Georg J. Vienna, BSc

Aligning Model Databases to Real World Objects

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg

Institute for Computer Graphics and Vision

Dipl.-Ing. Dr.techn. Denis Kalkofen

Graz, September 2016

Dedication

I would like to thank everybody who allowed me to finish this master thesis.

Special thanks go to Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg, who allowed me to write this thesis at the Institute for Computer Graphics and Vision of the Graz University of Technology and Dipl.-Ing. Dr.techn. Denis Kalkofen who gave me a lot of advice and guided me through this thesis.

Furthermore, I would like to thank all friends and colleagues, who supported and accompanied me through my whole student live.

The biggest thank goes to Nicola Scholl, for supporting me and especially proof reading this whole thesis. You are awesome!

I would also like to thank my parents, my siblings and especially my nieces, they always believed in me and are a big pillar in my life.

Abstract

This thesis attempts to discuss an implementation for the semantic scene reconstruction. The goal is to analyse a scanned scene, detect known objects, extract them and replace them with a virtual three dimensional representation of the object. The replacement should be exchangeable, meaning that it should be possible to align every model of the objects class with the scanned object. In the end an application is implemented which allows to scan a scene and replace objects with their three dimensional model counterpart. A possible use case of this application could be a refurnishing helper.

Contents

1 Introduction

The advent of RGB-D devices brought a lot of new possibilities in terms of computer vision and especially object recognition. RGB-D devices provide in addition to the colour information, like the one returned by common cameras, also the depth information of every pixel of the colour image. Each shot of such a device provides therefore a 2.5 dimensional representation of the scanned scene. This additional depth information allows reasoning more easily about the components of the scene. While various approaches exist to detect objects in a colour only image, it is hard to tell the exact pose of an object only by looking at the colour information, which is necessary for a semantic scene reconstruction.
Semantic scene reconstruction refers to a method to detect components of a scene and to semantically label them. By labelling them, the scene can be virtually reconstructed by using three dimensional models of the objects (Figure 1).

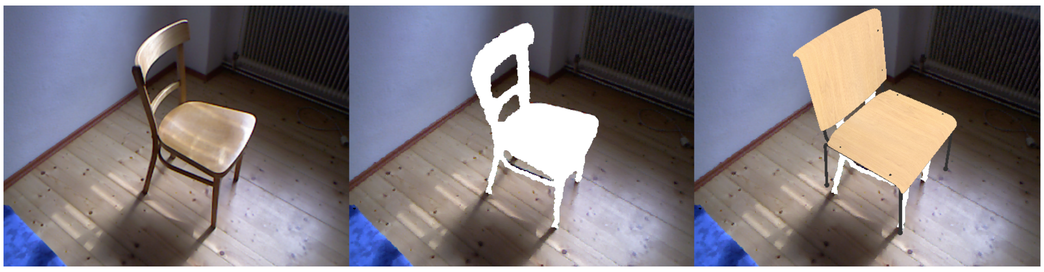


Figure 1: Colour image of a single shot scan (left), without the detected object (middle) and scene with replaced object (right).

An application for such a reconstruction is for example in the field of robotics. By analysing a scene and semantically reconstruct it, the robot can reason about the scene with the three dimensional representation of the real objects (Figure 2). The robot can therefore use the reconstructed scene to interact with the real object or how to move between the objects. Other use cases could be a remote support interface, where components of the scene are virtually overlaid with the three dimensional representation. By interacting with this representation reparation tasks can be explained more easily.
With the rise of affordable RGB-D devices, several consumer applications are possible as well. One application could be to the rearrangement or swapping of different pieces of furniture in semantically reconstructed scenes. Therefore, allowing to virtually remodel rooms.

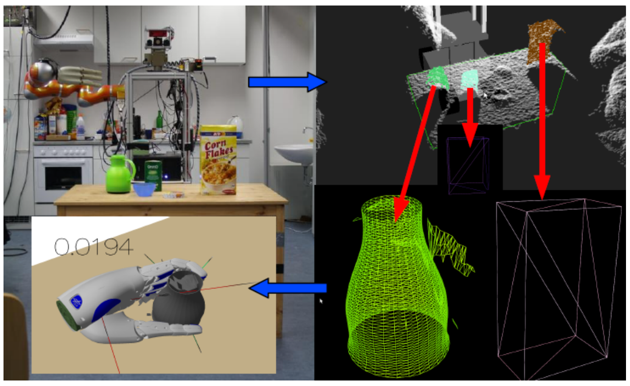


Figure 2: The scene is first scanned by the robot (top left). By analysing the point cloud (top right), the respective mesh models can be computed (bottom right) and can then be used for the grasp computation (bottom left). [20]

This last example for a possible application will by used to show the proceeding and the different methods of the semantically reconstruction in this thesis, where the main motivation is to semantically label a scene and align three dimensional representation of the detected objects within the scene. This master thesis will focus on the implementation of a prototype for the analysis of a scene. The result will be a small application which can be used to classify objects in a captured scene and replace them with an arbitrary three dimensional model of an object of the same class.

The goal is a simple application which should require as minimal user interaction as possible. The user should be able to scan a scene right inside the application. The scans are single shot scans, since this way even inexperienced user should be able to scan the scene, containing every object.
Additionally, saving, reloading and opening of several scans even made from sources outside the application should be possible.

After a scene is scanned the object detection and alignment should happen automatically or with as little interaction from the user as possible. It should be possible for the user to refine a step manually at any time. Even after objects are detected and aligned by the application the user can still adjust the position as well as the orientation of every model.
The user has also the possibility to add models on their own if the corresponding scanned object was not detected or for other reasons.

In the end the application should be user-friendly and easy to use. There are many use cases where such an application is helpful. A big use case is certainly in the field of interior design. With the help of this application an user could scan for example their living room and redecorate it in different styles, change furnitures, move furniture around all or add new furniture. Since a lot of companies provide free available three dimensional models of their furniture, a customer could use this application to view how the furniture would look in their own four walls, without actually buying it. This can help also the user to calculate which furniture and especially which size the furniture should have.

In most home planer the room needs to be created by the user too, requiring him therefore to measure the room, which is not only a time consuming activity, but also error prone. Instead, the only data needed for this application is a scan of the room, which is as simple as taking a photograph. Having the scan and the models in a real world size, it is easy for the user to figure out, if the given model would fit in the wanted position. A task which would otherwise require the user to measure and possibly convert it to the right length unit.

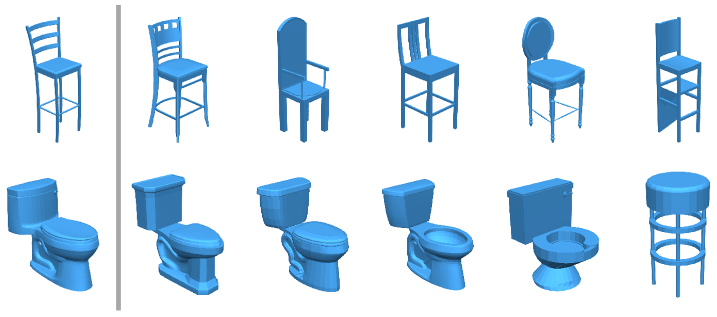


Figure 3: Query object in the first column and the 5 top database model matches. [17]

Other use cases of this application include the possibility to virtualize a scene. This could be used in fields such as arts, where it is easier, cheaper and especially faster to create various prototypes as a small real world model, which can then be scanned in, to create the virtual counterpart of them.

The general nature of this thesis allows for a lot of other applications.
An application, which is often named in various publications [17][19], is the database retrieval (Figure 3). Database retrieval describes the idea to search similar three dimensional models of a query object and also to align it so that the query object and the database models can be seen from the same view.

2 Related Work

The recognition and alignment of objects is an active research area. Several approaches exist, which take different attempts to solve the problem for special use cases. Various methods exists which use geometric, colour and other informations to achieve this goal. This chapter will describe some current state-of-the-art methods.
The two widely used pipelines - local and global recognition pipeline - are discussed, which are foremost used for geometric detection and description of objects.
The last section of this chapter will focus on methods working on images and therefore only colour and depth images.

2.1 Local Features

Local refers to the fact, that local feature descriptors are used. The local recognition pipeline is made out of several steps (Figure 4).

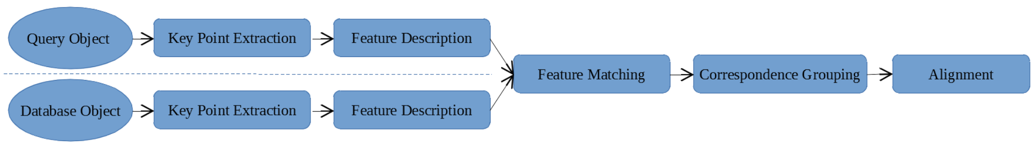


Figure 4: Different steps used in the local recognition pipeline.

The first step is the key point extraction step, followed by the feature description step. These two steps are made for both, the query object and the database objects. In the next step the resulting features are matched and corresponding features are grouped. Finally, these correspondences are used to find the relative alignment.

2.1.1 Key Point Extraction

The key point extraction step is needed to find interest points, which can later be used as starting point to describe the surrounding region.
Several key point detectors, such as Intrinsic Shape Signature [39] and Harris3D [28], are available. The main attributes of a key point detector are repeatability and distinctiveness. Repeatability means that with the same or similar input the detected key points should be the same. Distinctiveness

refers to the property, that the detected key points should be easy to describe and match.
Key points can be detected using various methods. In [16][1] key points are detected by down sampling the input uniformly and taking the resulting points as key points. However, this does not guarantee at all, that distinct key points are found.

The Intrinsic Shape Signature (ISS) uses the principal curvature to detect key points. This is done with the help of a covariance matrix defined as:

$$M(p_i) = \frac{1}{\sum_{j=1}^{k} w_i} \sum_{j=1}^{k} w_i (p_j - p_i)(p_j - p_i)^T,$$

where p_j are points lying within the search distance of the current point p_i and the weighting factor w_i, which is defined as:

$$w_i = \frac{1}{\|p_j\|}.$$

This matrix, more precisely the ordered eigenvalues thereof, are used to discard points where a repeatable local reference frame cannot be defined. The remaining points are then rated according to their saliency (the magnitude of the third eigenvalue). By using non maxima suppression the points are again reduced, such that only points with a high principal curvature are returned (Figure 5).

