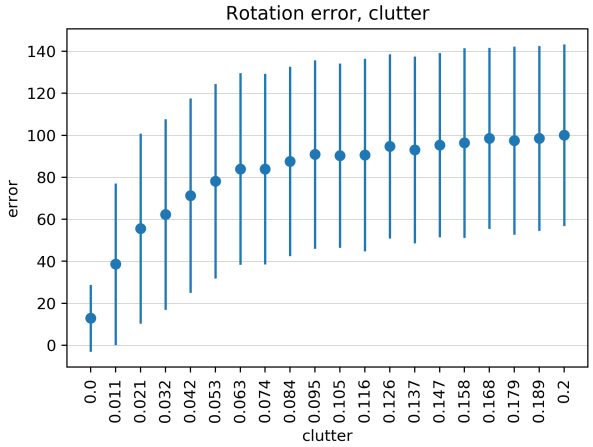
Figure 5.40: Evaluation with clutter in the target point cloud



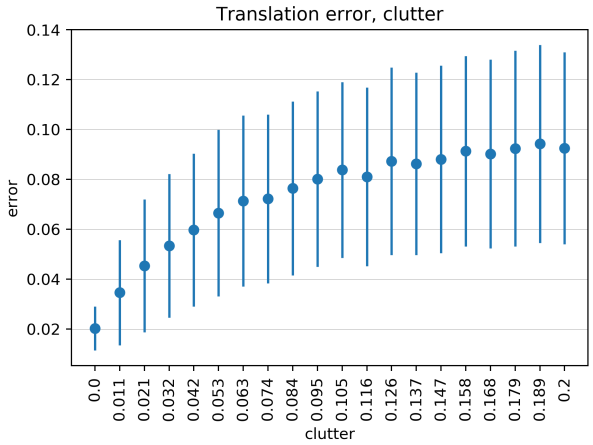
(a) Correct poses according to ADD



(b) ADD for increasing clutter



(c) Rotation error for increasing clutter



(d) Translation error for increasing clutter

Figure 5.41: Evaluation with increasing clutter in the target point cloud

The figure shows how adding an increasing number clutter to the ape point cloud affects the performance of the network. When 1.1% of the points in the point cloud are replaced by random points on a plane the percentage of correctly estimated poses drops from 87.9% to 39.2%.

5.3.2 $2D$ Data

In this section the networks were trained on $2D$ data, to see how the network performs for the easier case of $2D$ rotations and translations.
The network was trained on the fish point cloud in figure 3.16 with uniformly distributed rotations and a maximal translation $\|\mathbf{t}\| \approx 0.3$. The loss with the quaternions was 2.30 and the loss with the $2D$ rotation matrices was

$$\mathcal{L}_{\mathbf{R}}(\bar{\mathbf{P}}, \hat{\mathbf{P}}, \mathcal{M}) = w_{rot} \cdot \varphi_5(\bar{\mathbf{R}}, \hat{\mathbf{R}}) + w_{transl} \cdot L_{transl}(\bar{\mathbf{t}}, \hat{\mathbf{t}}), \tag{5.37}$$

with $\mathbf{R} \in SO(2)$, φ_5 which is adjusted to $2D$, and weights $w_{rot} = w_{transl} = 1$. The weights of the network were updated with the Adam optimizer (section 2.6.5) with learning rate 0.0005. Both networks train very well, as seen in figures 5.43 and 5.44. Figure 5.45 shows the evaluation of the two networks on data sampled with the training parameters and on data where the rotations are around the equator with $q_1 = 1$. Since rotations in $2D$ are much simpler than rotations in $3D$, here the predictions are very good. Rotations in $2D$ can be represented in $U(1)$ without the need to identify antipodal points, while for $3D$ rotations antipodal points are identified on S^3. Future work can be done in finding a representation for $3D$ rotations, which solves the issues of the representation with quaternions.
Figure 5.42 shows a predicted point cloud $\hat{\mathcal{M}}$ compared with its ground truth point cloud $\hat{\mathcal{M}}$.