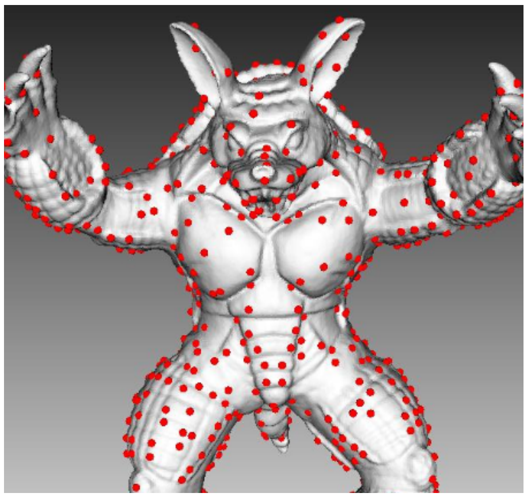
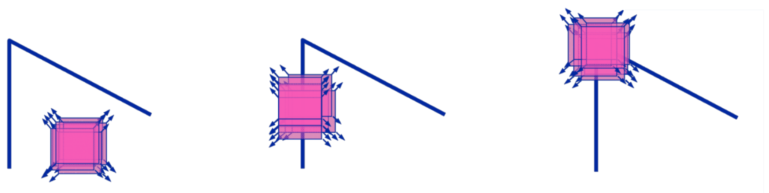


Figure 5: Key points (red) extracted with ISS. [30]

Another widely used detector is the three dimensional specialisation of the two dimensional Harris corner detector described in [12]. The basic idea of the Harris corner detector can be thought of as looking at an image through a small window. By shifting the window around changes in the images can be observed and classified as corner or edges (Figure 6).



(a) flat region: no change in all directions

(b) edge: no change along the edge direction

(c) corner: significant change in all directions

Figure 6: Harris detector: by looking at the image through a small window, the changes in the image can be observed and classified. [7]

The three dimensional version uses the surface normals, instead of the image gradients, to detect changes in directions.

2.1.2 Feature Description

The detected key points are now used as starting points for the local feature descriptors. The goal of a local feature descriptor is to describe the surface around a key point as general as possible and to encode it into a comparable representation, while still preserving the characteristic of the surface.

First basic concepts common to some descriptors are described.

Local Reference Frame Local descriptors take a feature point which can be easily detected and describe the surface around this point. As the local descriptors are independent from each other a local coordinate system has to be introduced, in which the descriptor is computed. This coordinate system should be fixed in all axes, unique, unambiguous and easy to construct and reconstruct. Some early descriptors ([16][9]) fixed the coordinate system on only one axis along the normal of the feature point, because of this it is necessary to compare rotated versions of their descriptors against each other. Later [32] and [31] fixed two axes and consequently also the third axis by introducing a local reference frame. To create the frame of a feature point

$\mathbf{p}$ a weighted covariance matrix $\mathbf{M}$ is constructed using points lying in the radius R around $\mathbf{p}$:

$$\mathbf{M} = \frac{1}{\sum_{i:d_i \leq R}(R - d_i)} \sum_{i:d_i \leq R} (R - d_i)(\mathbf{p_i} - \mathbf{p})(\mathbf{p_i} - \mathbf{p})^T$$

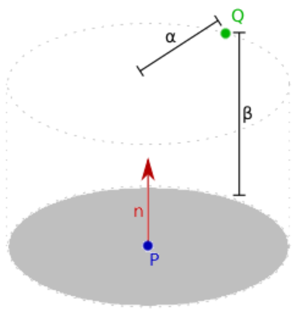
with the euclidean distance (l_2 distance)

$$d_i = \|p_i - p\|_2.$$

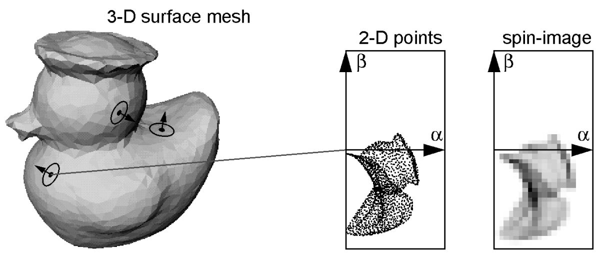
From this matrix the eigenvalues are computed. The normal direction is now given by the eigenvectors corresponding to the smallest calculated eigenvalue. This direction together with the other two eigenvector provides an estimate of the three unit vectors of the reference frame. Due to the sign ambiguity of the eigenvalue decomposition another step is required to make them unambiguous. This happens by reorient the first, the one with the biggest eigenvalue, and the third, the one with the smallest eigenvalue, so that they are coherent with the majority of the presenting vectors. The last unit vector is computed as the cross product of the two others.

In the following, some widely used local feature descriptors are listed in chronological order.

SpinImages The SpinImage descriptor is a local shape descriptor introduced 1999 by Johnson et al. [16]. It uses a point and its normal to define a partial coordinate system. In this coordinate system the other points are represented using two cylindrical coordinates. The radial coordinate α is the perpendicular distance between the point and the surface, the elevation coordinate β is the distance from the point to the tangent plane (Figure 7a). The coordinate system is only fixed by the surface normal, so the third angular coordinate cannot be used. Thus, all possible rotations have to be compared if the descriptor is matched. To create a SpinImage all points in the supporting region of the point p are described in the cylindrical coordinate system shown in Figure 7a and inserted in a 2D accumulator, where α and β are the bin indices. The bins are then bi-linearly interpolated to smooth out the contribution.
To match two models the SpinImage's of the vertices of the models are computed and stored in a database. A random point of the scene is taken and its SpinImage is calculated. The SpinImage is then matched against the database by using the l_2 distance. This procedure is repeated for about 100



(a) Coordinate system used to create a SpinImage.



(b) Surface mesh and the resulting SpinImage of one point. [16]

Figure 7: Coordinate system and creation of SpinImage's

point correspondences, which are then grouped and removed from outliers. Several other descriptors [6][9] where introduced, which provide better results and have a better performance than SpinImage. Therefore, this descriptor is not considered in this thesis.

3D Shape Context (3DSC) The 3D Shape Context descriptor was introduced in 2004 by Frome et al. [9]. This local shape descriptor creates a spherical grid around the key point, where the north pole of the sphere lies along the surface normal of the key point. The bins of the descriptor are created by dividing the height and the azimuth of the sphere uniformly. Radial the bins are logarithmically divided. Each bin gets filled with the weighted count of points that fall into the bin. The weighted count is calculated with:

$$w(p_i) = \frac{1}{\rho_i \sqrt[3]{V(j,k,l)}}$$

where ρ_i is the local point density around the bin. Each rotation around the north pole of the description has to be checked, since the reference frame is only fixed along the surface normal. The number of divisions are free, so the length of the descriptor is given by **J**x**K**x**L**.
The found descriptors are matched using the l_2 distance. To match two collections of descriptors a cost is calculated by summing up the minimal distance between a descriptor p_m in the query collection S_q and another descriptor q_k in the database collection S_i:

$$\text{cost}(S_q, S_i) = \sum_{k \in \{1,\ldots,K\}} \max_{m \in \{1,\ldots,M\}} \text{dist}(q_k, p_m)$$

3DSC is superseded by USC, it was therefore not taken into consideration.

Unique Shape Context (USC) The USC descriptor [31] is derived from the previously described 3DSC and introduces additionally a fixed frame, so that the descriptor is fixed in all three axes. The reference frame is created by eigenvalue decomposition of the weighted covariance Matrix $\mathbf{M}$ of points within the neighbourhood of the feature point (see 2.1.2). After the eigenvalue decomposition there is still a sign disambiguation, so the sign of biggest and smallest eigenvalue are reoriented so that they are coherent with most of the representing vectors. The last axis is obtained with the cross product of the two other axes. Afterwards the same procedure as in 3DSC is used. The calculation of the reference frame brings no big overhead, in fact it reduces the overall computation time of the matching algorithm as not all rotations have to be checked.
This descriptor is not taken into consideration in this thesis, because according to Alexandre [3] SHOT has a better recognition rate.

Signature of Histograms of OrienTations (SHOT) Signature of Histograms of Orientations is a local surface descriptor introduced by Tombari et al. [32]. It creates a repeatable reference frame (see 2.1.2) around a key point by eigenvalue decomposition of $\mathbf{M}$. The resulting vector is still not repeatable as the sign could be different. Therefore, the vector which has the most directional support is selected. From this reference frame a spherical grid is created, by dividing the height in 2 parts, the azimuth in 8 parts and in 2 radial parts. From each part a histogram with 11 bins is created by calculating the angle between the point normal and the feature point. Consequently, the length of the descriptor is 352 ($2 \cdot 8 \cdot 2 \cdot 11$), plus 9 for storing the reference frame.
The found descriptors can be matched using the l_2 distance.
This descriptor was successfully used in Tombari et al. [32] to match and align two representations of the same object. It is therefore taken into consideration to be used to match also similar objects against each other.

Signature of Histograms of OrienTations COLOR (SHOTCOLOR) SHOTCOLOR [33] is an extension of SHOT. Additionally, to the geometric information of a point it takes the colour information to calculate the descriptor. The colour difference l is calculated in RGB or CIE colour space:

$$l(Rq, Rp) = \sum_{i=1}^{3} \|Rp(i) - Rq(i)\|,$$

where R_q,R_p are colour triplets (e.g RGB) of two points.
Even-tough this descriptor is an improvement of the previously introduced SHOT descriptor, it can not be used because it requires the colour information of the objects to match.

Rotation-Invariant Feature Transform (RIFT) The Rotation-Invariant Feature Transform [18] is a local surface and intensity descriptor. It takes concepts from the 2D computer vision. It fits a circular patch around the point and divides it into concentric rings (Figure 8a). For all points lying in such a ring a histogram is populated (Figure 8b). The histograms are created using the angle between the orientation of the gradient and the vector pointing outward from the centre, to make it rotational invariant.