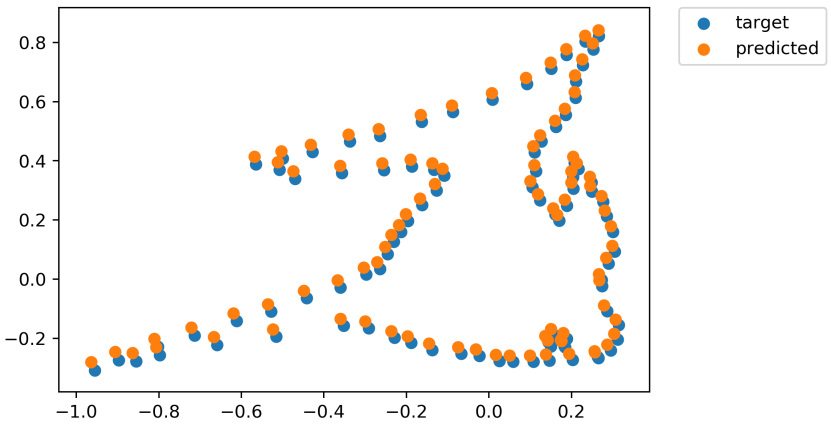


Figure 5.42: Target and predicted fish point cloud

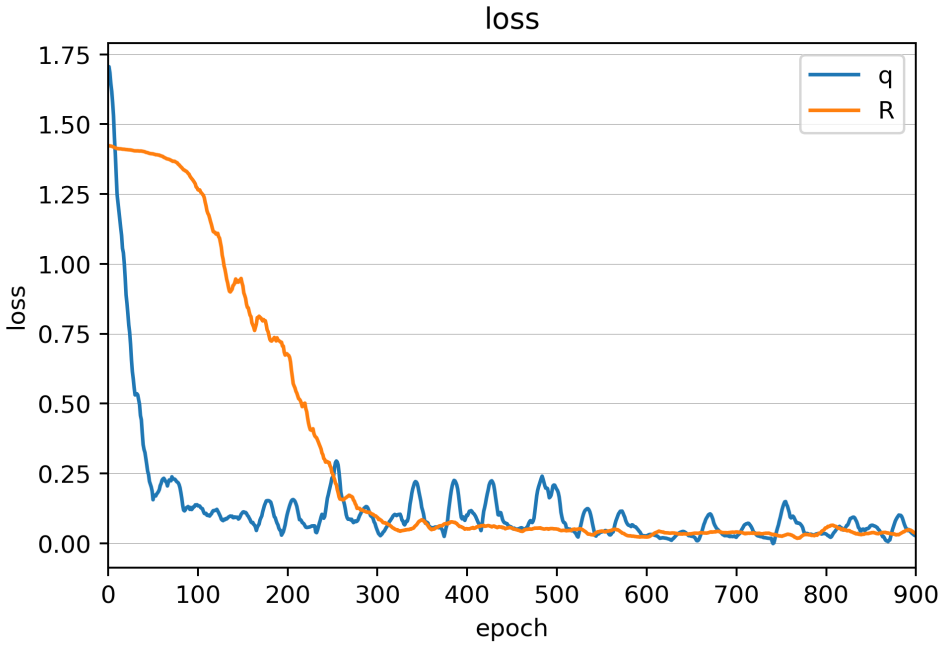
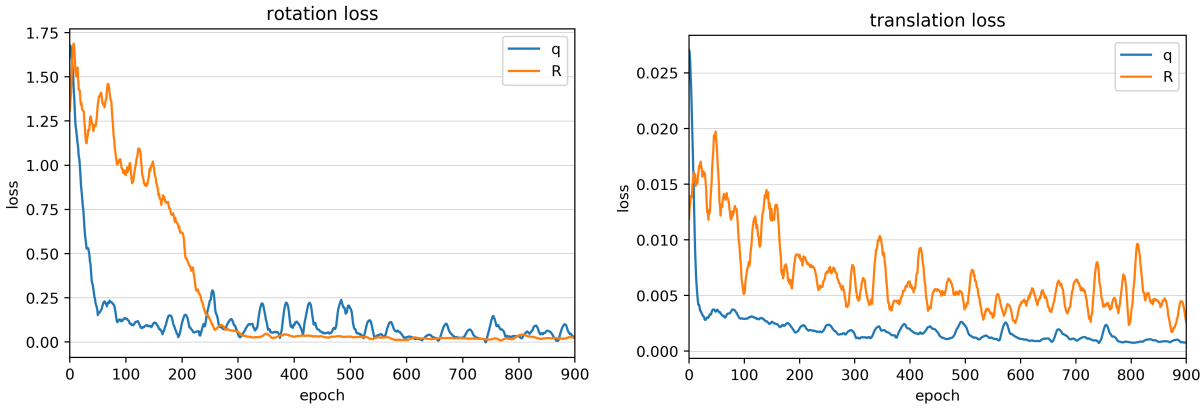


Figure 5.43: Loss during training with $2D$ data

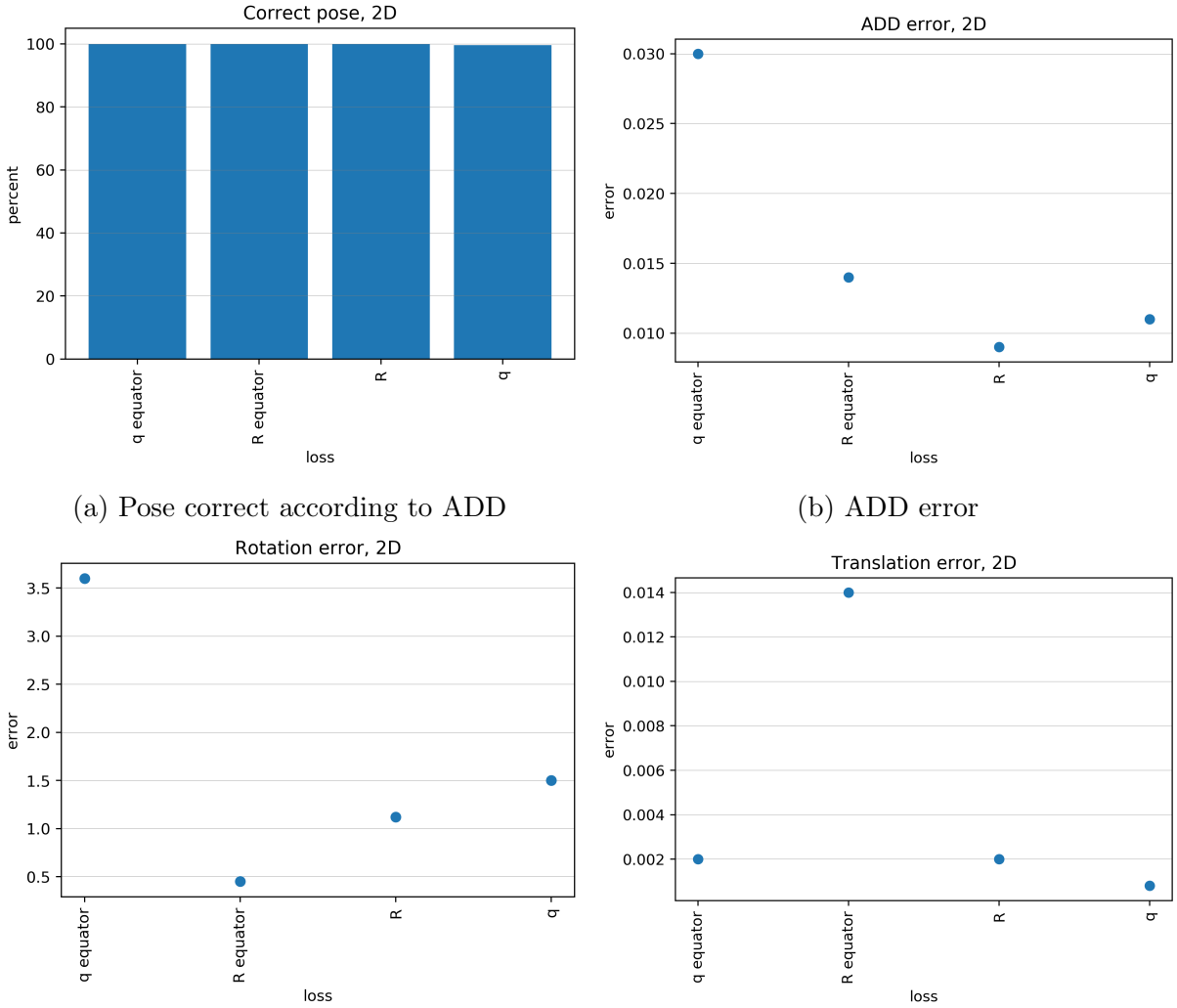
The loss function for the orange graph was 5.37 and the loss function for the blue graph was 2.30. The weights w_{rot}, w_{transl} were for both losses equal to one.