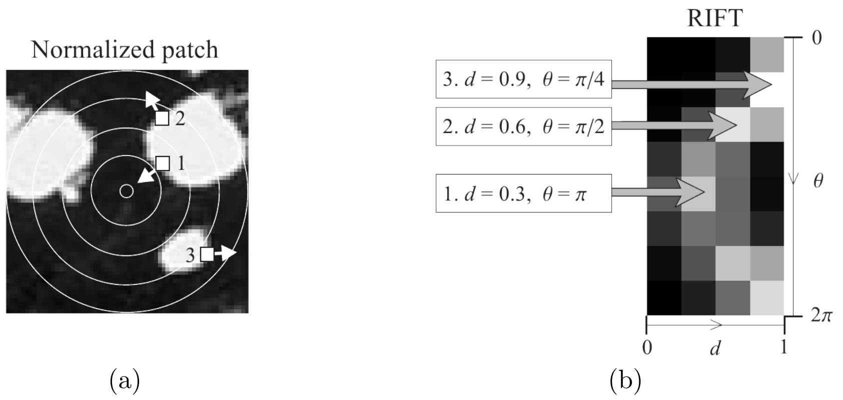

Normalized patch
1
2
3
RIFT
3. $d = 0.9,\ \theta = \pi/4$
2. $d = 0.6,\ \theta = \pi/2$
1. $d = 0.3,\ \theta = \pi$
0
θ
2π
0
d
1
(a)
(b)

Figure 8: Three sample points in the normalized patch (a) map to three different locations in the descriptor (b) [18]

The found descriptors can be matched using the l_2 distance.
This descriptor uses the intensity to describe the surrounding of a key point. Since the goal is to match arbitrary objects, which do not necessarily have the same colour or intensities, this descriptor will not be taken into consideration.

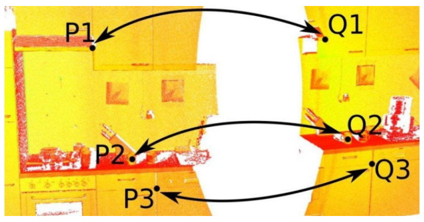
Point Feature Histogram (PFH) The Point Feature Histogram descriptor is a local surface descriptor introduced by Rusu et al. [23]. The descriptor encodes a local surface by using the surface normals and the curvature estimate. It uses two points p and q to create a fixed reference frame, with axes u,v and w, in one of the points p:

$$u = \text{surface normal at } p$$
$$v = u \times \frac{p-q}{\parallel p-q \parallel}$$
$$w = u \times v$$

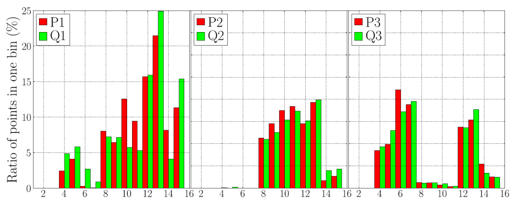
Based on this reference frame the following angular values are computed:

$$\beta = v \cdot n_q$$
$$\gamma = u \cdot \frac{p-q}{\parallel p-q \parallel}$$
$$\delta = \arctan(w \cdot n_q, u \cdot n_q)$$

with n_q = surface normal at q.
The three angular values (β, γ, δ) combined with the Euclidean distance ($\parallel p-q \parallel$) between the points are used to describe the relationship between the two points. They are used to creating a histogram for each point using the relation of the point with its k neighbours. The histogram is binned into 16 bins (Figure 9b). The combination of these histograms is the final descriptor.



(a)



(b)

Figure 9: Sample points of two different point clouds and their feature histograms [23]

The computation of this descriptor is expensive as each point is compared with its k neighbours, leading to $O(n \cdot k^2)$ computations on a point cloud with n points.
The found descriptors can be matched using the l_2 distance.
This descriptor is superseded by FPFH , it is therefore not taken into consideration.

Point Feature Histogram RGB (PFHRGB) The Point Feature Histogram RGB [15] is an extension of the PFH descriptor. It uses the available RGB information of the points, therefore doubling the number of bins in the histograms. The additional bins are computed using the ratio between each colour channel of the RGB values of the points.
The found descriptors can be matched using the l_2 distance.
Since this descriptor uses the colour information, it cannot be used to match arbitrary objects.

Fast Point Feature Histogram (FPFH) The Fast Point Feature Histogram is an enhancement of the PFH. It uses the same methods as introduced by PFH, but tries to reduce the computational complexity. The complexity is reduced by discarding the 4th (distance) value and the decorrelation of the remaining histogram dimensions.
The found descriptors can be matched using the l_2 distance.
This descriptor provides a fast and reliable way, to describe a key point and the surrounding. Therefore, it is considered in this thesis as key point descriptor.

Radius-Based Surface Descriptor (RSD) The Radius-Based Surface Descriptor is a local surface descriptor introduced by Marton et al. [20]. To calculate it each point in the neighbourhood of the feature point is used to fit a sphere in so that the points lie on the surface of the sphere and their normals are normals of the sphere. From these spheres the sphere with maximum and minimal radius are saved.
To match two descriptors against each other the radii are compared.
This descriptor has the disadvantage that it gives the best performance with objects, which have a single body. On objects which have several small parts (e.g. a handle or chair legs), the recognition rate can be really low. Therefore, this descriptor is not taken into consideration.

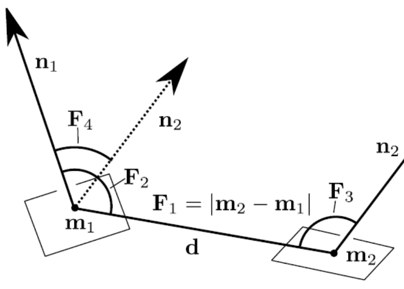
Point Pair Feature (PPF) The Point Pair Feature descriptor introduced 2010 by Drost et al. [6] uses oriented point pairs, their relative position and orientation to describe an object (Figure 10a). The point pair feature **F** is defined as:

$$\mathbf{F}(m_1, m_2) = (d, \sphericalangle(n_1, d); \sphericalangle(n_2, d); \sphericalangle(n_1, n_2))$$

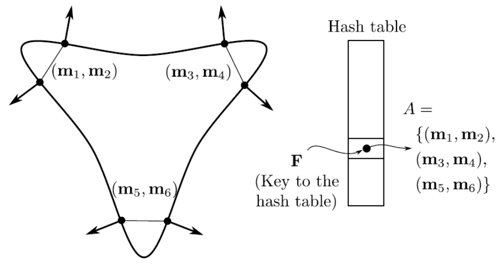
where m_1, m_2 are the points, n_1, n_2 their normals and d the distance vector between the points

$$d = \|m_2 - m_1\|_2.$$

$\sphericalangle$ means angle between the vectors.



(a) $\mathbf{F} = (\mathbf{F}_1; \mathbf{F}_2; \mathbf{F}_3; \mathbf{F}_4)$



Hash table
$(\mathbf{m}_1, \mathbf{m}_2)$
$(\mathbf{m}_3, \mathbf{m}_4)$
$(\mathbf{m}_5, \mathbf{m}_6)$
$\mathbf{F}$
(Key to the hash table)
$A = \{(\mathbf{m}_1, \mathbf{m}_2), (\mathbf{m}_3, \mathbf{m}_4), (\mathbf{m}_5, \mathbf{m}_6)\}$

(b) Point pairs with similar feature vector are saved in the same slot.

Figure 10: Point pair feature **F** of two oriented points (a) and global model description using a hash table (b). [6]

To build the descriptor of a model the point pair feature of point pairs that lie on the model surface are calculated. Then the feature vector is sampled and equal feature vectors are grouped and saved in a hash table with **F** as index.
During matching a random point pair is selected and the feature **F** is calculated. Using the hash table (Figure 10b) it is easy to find the matching points and to align the models.
According to Alexandre [3] the FPFH descriptor provides a better recognition rate, than this descriptor. Therefore, PPF will not be used.

Normal Aligned Radial Feature (NARF) The Normal Aligned Radial Feature descriptor is a local surface descriptor, which uses range images as input. It was introduced by Steder et al. [29] in 2011. To detect stable points for every point in the range image the surface changes in its neighbourhood and the dominant direction of this change is computed. Stable points are then found by comparing the direction with those of the neighbours. Good key points will be points that lie near a object's corner.
The NARF descriptor creates a local range patch around the key point. Then a star pattern of beams is overlaid on the point and for every beam

a value of how much the surface changes along the beam is computed. The concatenation of these values produces the descriptor. The descriptor is still not rotational invariant as it can be rotated around the normal, therefore the descriptor is calculated for each possible rotation and the angle with the highest value is considered the dominant orientation.
This descriptor is not taken into consideration in this thesis, since it uses range images.

Tri-Spin-Image (TriSI) The TriSI descriptor [38] is a local surface descriptor. It takes the surface around a key point and uses it to create a local reference frame. On each axis of the reference frame a SpinImage of the local surface is created. The descriptor is the concatenation of the three SpinImage's $TriSI = SI_1, SI_2, SI_3$. Because of this the size of the descriptor is too big, therefore the descriptor gets projected to a principal component analysis subspace.
For performance reasons this descriptor was not taken into account.

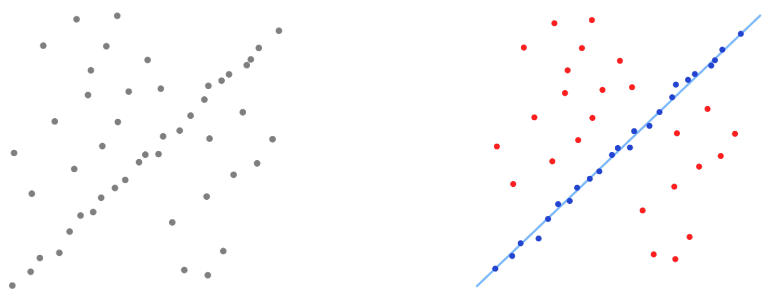
Rotational Projection Statistics (RoPS) The RoPS descriptor [11] uses the local surface around a point to create a local reference frame that is unambiguous. The local surface is then brought in this reference frame, rotated around the x axis by θ and projected to each axis plane, which are then partitioned, leading into 3 matrices **LxL**. These matrices are then encoded into several statistics and concatenated to form a sub descriptor for this rotation. The procedure is repeated for each axis with a different θ. The typical length of RoPS is 135.
This descriptor is not taken into consideration, since in most cases it provides nearly the same results as the previously introduced SHOT.

2.1.3 Feature Matching and Alignment

The feature description of each feature of the query object needs to be matched against the features of the database object. Once all correspondences between the features has been found, these correspondences need to be grouped. This grouping is done by enforcing geometrical consistency between the key points of all correspondences.

With the given correspondences a good alignment, which suffices all correspondences needs to be found. The correspondences grouping does not guarantee, that every correspondence in the group is consistent to one single

transformation. Therefore, another step is needed to check if every correspondence leads to the same transformation and to find the transformation which gives the most consensus across the correspondences.
To find such a consensus several methods exists. The random sample consensus (RANSAC) [8] tries to find a transformation using only a few random correspondences. If the error of all correspondences using the computed transformation is small the transformation is taken as final transformation, else other random samples are used (Figure 11).