(a) Rotation loss with $2D$ data

(b) Translation loss with $2D$ data

Figure 5.44: Losses during training on $2D$ data



(a) Pose correct according to ADD

(b) ADD error

(c) Rotation error

(d) Translation error

Figure 5.45: Evaluation on $2D$ data

Here one can see that in the $2D$ case there is no higher error around the equator and that both the networks trained with rotation matrices and quaternions have a similar performance.

5.4 Training on the Linemod Dataset

Two different networks were trained on the Linemod dataset. The first network, called *translation-network*, was trained with the translation loss 2.15 to predict only the translation. The second network, called *rotation-network* was trained on centred data[17], to predict rotations and translations. The idea behind using two separate networks to predict the pose was that the translation-network predicts the translation, then this information is used to center the point cloud and the rotation-network can predict the rotation and refine the prediction for the translation on the centred data. This was necessary because the object diameters were relatively small compared to the translation (see section 4.1.5). With non-centred data, the networks that should predict both translation and rotation, converged to one rotation because it could not distinguish between separate points in the target point cloud.

Two different loss functions were used to train the rotation-network. Because of partial

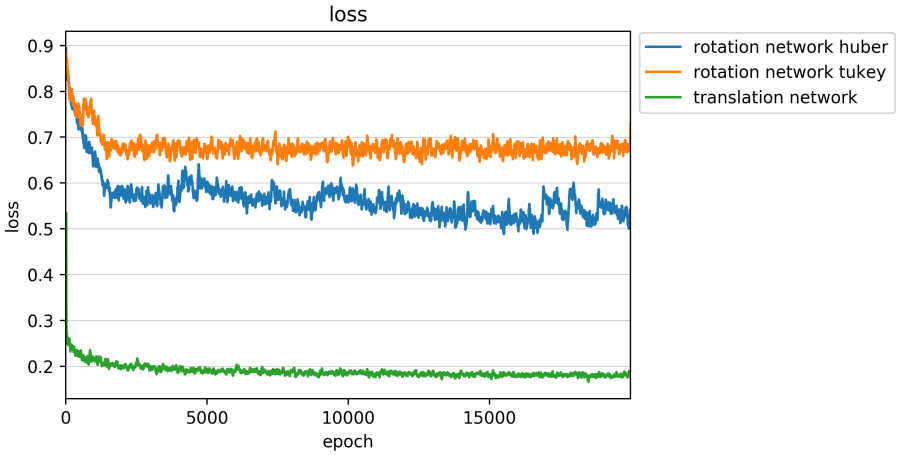


Figure 5.46: Loss during training with Linemod dataset

The figure shows in blue the loss during training with $\mathcal{L}_1$, in orange the loss with $\mathcal{L}_2$ and in green the loss of the translation-network. The translation-network converges to the ground truth, while the rotation networks stay at a high loss.

visibility and clutter, the Huber loss 2.27 and the Tukey loss 2.29 were used in $\mathcal{L}_1$ and $\mathcal{L}_2$:

$$\mathcal{L}_1(\bar{\mathbf{P}}, \hat{\mathbf{P}}, \mathcal{M}) = w_{res} \cdot L_{hub}(\bar{\mathcal{M}}, \hat{\mathcal{M}}) + w_{rot} \cdot \varphi_1(\bar{\mathbf{q}}, \hat{\mathbf{q}}) + w_{transl} \cdot L_{transl}(\bar{\mathbf{t}}, \hat{\mathbf{t}}) + w_{reg} \cdot L_{reg}(\hat{\mathbf{q}}), \tag{5.38}$$

$$\mathcal{L}_2(\bar{\mathbf{P}}, \hat{\mathbf{P}}, \mathcal{M}) = w_{res} \cdot L_{tukey}(\bar{\mathcal{M}}, \hat{\mathcal{M}}) + w_{rot} \cdot \varphi_1(\bar{\mathbf{q}}, \hat{\mathbf{q}}) + w_{transl} \cdot L_{transl}(\bar{\mathbf{t}}, \hat{\mathbf{t}}) + w_{reg} \cdot L_{reg}(\hat{\mathbf{q}}). \tag{5.39}$$

The weights were $w_{res} = w_{rot} = w_{transl} = 1$, $w_{reg} = 0.01$ and the translation weight was gradually reduced during training to $w_{transl} = 0.25$. The parameters for the losses were $\delta = 1$ for the Huber loss and $c = 0.7$ for the Tukey loss. The learning rate was set to 0.0005 and the Adam optimizer was used to update the weights of the network.

[17]The target point cloud was centred to zero mean and then a small translation was applied to account for inaccuracies in the translation-network.

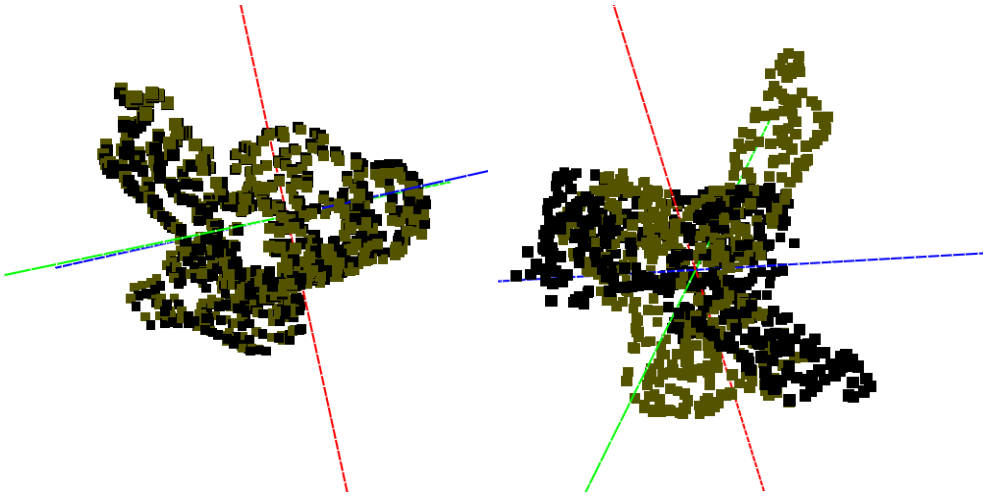
Figure 5.46 shows the losses for the different networks. The translation network converged quite well to the ground truth translation, while the rotation-networks did not train well. This is due to the partial visibility and clutter in the target point cloud, where the network had already problems on the synthetic dataset. An approach to handle clutter can be to add colour information to the point cloud that could be used to single out the points that belong to the objects. Another possible solution could be to train the networks first on data without partial visibility and clutter and then refine the weights of the network on the real data. This method is called *transfer learning* and is successfully used in image classification[OBLS14].

6 Conclusion

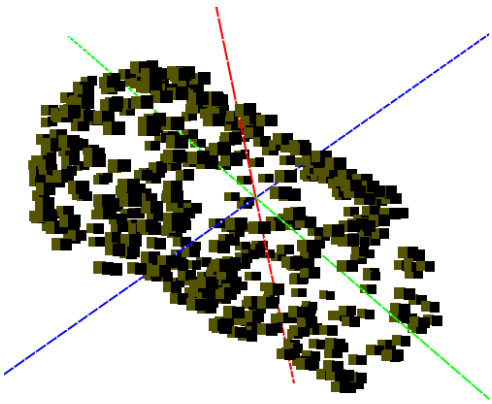
The networks performed quite well on synthetic data showing that the concept of using the modified PointNet architecture to estimate poses on point clouds is effective. However, with real datasets that include clutter and occlusions, the methods did not converge. Future work can be done in testing to see if additional information like surface normals or colour improve the performance of the method. An approach to improving the convergence of the network can be transfer learning, where the network is first trained on data without clutter and occlusions and then the network weights can be refined with training on data with occlusions and clutter.
The representation of $3D$ rotations with quaternions and axis-angle create errors for rotations of $180°$ and the representation with rotation matrices $\mathbf{R} \in SO(3)$ did not converge in this thesis. Future work here can be done in finding a representation of rotations, which lets the network converge and does not have a higher error for rotations of a certain angle.

A Examples for Predicted Poses

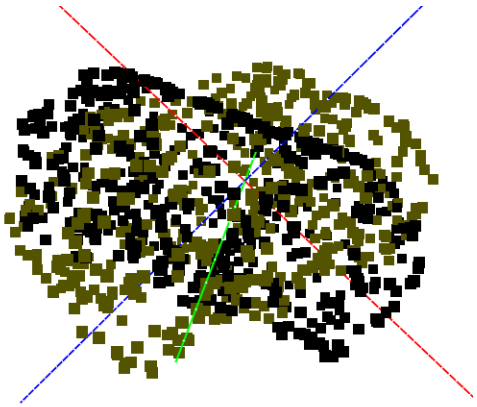
The poses here were predicted using the network in section 5.3.1. The figures show the estimated pose in yellow and the ground truth pose in black.



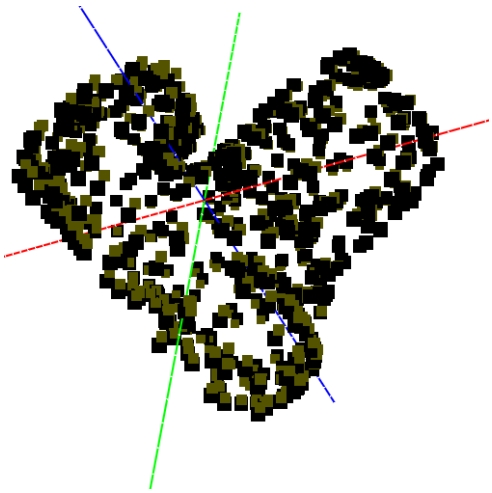
(a) Correct estimated benchvise (b) Incorrect estimated benchvise