(a) Data set with many outliers.

(b) Line found using RANSAC.

Figure 11: Example using the RANSAC method to fit a line into a data set of points. [35]

Since the overall computational complexity of the alignment using the standard RANSAC method is high, Rusu et al. [26] try to reduce this complexity by using a similar approach. The sample consensus initial alignment (SAC-IA) uses three steps to find the best alignment of a set of features. In the first step n random features of the query object are selected, where each new feature sample should lie outside a minimal distance of the other samples. In the next step for each of these samples matching features in the database object are searched. A random feature of the matching feature is taken and the two samples are considered correspondences. With this list of correspondences a rigid transformation and an error metric is computed in the last step. As error metric the Huber penalty measure is used:

$$L_h(e_i) = \begin{cases} \frac{1}{2}e_i^2 & \|e_i\| \leq t_e \\ \frac{1}{2}t_e(2\|e_i\| - t_e) & \|e_i\| > t_e \end{cases}$$

These three steps are repeated until the transformation with the best error metric is found. SAC-IA provides a fast and efficient way to find a good alignment between two feature sets (Figure 12).

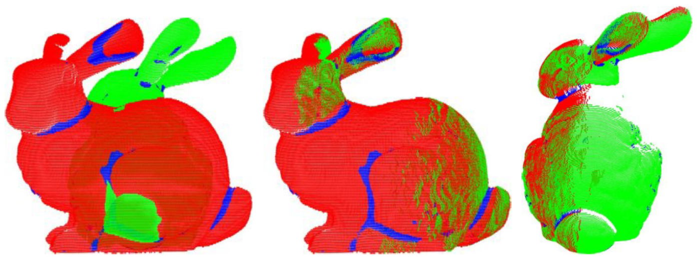


Figure 12: Alignment using SAC-IA. Left unaligned partial views, middle and right alignment result. [26]

2.2 Global Features

While the local recognition pipeline, tries to detect an object by matching database objects to the scene, the global recognition pipeline tries to find a matching object of a certain query object in the database.
Since the global feature descriptors work only in terms of an object, the first step in the global recognition pipeline is to get a clean object from the scanned scene. After the clean object is obtained a descriptor of the object as a whole is calculated. The next step depends on the descriptor. If the features of the descriptor can be used to obtain a transformation between the query and the database object, then the features are matched and a transformation is searched. If the descriptor can only be used to classify the object, then a further step is needed, to align the classified object with a database object of the same class (Figure 13).

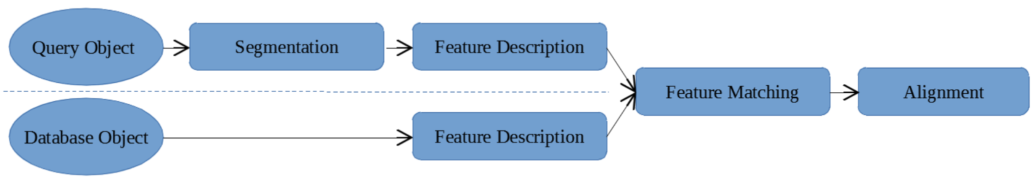


Figure 13: Different steps used in the global recognition pipeline.

2.2.1 Segmentation

Segmentation (or clustering) of a scene refers to the methods used to break a scene down into smaller parts or sections. The ideal goal of a segmentation is to retrieve a cluster which contains one object exclusively. Several algorithms exists which try to perform clustering on an arbitrary point cloud.
The **euclidean segmentation** as described in [25] uses a specialisation of the flood fill algorithm [34]. The flood fill algorithm works, by taking a starting point and a search criterion. Starting from the start point, each neighbour which fulfils the search criterion is visited. This is done recursively, so in the end all neighbours which are connected and fulfil the search criterion are visited once. Figure 14 shows an example for the flood fill algorithm using colours as search criterion.

The euclidean segmentation is derived as follows. Starting from an initial point p_i of the input cloud P every point p_j which fulfils the criterion

$$\min \|p_i - p_j\|_2 \geq d_{th},$$

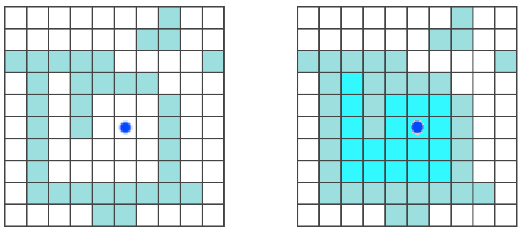


Figure 14: Starting from a initial point (blue), all connected white tiles are coloured in turquoise.

where d_{th} is the maximum allowed distance threshold, is added to a cluster. This step is repeated until all points of the point cloud P are port of the list point clusters.

An improvement of the euclidean segmentation is the **region based segmentation** described in [25]. In this algorithm the points are not only clustered by the minimal distance criterion, but also by a criterion regarding the change of normal direction between points. The additional criterion is defined as:

$$\arccos(\langle \vec{n_i}, \vec{n_j} \rangle) \leq \alpha_{th},$$

where n_i, n_j are the normals of the points and α_{th} maximal allowed angle difference between points in the same cluster.

A different approach is taken by the **minimum cut (min-cut) based segmentation** presented in [10]. The min-cut based segmentation algorithm performs binary segmentation, by dividing the point cloud into two separate clusters (Figure 15). One cluster is considered the foreground cluster which contains only points of the object to be extracted. The other cluster contains only points which does not belong to the object and are therefore considered as background. The method takes as input a point which is known to be around the centre of the object and a search radius which is approximately the size of the object to be extracted. First a k-nearest neighbours graph of the input cloud is constructed and weights are assigned to each edge of the graph. The weight of an edge i is defined as

$$w_i = \exp\left(-\frac{{d_i}^2}{\sigma}\right),$$

where d_i is the distance between the points of the edge and σ a normalisation factor. Further each point is assigned a background penalty, which punishes points which are further away from the object centre. The back-

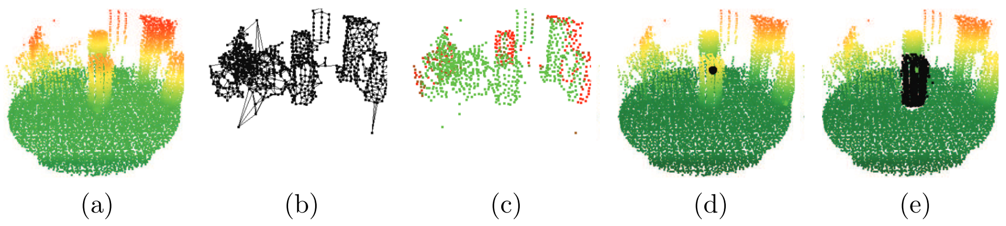


Figure 15: Input point cloud (a) and construction of the nearest neighbour graph (b). Background penalties visualized as red (high) and green (low) points (c). Chosen object's centre (d) and final segmentation using min-cut (e) [10]

ground penalty can be defined as

$$B(p) = \frac{disttocenter(p)}{radius}$$

where *disttocenter* is the distance from the current point to the input centre point and *radius* is the input radius. On the set up graph the minimum cut is now applied, which means that the graph is divided into back- and foreground points by cutting the smallest possible amount of edges. The segmentation can be refined by adding more point constraints (defining them as back- or foreground).

2.2.2 Feature Description

Having a segmented object the global feature descriptors can now be used to describe the object.
In the following, some widely used global feature descriptors are listed.

Curvedness-Orientation-Shape Map On Sphere (COSMOS) COSMOS is a global surface descriptor introduced 1997 by Dorai et al. [5]. Each point gets categorised by the surface index SI and the radius R. Surface patches are created, by combining neighbouring points which have the same category. Using a shape mapping function each patch is represented globally and can be used as a descriptor.
The performance of this descriptor is not sufficing and it is therefore not considered further.

Global Fast Point Feature Histogram (GFPFH) The Global Fast Point Feature Histogram [27] uses the FPFH described earlier to segment the objects into geometric primitives. From that a histogram is created consisting of the geometric primitives of each patch.
This descriptor is not used in this thesis, because the ESF descriptor has a better performance in matching objects.

Global Radius-Based Surface Descriptor (GRSD) The Global Radius-Based Surface Descriptor was introduced in 2011 by Marton et al. [21]. It first divides the input in voxels, which then gets categorised in geometric categories (plane, sphere, edge, ...) using the RSD. The resulting histogram of the categories is then the GRSD descriptor.
Since the ESF provides a better performance than this descriptor, the GRSD is not used in this thesis.

Viewpoint Feature Histogram (VFH) Viewpoint Feature Histogram [24] is based on the FPFH descriptor. It is a global descriptor and produces only one descriptor for a input cloud. First the centroid c of the cloud is computed using the average of each point, then the vector n_c between the centroid and the viewpoint is computed and the local reference frame is computed as followed:

$$\begin{aligned} u &= n_c \\ v &= (p - c) \times u \\ w &= u \times v \end{aligned}$$

The angle values are computed as in PFH, further a new angle is introduced:

$$\alpha = \arccos\left(\frac{\text{point normal}.c}{\parallel c \parallel}\right)$$

The final VFH descriptor is build using the four histograms, leading to a total descriptor length of 263.
Since this descriptor is superseded by the following CVFH descriptor, it is not considered as potential global descriptor in this thesis.

2.2.3 Clustered Viewpoint Feature Histogram (CVFH)

Clustered Viewpoint Feature Histogram [2] is an extension of the Viewpoint Feature Histogram, which is more robust in handling cluttered and occluded scans. It creates stable regions and computes the centroid c and normal n_c of this region. The reference frame is then calculated the same way as in the VFH but this time using c and n_c, resulting in α, γ, δ, β plus a SDC component:

$$SDC = \frac{(c - p_i)^2}{max\{(c - p_i)^2\}}$$

Leading to (α, γ, δ, SDC, β) and length of 308 of the descriptor.
According to Wohlkinger et al. [37] the ESF descriptor provides better results than this descriptor. Therefore, this descriptor is not considered further.