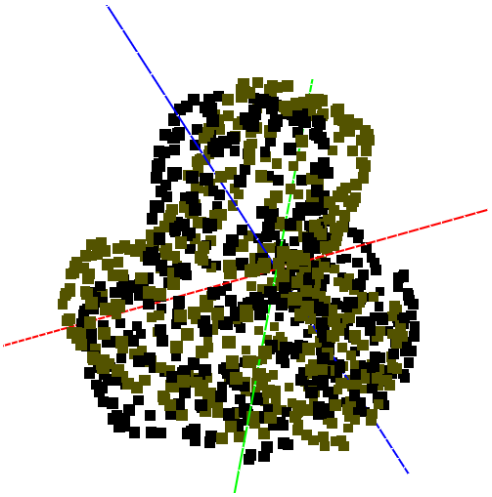
(a) Correct estimated eggbox



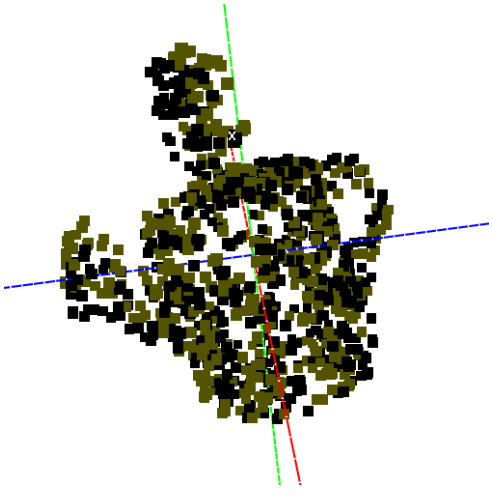
(b) Incorrect estimated eggbox



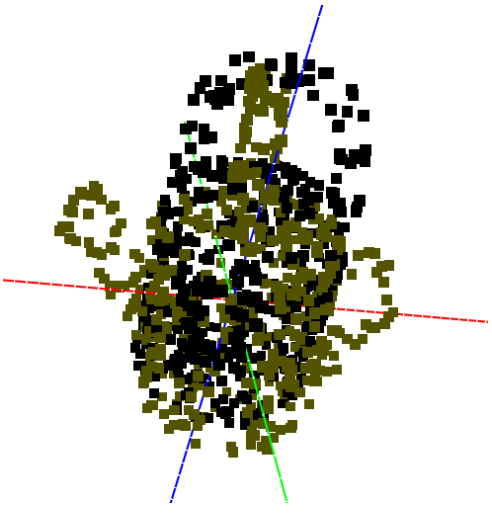
(a) Correct estimated cam



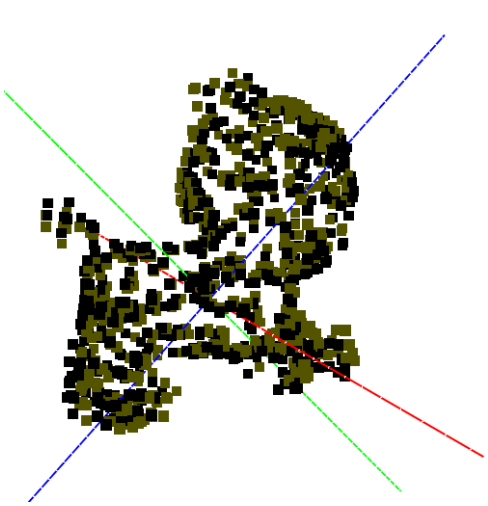
(b) Incorrect estimated cam



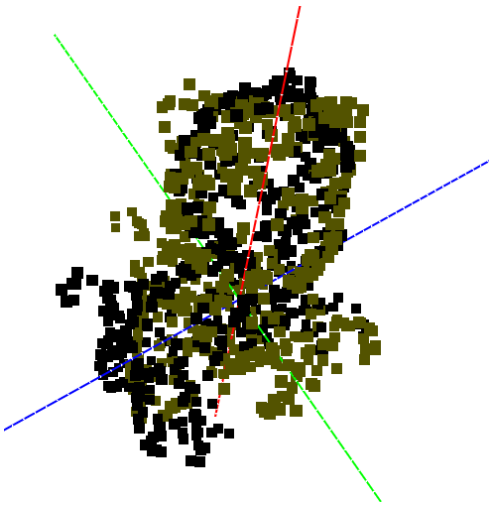
(a) Correct estimated can



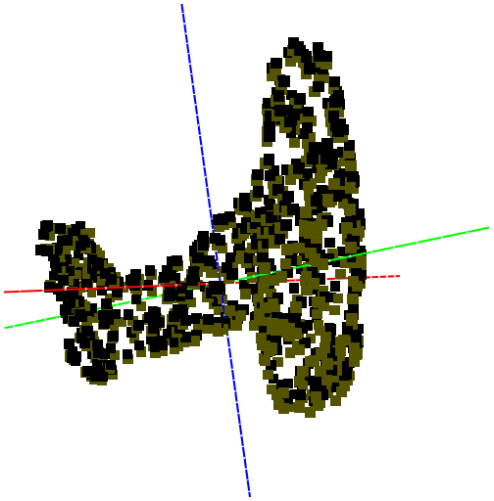
(b) Incorrect estimated can



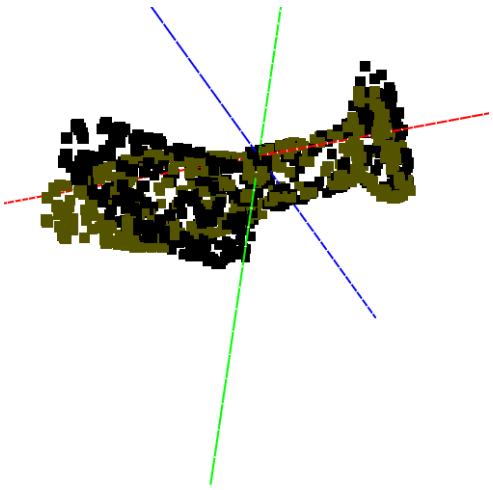
(a) Correct estimated cat



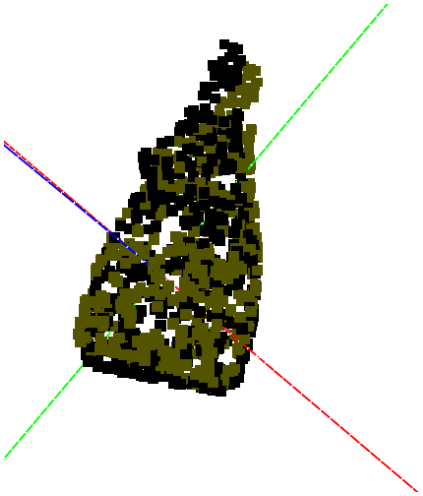
(b) Incorrect estimated cat



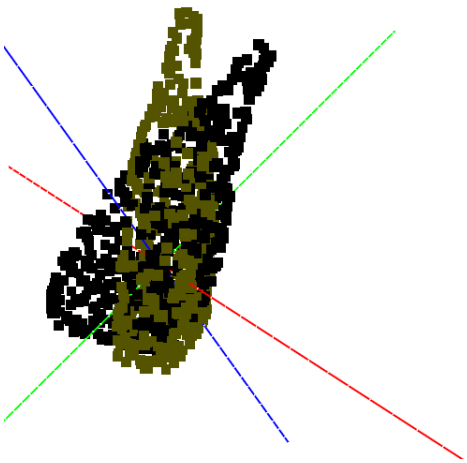
(a) Correct estimated drill



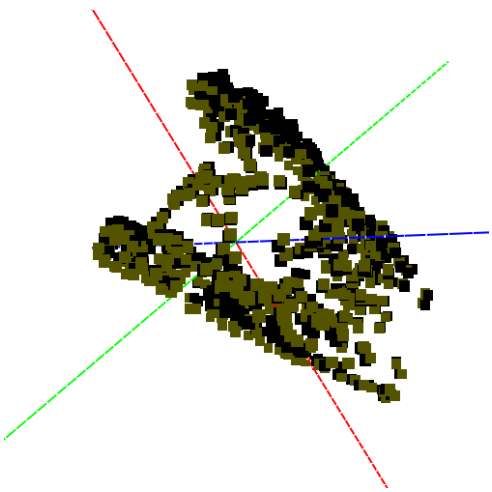
(b) Incorrect estimated drill



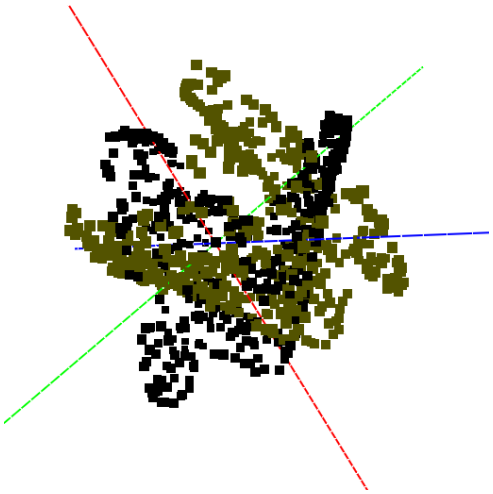
(a) Correct estimated glue



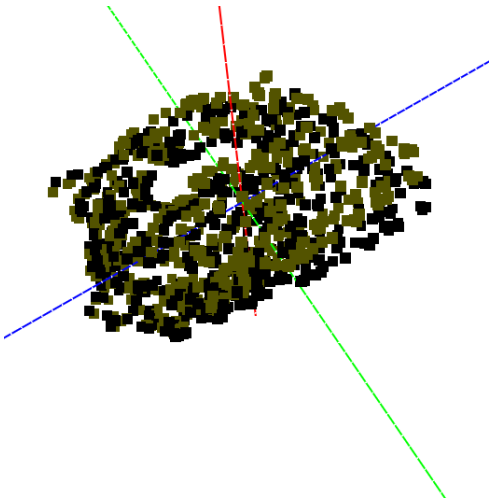
(b) Incorrect estimated glue



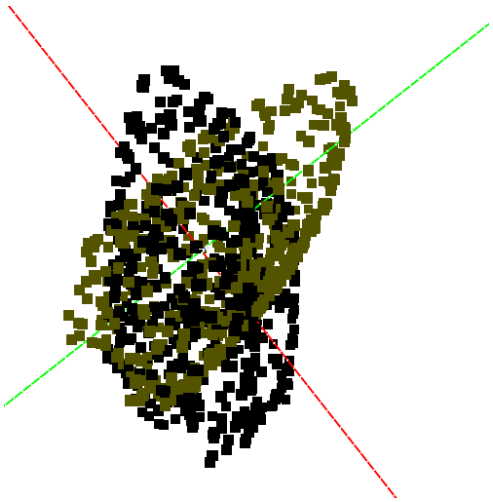
(a) Correct estimated hole punch



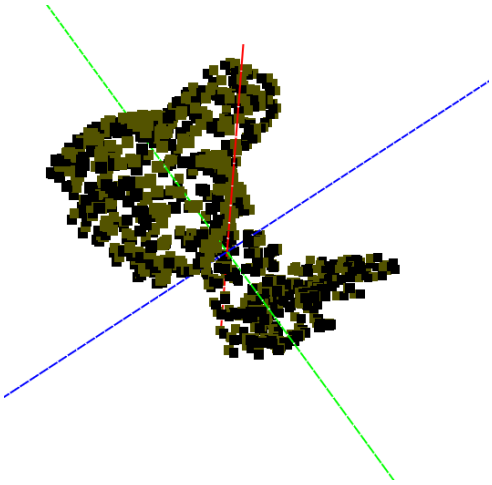
(b) Incorrect estimated hole punch



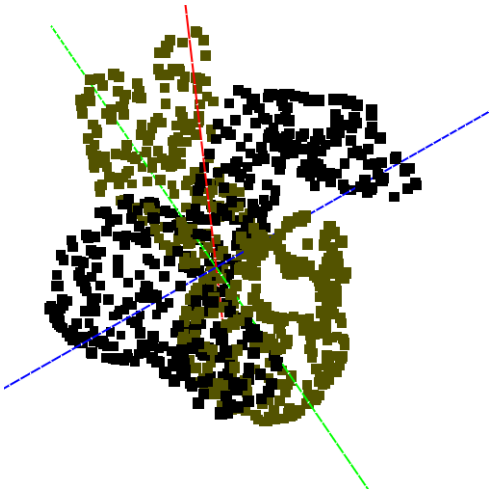
(a) Correct estimated iron



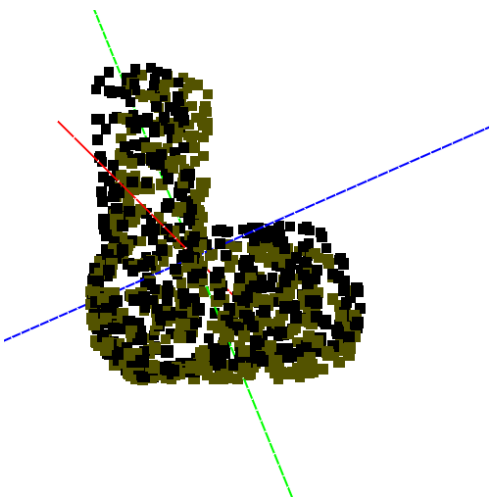
(b) Incorrect estimated iron



(a) Correct estimated lamp

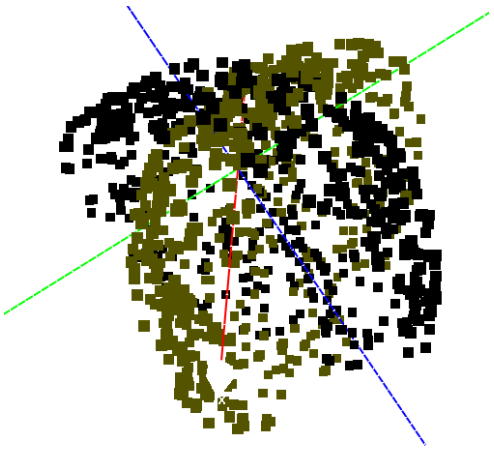


(b) Incorrect estimated lamp

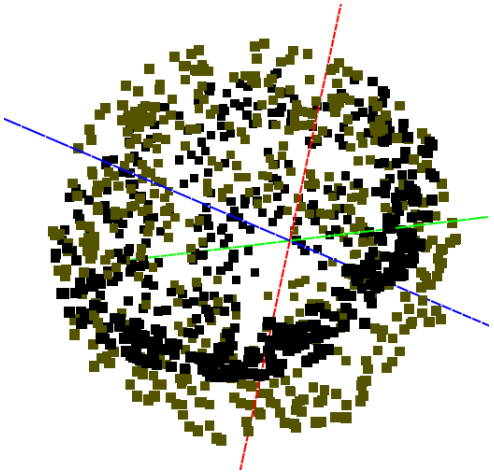


(a) Correct estimated phone

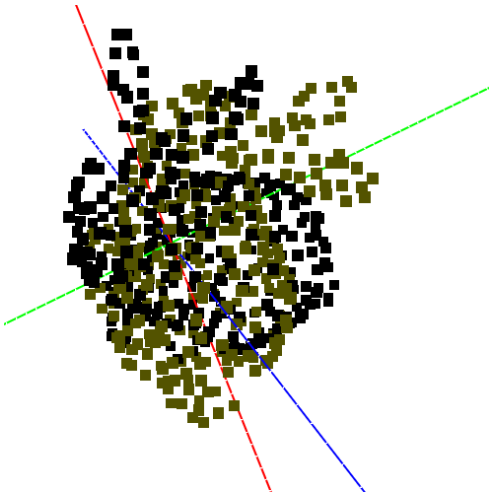
(b) Incorrect estimated phone



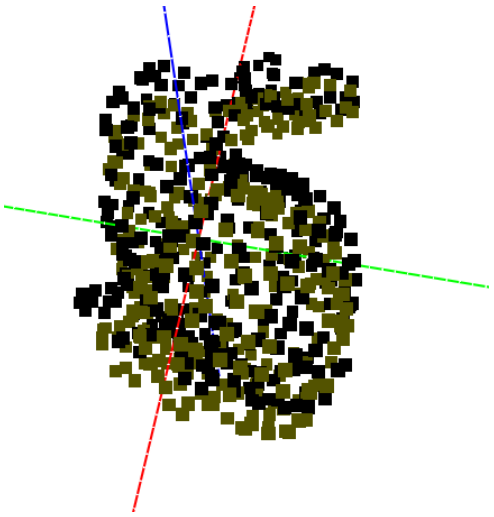
(a) Incorrect estimated bowl



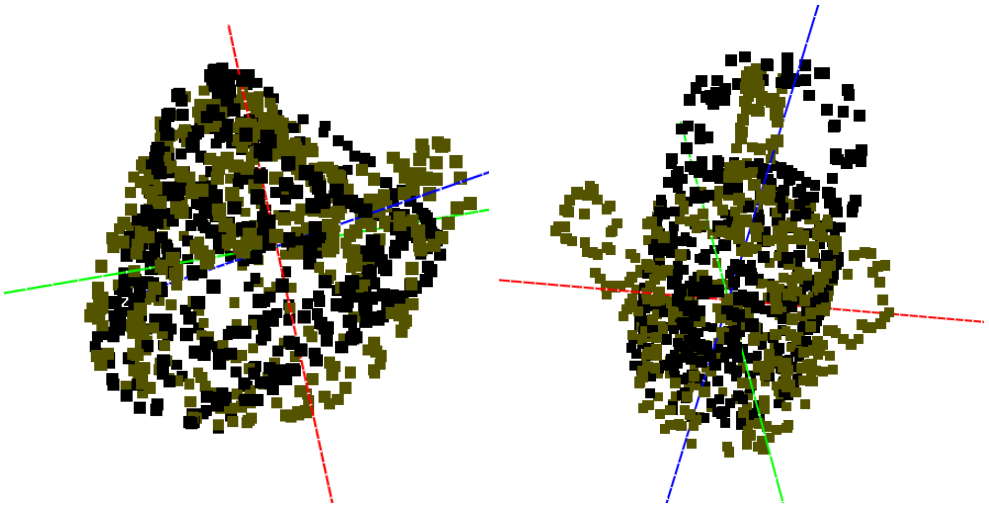
(b) Incorrect estimated bowl



(a) Incorrect estimated bunny



(b) Incorrect estimated bunny



(a) Incorrect estimated cup

(b) Incorrect estimated cup

List of Figures

References

[Ano] Anonymous. Explaining the Ambiguity of Object Detection and Pose Estimation from Visual Data.