Ensemble of Shape Functions (ESF) The Ensemble of Shape Functions descriptor is a global shape descriptor introduced by Wohlkinger et al. [37] in 2010. To calculate this descriptor a 64x64x64 voxel grid is created from the input. From that 10 histograms with 64 bins each are created. 3 random points are selected and their angle are inserted in a histogram. Additionally, the location of the points are taken into account leading to 3 separate histograms (ON, OFF and MIXED). The same thing is done by choosing 3 random points and their area and with 2 random points and their distance. The last histogram uses the results from the distance MIXED histogram. It calculates the ratio between ON and OFF on the line and saves it to the histogram.
ESF provides good results and also a good performance in comparison to VFH and CVFH [37]. Therefore, it is used as global descriptor in this thesis.

A2h The A2h [17] global shape descriptor divides the point cloud of a scan in 3 bins along the up axis (z). Between each point pair in a bin the angle between the point normals are computed. The angle is than inserted into 50 bins covering 0-360. The frame is not fixed so each rotation around the z axis has to be compared. (In the paper they do this with a preprocessing step, by calculating a 9x9x9 voxel grid and comparing this to get the rotation)
To due the fact that this descriptor requires the up axis of the object to be known, it was not considered further. However, some other parts (voxel grid) of the method described in the paper are used.

LI2015 This global shape descriptor introduced in 2015 by Li et al. [19] uses a combination of 2d and 3d functions. To detect key points a combined

2D/3D corner detection is applied to find feature points.
These key points are described by both their local and global contexts. Further, with this descriptor it is not only possible to match two objects, but also to register an object in respect to another. Because of this the descriptor K_i is composed of two parts a global description of supporting primitives G_i and a local neighbourhood description L_i. For the global part planes and lines are used, where planes are favoured because they are more robust. For each key point the most prominent horizontal plane and the two most prominent vertical planes or lines are saved. The local part differs between database and scanned objects. For the database object key points the local neighbourhood geometry is characterized by a *local unsigned distance function* with the help of this function it can be determined how much the scan lies outside the surface of the database model. The neighbourhood of scanned key points can be incomplete, therefore a grid around the key point is computed and each cell is categorised as *known empty space*, *known occupied space* and *unknown observations*. When computing the grid only points that are connected to the key point are taken into account.
To match two models first the global description is used to align the two key points. Then the distance between the two descriptors $d(K_i, K_j)$ are computed by comparing the local parts.
Since this descriptor can only work on a scene which is completely scanned and not with a single shot scan, it is not considered as descriptor in this thesis.

2.2.4 Feature Matching

Since the global feature descriptor understand the notion of an object and every descriptor describes the object with a vector, the matching of a query object to a database model can be done by comparing the two objects using the Manhattan, the Euclidean or any other distance metric.

2.2.5 Alignment

After the corresponding database object has been found the object's position and scale can be determined by taking the bounding box of both objects and calculating the translation of the middle points and the size differences.
To find the rotation of the query object with respect to the database query either the information returned by the descriptor can be used or other methods need to be used. In [2] a camera roll histogram (CRH) is used, which

is stored along-side the descriptor to find the final pose. The CRH is computed by projecting the normals of each point onto a plane orthogonal to the camera plane. The angles between the projected normals and the up view vector is then used to create the histogram. The angle are discretised into a resolution of 4 degrees, leading to a histogram with 90 bins. The rotation of the query object can now be determined by comparing the CRH saved along with the descriptor and the CRH constructed from the query object. Kim et

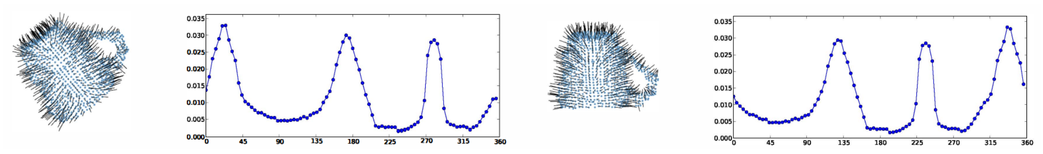


Figure 16: A model with different orientations (left) and the corresponding camera roll histogram (right). [2]

al. [17] uses a density voxel representation of the point clouds to find a good alignment between the two objects. To find it, a voxel grid representation of each possible orientation of the database object is created and matched against the voxel grid of the query object. The rotation is then the rotation of the voxel grid with the smallest error.

2.3 Image based retrieval

A lot of approaches ([4][13]) have been made to recognize an object and to estimate the pose of such an object using images.
HOG (Histogram of Oriented Gradients) [4] tries to detect an object with the help of image gradients. First the image gradients are computed, then the resulting image is divided in small cells. Inside each of these cells a histogram of oriented gradients is calculated. To increase the performance, the local histogram is detected by using the calculated intensity over a bigger block. With the help of this value, all cells in the block are normalized. The concatenation of the histograms of all blocks form the final feature vector.
LineMod (multi-modal LINE) introduced in [13] uses colour and depth information to detect and align objects. By using both informations LineMod creates two so called modalities (Figure 18). The first modality are two dimensional colour gradients found in the colour image. Image gradients provide good and reliable cues for images with texture-less objects. Further, they are also robust to illumination change and noise. The second modality are three dimensional surface normals which are calculated using the depth image. The two modalities are normalized and then quantized into equally

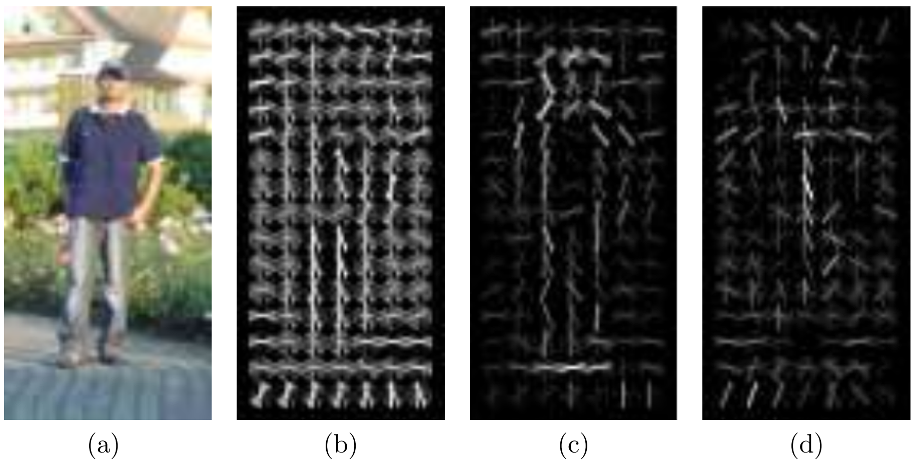


Figure 17: Test image (a) and the computed HOG descriptor (b). (c, d) HOG descriptor weighted by positive and negative weights. [4]

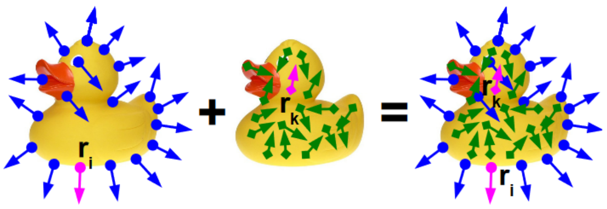


Figure 18: 2D colour gradients, 3D surface normals and the combination as multiple modalities. [13]

distributed bins as shown in Figure 19.
Since the modalities does not provide any interests points, special approaches need to be done to detect an object in an image. The features are first quantized in 8 bins (45°) and then spread around the initial position leading to a binarized image (Figure 20).

Using these binary images response maps can be precomputed with the help of look up tables for each possible query feature. The features are invariant to small translations, since they are previously spread. This also means that during the response map creation only every i'th pixel has to be considered.

Due to the use of modalities obtained from colour and depth information LineMod is able to detect texture-less objects and also to detect objects in a cluttered scene.
Like HoG, LineMod relies on hand-crafted representations of the templates,

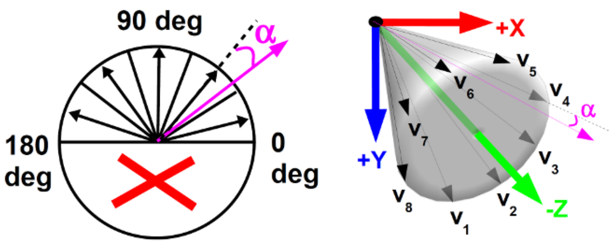
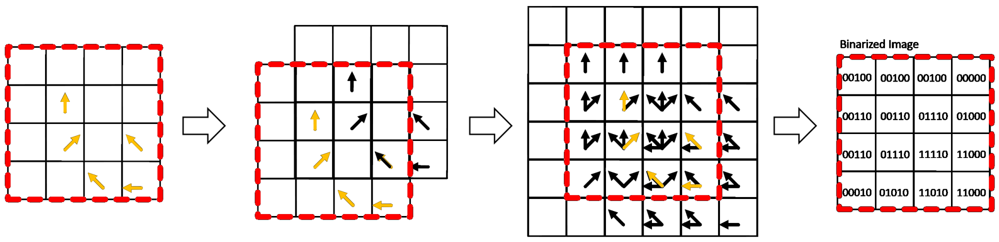


Figure 19: Quantisation of the normalized modalities. [13]



00100	00100	00100	00000
00110	00110	01110	01000
00110	01110	11110	11000
00010	01010	11010	11000

Figure 20: Modalities to binary image conversion. [14]

which are suboptimal compared to statistically learned features.