[Bak02] Andrew Baker. *Matrix Groups: An Introduction to Lie Group Theory*. Springer, 2002.

[Bal11] Pierre Baldi. Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, pages 37–50. JMLR.org, 2011.

[BBN] Benjamin Busam, Tolga Birdal, and Nassir Navab. Camera Pose Filtering with Local Regression Geodesics on the Riemannian Manifold of Dual Quaternions. *ICCVW 2017*.

[BEC+15] Benjamin Busam, Marco Esposito, Simon Che'Rose, Nassir Navab, and Benjamin Frisch. A stereo vision approach for cooperative robotic movement therapy. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, December 2015.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[BM92] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992.

[Bot10] León Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. *Proceedings of COMPSTAT'2010*, pages 177–186, 2010.

[CHC99] Chu-Song Chen, Yi-Ping Hung, and Jen-Bo Cheng. Ransac-based darces: A new approach to fast automatic registration of partially overlapping range images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(11):1229–1234, November 1999.

[CHL05] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 539–546, Washington, DC, USA, 2005. IEEE Computer Society.

[DBI18] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPFNet: Global Context Aware Local Features for Robust 3D Point Matching. 2018.

[DBKK16] Andreas Doumanoglou, Vassileios Balntas, Rigas Kouskouridas, and Tae-Kyun Kim. Siamese regression networks with efficient mid-level feature extraction for 3d object pose estimation. *CoRR*, abs/1607.02257, 2016.

[Die06] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[DUNI10] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model Globally, Match Locally: Efficient and Robust 3D Object Recognition. *CVPR*, 2010.

[ETW08] David Eberly, Geometric Tools, and Redmond Wa. Rotation Representations and Performance Issues. pages 1–13, 2008.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[GAA+12] Frederic Garcia, Djamila Aouada, Hashim Kemal Abdella, Thomas Solignac, Bruno Mirbach, and Björn Ottersten. Depth enhancement by fusion for passive and active sensing. In Andrea Fusiello, Vittorio Murino, and Rita Cucchiara, editors, *Computer Vision – ECCV 2012. Workshops and Demonstrations*, pages 506–515, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[GDG+17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. 2017.

[GKW+18] Georgios Georgakis, Srikrishna Karanam, Ziyan Wu, Jan Ernst, and Jana Kosecka. End-to-end learning of keypoint detector and descriptor for pose invariant 3D matching. 2018.

[GRL+98] Steven Gold, R Anand Rangarajan, Chien-Ping Lu, Suguna Pappus, and Eric Mjolsness. New Algorithms for 2D and 3D Point Matching: Pose Estimation and Correspondence. *Pattern Recognition*, 31(8):1019–1031, 1998.

[GSK+17] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[HCI+18] Rein Houthooft, Richard Y. Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved Policy Gradients. (21), 2018.

[HLI+13] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. *Lecture Notes in Computer Science*, 7724 LNCS(Part 1):548–562, 2013.

[HMK+18] Benjamin Hou, Nina Miolane, Bishesh Khanal, Matthew C. H. Lee, Amir Alansary, Steven McDonagh, Jo V. Hajnal, Daniel Rueckert, Ben Glocker, and Bernhard Kainz. Computing CNN Loss and Gradients for Pose Estimation with Riemannian Geometry. pages 1–9, 2018.

[HMO16] Tomáš Hodn, Jií Matas, and Štpán Obdržálek. On evaluation of 6D object pose estimation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9915 LNCS:609–619, 2016.

[HR78] D. Harrison and D.L. Rubinfeld. Hedonic Housing Prices and the Demand for Clean Air. *Journal of Environmental Economics and Management*, 5:81–102, 1978.