Wohlhart et al. [36] circumvent that in their approach by using a convolutional network. The convolutional neuronal network is used to calculate 16 dimensional descriptors. These descriptors can then be used to recognize and estimate the pose of a query object. Initially the descriptors of all template poses of an object are computed. These can be used to match the descriptor of an arbitrary image by searching the nearest neighbour of the template pose descriptors. The nearest neighbour can be found by taking the euclidean distance between the template descriptors and the query descriptor.
The descriptor computed by the neuronal network should therefore have two properties. The euclidean distance between two descriptors of the same object should be in relation to the difference of their pose and descriptors of two different objects should have a large distance.
In order to fulfil these requirements, the following cost functions are used to find the network parameters. The triplet cost function uses three samples s_i,s_j and s_k from the training dataset (image data x and pose p), where either two samples (s_i,s_j) are from the same object and the other (s_k) from another object or all samples are from the same object, but two poses p_i, p_j are more similar to each other than the other two (p_i, p_k):

$$c(s_i, s_j, s_k) = max\left(0, 1 - \frac{\|f_w(x_i) - f_w(x_k)\|_2}{\|f_w(x_i) - f_w(x_j)\|_2 + m}\right), \qquad (1)$$

where m is a margin in set to 0.01 and $f_w(x)$ is the output of the CNN for an input image x.
The overall term is then simply the sum of each triplet:

$$\mathcal{L}_{triplets} = \sum_{(s_i,s_j,s_k)\in\mathcal{T}} c(s_i, s_j, s_k). \tag{2}$$

The pair cost function is used to learn the network to be robust against noise or changes in illumination. This is enforced during the training by defining that two poses of the same object with a small distance should also have a small distance in their descriptor:

$$\mathcal{L}_{pairs} = \sum_{(s_i,s_j)\in\mathcal{P}} \|f_w(x_i) - f_w(x_j)\|_2^2. \tag{3}$$

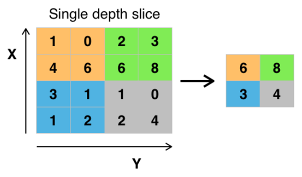
The overall requirement can now be defined as

$$\mathcal{L} = \mathcal{L}_{triplets} + \mathcal{L}_{pairs} + \lambda\|w'\|_2^2, \tag{4}$$

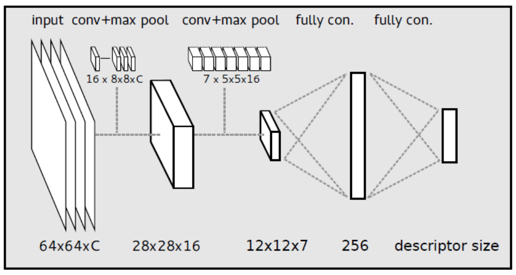
where the last term is a regularization term over the parameters of the network.

The structure of the network as seen in Figure 21b consists of four layers. The first two convolute the input images with a set of filters. Max-pooling which samples the input down (Figure 21a) and rectified linear (ReLU) activation function (Equation 5) defined as:

$$f(y) = max(0, y). \tag{5}$$



(a) Max pooling and subsampling over a 2 x 2 area



(b) Overall structure of the network

Figure 21: Network structure as described in Wohlhart et al. [36]

The following two layers are fully connected. The first employs again a ReLU activation, whereas the last layer has the resulting descriptor as output.

To train the network two training sets are taken for every object. One of it is synthetically generated using three dimensional CAD models of the objects, the other is manually generated using a RGB-D device.
Additionally, to these, various variants of the synthetic training set are created, by adding background noise around the objects. Using these the network should learn to ignore the background behind the objects.

It has shown that the convolutional neuronal network approach described in [36] outperforms the results of LineMod [13] and HOG [4] in terms of pose error, descriptor length and also retrieval speed.
Further it is shown in the paper, that it is possible to generalize the descriptor. The descriptor generalizes well in an experiment, where the query object is not known during the trainings phase. The resulting pose estimation returned a good recognition rate.

3 Method

To solve the problem of detection and especially the alignment of three dimensional models with an incomplete scan of an object belonging to the same class a method has to provide several properties. The method should be able to classify an object in a general way and should be invariant to scale since the object to be detected may vary in size. It should also be able to detect the object in various not necessarily common positions and orientations. Since no single method has all of these properties, this thesis tries to solve the problem by using various different approaches, which in combination allow to detect and align incomplete scans with a model. In the following the different components will be explained and in the end the derived method will be discussed.

3.1 Local Feature Analysis

The first approach takes the standard local pipeline way described earlier. The pipeline can be divided in two parts. The pre process part is done off-line, it can be seen as a set up phase for the whole approach. During this set up a database is created, which is used in the next part.
The on-line part is the part of the pipeline which runs during the recognition. The scene is processed and objects are matched against the database.

3.1.1 Off-line

The initial step in the pipeline is to create a database with three dimensional models of every object, that should be detected in the scanned scene. The database is set up using freely available CAD models from an open source model library [1]. For a better further processing they are converted to a more suitable format (Figure 22a). Since the models come from different unknown sources, their sizes can vary and there can be a huge difference to the dimensions of a real world object. Therefore, the models are resized so that they all have the same size and they are in the characteristic dimensions, which a real object would have. After that the models are converted into point clouds (Figure 22b), so that they have the same representation as the scanned scene. Finally, the normals of every point in the cloud are computed (Figure 22c), by computing the Least-Squares plane fit for the nearest neighbours of each

[1] `https://3dwarehouse.sketchup.com/`

point.

This point clouds can then be used to extract key points (Figure 23), which

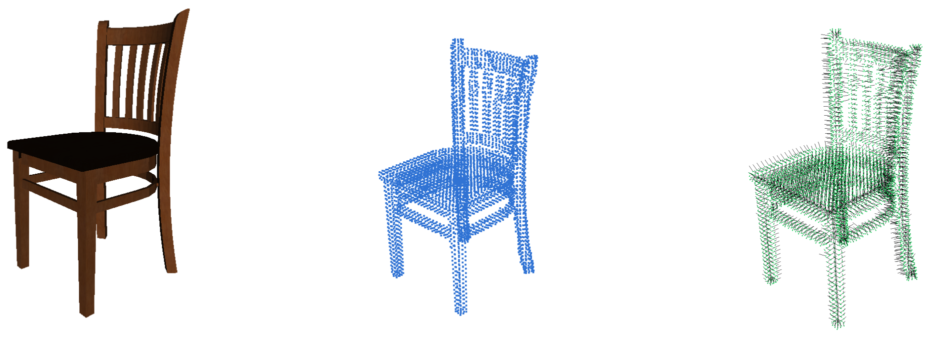

(a) 3d model (b) point cloud representation (c) point cloud with calculated normals

Figure 22: Pre process steps done on every database model

are stable, distinct and detectable using well defined criteria. The three dimensional Harris corner detector (2.1.1) is used. It is fast and especially it is adjustable. This means that the amount of key points that are found are dependent of a search radius parameter. Since the sizes of every model and searched object are the same, the search radius parameter can be chosen to be relative to the object size.

These key points are then used by the Fast Point Feature Histogram (FPFH) feature descriptor. The descriptor describes the surface around the key points in a distinct and reproducible way. The result is a feature vector (Figure 23), which can then be used to find similar surface parts.

After the pre process part the database contains the three dimensional CAD representation and their key points and the corresponding feature vectors of each model.

3.1.2 On-line

The on-line part of the pipeline starts by taking a single image scan from a scanning device, which results in a RGB image and a depth image. These images are then merged and converted into a point cloud with colour information.

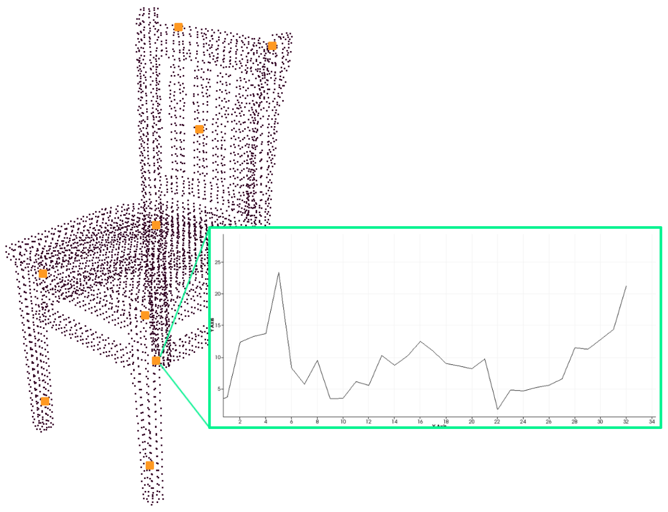


Figure 23: Key points (orange) extracted with the Harris key point detector and the FPFH histogram of one key point.

The constructed point cloud has a high amount of points (Figure 24a), which is undesired as it takes up too many resources and computation time to process this cloud. Therefore, the point cloud is sampled to a smaller point cloud, with no colour information included, as that is not needed in the further processing anyway. The sampled point cloud is much smaller (Figure 24b), but contains still enough information to do the same operation and to give the same result as the whole scan would. The sampled point cloud

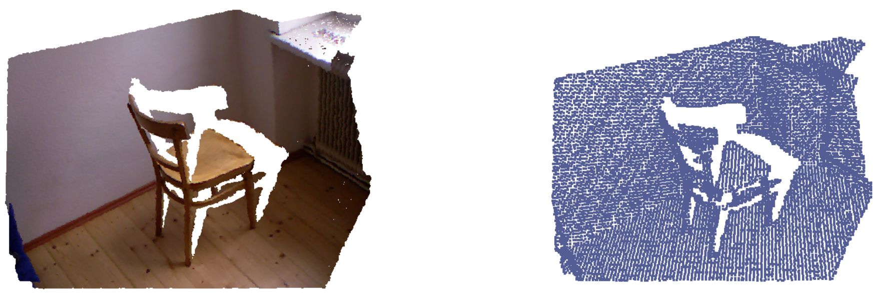

(a) Constructed point cloud (b) Sampled point cloud

Figure 24: The constructed point cloud contains many points (307200), by sampling it down the amount of points can be reduced (14291), while still preserve all the details needed to detect an object.

is then processed like the model point clouds in the pre process step. The normals are calculated and the key points and their feature descriptor are computed for the whole scene.