[Hub92] Peter J. Huber. *Robust Estimation of a Location Parameter*, pages 492–518. Springer New York, New York, NY, 1992.

[Huy09] Du Q. Huynh. Metrics for 3D rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[HZL+15] Tomas Hodan, Xenophon Zabulis, Manolis Lourakis, Stepan Obdrzalek, and Jiri Matas. Detection and fine 3D pose estimation of texture-less objects in RGB-D images. *IEEE International Conference on Intelligent Robots and Systems*, 2015-Decem:4421–4428, 2015.

[IR96] Sandy Irani and Prabhakar Raghavan. Combinatorial and experimental results for randomized point matching algorithms. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, SCG '96, pages 68–77, New York, NY, USA, 1996. ACM.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 12 2014.

[KCJ+] Wei Ke, Jie Chen, Jianbin Jiao, Guoying Zhao, and Qixiang Ye. SRN: Side-output Residual Network for Object Symmetry Detection in the Wild.

[KMT+16] Wadim Kehl, Fausto Milletari, Federico Tombari, Slobodan Ilic, and Nassir Navab. Deep learning of local RGB-D patches for 3D object detection and 6D pose estimation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9907 LNCS:205–220, 2016.

[KMT+17] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:1530–1538, 2017.

[KNO11] Mikko Kytö, Mikko Nuutinen, and Pirkko Oittinen. Method for measuring stereo camera depth accuracy based on stereoscopic vision. (May 2014):78640I, 2011.

[Kot07] Sotiris B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, 31:249–268, 2007.

[KSH12a] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[KSH12b] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[KTD+16] Rigas Kouskouridas, Alykhan Tejani, Andreas Doumanoglou, Danhang Tang, and Tae-Kyun Kim. Latent-Class Hough Forests for 6 DoF Object Pose Estimation. pages 1–14, 2016.

[Lab99] Pacific Marine Environmental Laboratory. UCI machine learning repository, 1999.

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[LD60] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):pp. 497–520, 1960.

[LE06] Gareth Loy and Jan-Olof Eklundh. Detecting symmetry and symmetric constellations of features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 508–521, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Lee03] John M. Lee. *Smooth Manifolds*, pages 1–29. Springer New York, New York, NY, 2003.

[LF05] Vincent Lepetit and Pascal Fua. Monocular Model-Based 3D Tracking of Rigid Objects: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2005.

[LMA07] Pierre M. Larochelle, Andrew P. Murray, and Jorge Angeles. A Distance Metric for Finite Sets of Rigid-Body Displacements via the Polar Decomposition. *Journal of Mechanical Design*, 129(8):883, 2007.

[LP01] Urs Lang and Conrad Plaut. Bilipschitz embeddings of metric spaces into space forms. *Geometriae Dedicata*, 87(1):285–307, Aug 2001.

[LPY+17] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head R-CNN: in defense of two-stage object detector. *CoRR*, abs/1711.07264, 2017.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[MN03] Niloy J. Mitra and An Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, SCG '03, pages 322–328, New York, NY, USA, 2003. ACM.

[Moa02] M. Moakher. Means and averaging in the group of rotations. *SIAM Journal on Matrix Analysis and Applications*, 24(1):1–16, 2002.

[MS10] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(12):2262–2275, December 2010.

[Mur13] Kevin P. Murphy. *Machine learning : A probabilistic perspective*. MIT Press, 1 edition, August 2013.

[OBLS14] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[PGA+16] Cristiano Premebida, Luis Garrote, Alireza Asvadi, A. Pedro Ribeiro, and Urbano Nunes. High-resolution LIDAR-based depth mapping using bilateral filter. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 2469–2474, 2016.

[Pol00] Jan Poland. Three different algorithms for generating uniformly distributed random points on the N-sphere. 2000.

[QSMG16] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016*, pages 601–610, dec 2016.

[RBB09] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D registration. *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009.

[RCT13] R. Rios-Cabrera and T. Tuytelaars. Discriminatively trained templates for 3d object detection: A real time scalable approach. In *2013 IEEE International Conference on Computer Vision*, pages 2048–2055, Dec 2013.

[RE10] Daniele P. Radicioni and Roberto Esposito. *BREVE: An HMPerceptron-Based Chord Recognition System*, pages 143–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[RHGS15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June-2015:1–9, 2015.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017.

[SWW73] A.A. Sagle, R.E. Walde, and R. Walde. *Introduction to Lie Groups and Lie Algebras*. Pure and Applied Mathematics; A Series of Monographs and Tex. Academic Press, 1973.

[T. 18] E Brachmann W Kehl A Buch D Kraft B Drost J Vidal S Ihrke C Sahin F Manhardt F Tombari T Kim J Matas C Rother T. Hodan F. Michel. BOP: Benchmark for 6D Object Pose Estimation. *European Conference on Computer Vision (ECCV)*, 2018.

[Tho17] Martin Thoma. Analysis and Optimization of Convolutional Neural Network Architectures. (August), 2017.

[TSF17] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-Time Seamless Single Shot 6D Object Pose Prediction. 2017.

[Tur] Greg Turk. The stanford bunny. "https://www.cc.gatech.edu/~turk/bunny/bunny.html".

[VBVC17] Marco Venturelli, Guido Borghi, Roberto Vezzani, and Rita Cucchiara. From Depth Data to Head Pose Estimation: a Siamese approach. 2017.

[VLM] Joel Vidal, Chyi-Yeu Lin, and Robert Martí. 6D Pose Estimation using an Improved Method based on Point Pair Features.

[Voi18] John Voight. *Quaternion algebras*. Springer Graduate Texts in Mathematics series, 2018.

[WL15] Paul Wohlhart and Vincent Lepetit. Learning descriptors for object recognition and 3d pose estimation. *CoRR*, abs/1502.05908, 2015.

[WWH97] P. Wunsch, S. Winkler, and G. Hirzinger. Real-time pose estimation of 3d objects from camera images using neural networks. 4:3232–3237 vol.4, April 1997.

[YLCJ16] Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2241–2254, 2016.

[ZK15] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4353–4361, 2015.

[ZKR+18] Yinda Zhang, Sameh Khamis, Christoph Rhemann, Julien Valentin, Adarsh Kowdle, Vladimir Tankovich, Michael Schoenberg, Shahram Izadi, Thomas Funkhouser, and Sean Fanello. Activestereonet: End-to-end self-supervised learning for active stereo systems. 07 2018.