These feature descriptors can now be used to search corresponding features in every database model. The correspondences found are not necessarily geometrically correct and it is not guaranteed, that every feature in the scene has exactly one corresponding feature in the model (Figure 25a). Therefore,

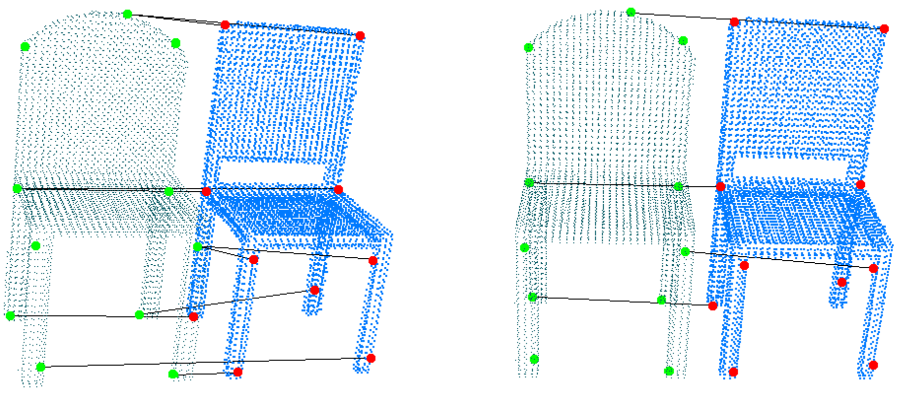
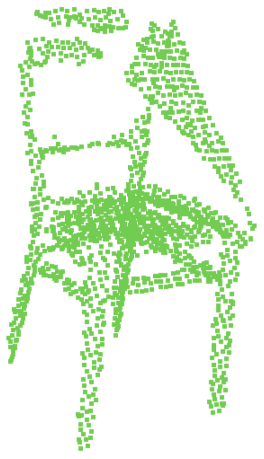

(a) all correspondeces found (b) cleaned correspondences

Figure 25: The initial found correspondences (left) can contain correspondences where a feature corresponds to multiple features of the other object. Only after removing these a clean set of correspondences (right), which propagate a similar transformation, is obtained.

the correspondences need to be cleaned, such that only correspondences with one to one relation remain (Figure 25b). Since each of these correspondences provide their own result of the relative transformation of the objects, a good consensus needs to be found.
To find these consensuses the SAC-IA method (2.1.3) is used. It provides a fast and reliable way to find a good consensus between the transformation proposed by each correspondence. This way a preliminary position and orientation of an object in the scene is found.
Due to the fact that only a few key points are used and therefore the number of correspondences can be really low, finding a good consensus, does not necessarily mean, that an object was found. To verify that the matched features are part of a sought object a further step is required.

This verification step uses the bounding box of the matched database model to obtain the points belonging to the detected object (Figure 26). A voxel


(a) Extracted points of a correctly matched object

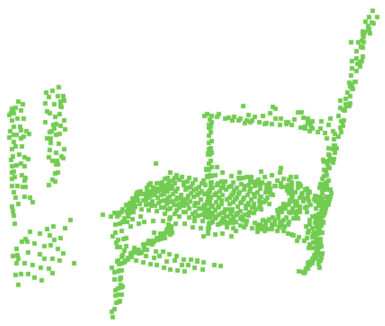
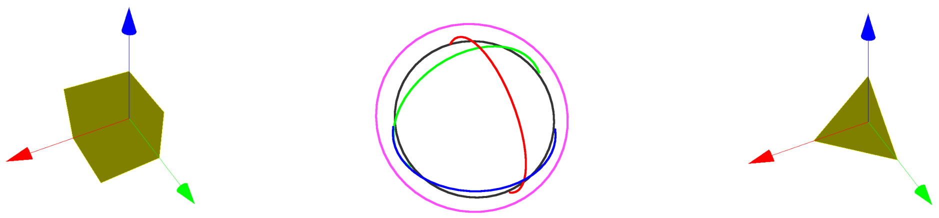

(b) Extracted points of a wrongly matched object

Figure 26: A successful alignment, does not guarantee that a correct object has been found. Only after the verification step wrong detections can be eliminated. The objects are extracted from the scene and then compared with the matched model.

grid representation of these points is created using the methods described in [17].
This voxel grid is then compared with the voxel grid of the matched model. The comparison is done by detecting the number of voxels that are present in both, the model and the scene part and the number of voxels that are in the scene but not part of the object. To get a matching error from these two values, the following threshold is defined: The number of equal voxels needs to be at least 70 percent, this has proven to be a good value. A higher threshold is not efficient, as the scan returns a 2.5D representation of the scene, so there is always a part of the scanned object which will not be present in the scan. The number of voxels which exists only in the scene should be minimized, we take therefore this number as matching error if the other threshold holds.
Although with this error metric in some cases an otherwise correctly detected object, is seen as not, it ensures that only valid objects are detected and that wrong detections are minimized if not eliminated.

If more than one object should be found, the scene can be cleaned by removing the points of the already detected objects. With the cleaned scene the procedure can be repeated until no further object is found.

To give more comfort to the user and to allow him to change the alignment any way he wants, some user tools are implemented as well.
The first tool allows the user to change the position of the aligned model (Figure 27a). It uses a "translate-gizmo", which is well known from other 3D applications. The "gizmo" can be used to freely position the model in all dimensions or it is also possible to constraint the movement to one or two specific dimensions by clicking on the appropriate axes.
The second tool allows the user to rotate the model (Figure 27b). Same as for the first tool the axis of rotation can be restrained by the user.
The third tool allows the user to change the scale of the model (Figure 27c). Although even with this tool it is possible to restrain the axis of scaling, the main application of this tool is to uniformly scale a model across all axes.



(a) translate (b) rotate (c) scale

Figure 27: Implemented "gizmos" to help the user to refine the alignment of models.

3.2 Global Feature Analysis

The next step is to use a global descriptor pipeline. The global pipeline consists of clustering, classification and alignment. Same as for the local pipeline a pre process step is introduced, which creates the database used throughout the pipeline.

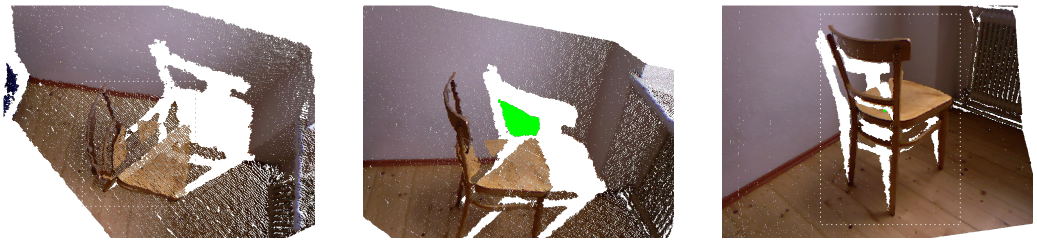
3.2.1 Pre processing

During the pre processing step the database is populated. The procedure is nearly the same as in the local pipeline. The models are downloaded, converted into a point cloud and other data which will be needed later is created (ESF descriptor, Voxel grids, ...). Unlike the models in the local pipeline, the models used in this database does not need to have the same size as for the global descriptors, the relative size of object is of no importance.

3.2.2 Clustering

As seen in 2.2.1 several methods to segment a scene into smaller parts are available.
They all use different approaches and as such they report different clusters. Since none of theses satisfies the premise to segment a scene into specific objects and therefore none of theses methods can be used to automatically extract an object from the scene, an interactive approach is taken instead. It is fairly easy for an user to detect and define an object in a scene and by using a simple interaction. Like this a good segmentation of the scene can be created, which is a really challenging task to automate. Therefore, the user is given some tools which require only simple interactions, like drawing a rectangle or picking unwanted parts of the scene.
In particular three tools are integrated into the implemented application which can be used to remove unwanted points from the scene.
The first tool is a rectangle selection tool (Figure 28a). With this tool the user can draw a rectangle around the points he wants to be retained. This tool is easy to use and provides a good way to remove big portion of points.
The second tool is a cluster selection tool (Figure 28b). It uses the region growing segmentation. After the scene segmentation the tool can be used to select different clusters of the scene and remove them. Therefore, this tool is used to refine the object and remove smaller parts of the scene.
The third tool uses the min cut based segmentation described earlier (Figure

28c). This algorithm needs two parameters as input, a foreground radius and a point which lies near the centre of the object. These parameters can be found, by letting the user draw a bounding box around the object to be extracted. Given this rectangle the foreground radius can be defined as maximum dimension of the bounding box created by the points inside. A point lying near the centre of the object, can be found by taking a point from the centre of the drawn rectangle. After the first segmentation, the user can refine it by defining points as background points. This tool can by used to semi automatically extract an object from the scene.


(a) Rectangle selection tool, every point lying inside the volume spanned by the projected rectangle will be selected.

(b) Region selection tool, by clicking on a region the whole cluster obtained by the region based segmentation, will be selected.

(c) Min cut selection tool, the selected green part are used as foreground points parameter for the min cut segmentation.

Figure 28: Selection tool implemented to help the user to extract the wanted object.

With the help of these three tools the segmentation step can be achieved fast, requiring only a little portion of user interaction.

3.2.3 Classification

Having the extracted object, a database model which matches the object, needs to be found.
For this the Ensemble of Shape Functions (ESF) global descriptor (2.2.3) is used. The ESF takes as input a point cloud and returns a 640 element feature vector. This vector can then be used to compare two objects.
A special property of the ESF is that not only the same object returns the same vector, but also objects which are similar or belong to the same class, return a vector which differs only marginal. Since this thesis tries to not only detect objects which have a model representation in the database, but also

unknown objects of a class, this property is of good use. This way the ESF can be used to classify an object.
To keep the classification as general as possible the class of an object is detected by comparing the ESF vector with every model of every class in the database. The class of an object can then be found by taking the class with the least average error over all of its models.

For this thesis the accuracy of the ESF applied over the whole database is enough. As there can always be the possibility that an object is wrongly classified, the user is given the chance to correct the detected class manually after the automatic classification step.

3.2.4 Alignment

Given the now classified object the orientation of the object needs to be figured out, to allow to align the three dimensional model to the scanned object.

Voxel Grid
The first approach uses a variation of the voxel grid method proposed in [17].
The method described in the paper creates a voxel grid representation of every possible orientation of the database models. During the alignment a voxel grid representation of the scene object is created. By matching these representations the orientation of the scene object, with respect to the model can be found.
This alignment works quite good and they are able to align a variety of different models of the same class to a scan. However, they use a constraint to get these results. The Z-axis or the ground plane of the scan has to be known and therefore the search space for possible orientations is only the rotation around the Z or Upper axis. The use of this constraint is comprehensible, as the search space without this constraint grows a lot in comparison of 36 possible orientations using the 10 degree step from the paper. They also use a voxel grid with 9x9x9 voxel, which take up a lot of memory space.

Further experimentations during this thesis have shown that the voxel grid can be reduced to a simple two dimensional grid, by projecting each point down to the fixed ground plane. The results using this grid are the same, in some cases even better, as with the three dimensional counterpart, however the needed memory space is reduced, in fact it was possible to reduce the grid size to 6x6 without any visible loss in the alignment precision.

Since this approach needs to know the Upper axis to successfully find the orientation of the object, further user interaction is needed. Similar to the previous user interactions, also this interaction is designed to be as minimal as possible. To get the Upper axis, the simplest way is to let the user select a point which lies on the ground plane. Using the region based segmentation described in 2.2.1 the cluster which contains the point can be used to find the ground plane using the random sample consensus described in 2.1.3.
This way a good alignment result can be found if the ground plane is previously known.

CNN

To remove the Upper axis constraint and to reduce the memory usage an additional method using a convolutional neuronal network is implemented. This method follows the approach of Wohlhart et al. [36]. The goal of this paper is to train a neuronal network, which is able to find the model and the pose of an object, by analysing the colour and depth images of a scan.
Even though Wohlhart et al. used the neuronal network to both detect and align models, in our case the network will only be used to align a model with a scanned object. The main reasons behind this decision are of practical nature. Since the detection and matching of a model to a scanned object was already successfully tested and implemented with the help of the ESF descriptor, only the alignment part of the neuronal network is needed. Another reason lies in the fact that the classification of objects using the neuronal network adds more complexity, speaking of computation time and storage needed. The main reason regards the flexibility and extensibility of the whole system. Even though a classification using the neuronal network would be possible, it is not practical and feasible. To classify an object the neuronal network needs to be trained with the data of all classes, which is both time and storage consuming. Further the whole system needs to be trained together, meaning that if a class is added to the system the whole network needs to be retrained. By using the neuronal network only for alignment purposes, each class can be trained individually and therefore it is simple to add a class without touching the other classes.
To train the neuronal network synthetic images of scans are needed. Like in the paper the scans were generated using the 3D graphics software Blender[1].

[1]`https://www.blender.org/`

For the scans 301 template position are created. They all lie on a half dome over the object created using a regular icosahedron and subdividing it two times. Scans of the same model from additional positions are made by subdividing one more time. These additional 1241 scans will be used as training set. From each position three images are saved, the colour information from the frame buffer, the depth information from the depth buffer and a binary mask image, which displays only the model surface.
Since the scans in the global pipeline are already cleaned, meaning that only the extracted object and no disturbing background is in it, also the synthetically generated images are created without background. Also, contrary to the implementation in the paper no further background noise is added.

The binary mask image is used, to create the 64x64 pixel images, which are needed as input for the neuronal network. These images are created, by calculating the minimum bounding rectangle of the mask image. The bounding rectangle is then made square, by extending the smaller dimension with the difference to the higher dimension. Additionally, a fixed amount is added to both dimension, so that the objects are not touching the border of the images. The resulting image part is then scaled to have the extent of 64x64 pixel.

In the paper not only the pose, but also the model corresponding to the scanned object is detected. Because in this thesis the goal is to detect multiple objects of the same class and not only to match one object to one model, the to way train the network has to be changed. Generally speaking the goal is to train one network for each class, which is then able to detect the pose of an arbitrary object of this class with respect to one template model.

To achieve this, several adjustments has to be made. Instead of having multiple models of different classes, multiple models of the same class where used to train the neuronal network. The scans are then divided into template, training and test sets.
The template set contains the 301 template poses of one single model. This is seen as template model of the whole class. The matching will later be performed on this model.
The training set contains the remaining 1241 images of this first model and a random fraction of images of a second model. The remaining images are used to test the neuronal network after it has been trained.

Before the training a set of triplets is created, which is then used during the training phase. In the original implementation the triplets where se-

lected by fulfilling two conditions. The first is that if all samples are from the same object, the poses of two samples must be more similar than the pose of the other sample. The second is that two samples have to be from the same, whereas the other is from another object.
Since in this implementation only one object (or one class) is available, the second conditions is removed and the missing samples are created using the first condition.

The trained network is finally used to create a descriptor of every pose of the template model. This descriptor can then be used to match the descriptor of a query image to the right template pose. To use this neuronal network the scanned cloud, which has been cleaned and classified during the previous steps, needs to be converted into the colour, depth and mask images needed by the neuronal network. The conversion is easy if the cloud is held organized, meaning that the pixel position, where each point is constructed, is known. Special attention needs to be paid to the depth images, as these can be varying and the distances can differ a lot with the ones of the synthetically generated depth images. Therefore, every depth image, synthetic or real is normalized, such that the nearest point corresponds to 0 and the farthest point to 1.

With the descriptor of the real world images obtained with the neuronal network, a matching template pose can be found by comparing it (using the euclidean distance) with the template descriptors. The found pose represents now the pose, more precisely the orientation, of the scanning device during the scan. The orientation of the model is therefore the inverse of the found pose.

While this method can return a good alignment, in some cases where the scanning device is not held in an upright position, this method fails, since the training data did not contain images where the camera roll angle is changed. To solve this problem a further step is needed. Since the network can only work with images which where taken in an upright camera position, the input data of the query object is transformed by rotating the image consecutively and therefore receiving 180 different images. Theses images are then fed into the neuronal network. This results in 180 poses, which are now used to find the right pose of the object. To get which poses fits the object best, the model is transformed using the pose and then a voxel grid representation as described previously is created. The right pose can then be found by comparing the object representation with the representation of the transformed model.

Since the query objects are matched against the nearest template pose, this does not mean that the found pose is exactly the same. To refine the pose and bring it a little closer to the correct pose of the scanned object a further step is introduced. The alignment is refined by rotating the aligned model in the range of five degrees around each axis and comparing the resulting voxel grid representation with the voxel grid representation of the scanned object. This way the alignment can be vastly improved.

The final orientation can then be used to orient the model the same way as the scanned object. By computing the bounding box of both, the model and the object, the model can also be positioned to be right in the centre of the scanned object and to have the similar dimension. In the end two methods to find the alignment of an object are implemented.

4 Results

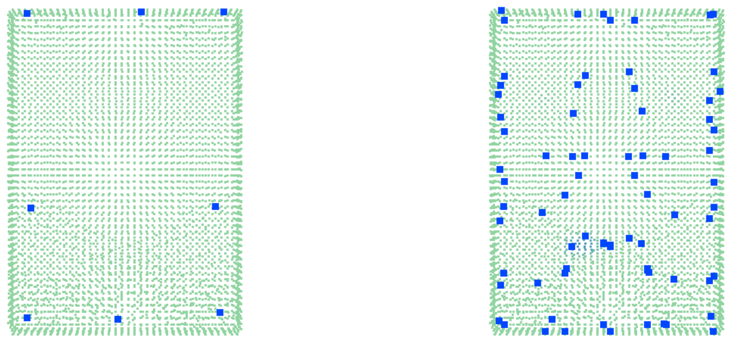
In this chapter the results of the various methods for detecting and aligning objects will be discussed. The first part talks about the results gathered with a local pipeline, while the second part discusses the different variants of the global pipeline approach and their results.

4.1 Local Pipeline

The local recognition pipeline uses local feature descriptors to detect objects. In a pre-processing step the implementation creates a database with the models and their local descriptors. Later these local descriptors are used to find and align the models in a scanned scene.

4.1.1 Key-point Detectors

During the implementation two state-of-the-art key-point detectors were compared. The Intrinsic Shape Signatures (ISS) detector by Yu Zhong et al. [39] and the three dimensional specialization of the Harris corners detector [28] were tested. Both take a point cloud with point normals as input and return stable feature points.



(a) Unmodified model (b) Scaled model

Figure 29: Key-points found by the Harris detector on the same model with different scales. In both cases the same search radius is used.

While the ISS takes no further parameter, the Harris key point detector needs an additional search radius parameter. This parameter controls how much of the surrounding points will be taken into consideration to determine the score of the key-point. If the parameter is high the key point amount will drop, since only the most distinctive point in a big neighbourhood will be selected as key point. However, if it is too low, a lot points with little details will be reported as key points (Figure 29).

Contrary the number of key-points reported by the ISS will always stay the same, even if the model gets scaled (Figure 30).

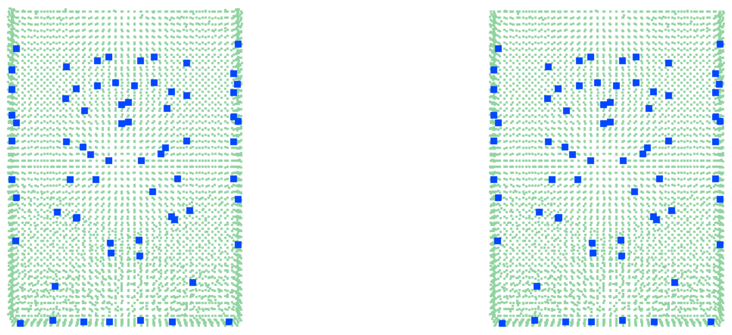

(a) Unmodified model (b) Scaled model

Figure 30: Key-points found by the ISS detectors. The detector is invariant to scale, therefore the same amount of key points is detected in the scaled and the unmodified model.

While in most applications such a behaviour is unwanted, in this case an adjustable key point descriptor is needed. Since the ISS is not controllable, it will always return a high amount of key points. We rather want a small set of key points, which describe the most distinctive parts of the object, due to the fact that we want to match objects which are highly dissimilar. This is why in our implementation the Harris key point detector is used. With this detector we can decide how many and which type of key points we will get.

This will help us in the further steps, as we only want to match the most distinctive parts of two rather different objects.
Therefore, even tough the ISS on the first glance provides better results, the Harris key-point detector will be used in the further computations.

4.1.2 Feature Point Descriptors

Same as for the key-points two state-of-the-art feature descriptors were tested. Those are the Fast Point Feature Histogram (FPFH) and the Signature of Histograms of Orientations (SHOT). They were tested using the key-points detected with the Harris key point detector. Both descriptors take a search radius parameter, which defines how many of the surrounding points will be taken into consideration to create the feature descriptor.

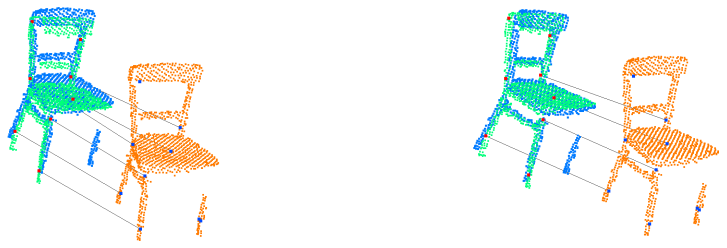

(a) FPFH (b) SHOT

Figure 31: Alignment of two incomplete scans (orange and green) of the same chair. The alignment of the orange scan with the green scan is shown in blue.

Both descriptors performed equally well during all the tests. As can be seen in Figure 31 both, FPFH and SHOT, are able to align two scans from different directions of the same objects. That is not surprising, as they both where made for such a use case. But as further tests have shown (Figure 32) they both are also able to give a good initial alignment of two quite dissimilar models of the same class.

Since the results of both descriptors where nearly the same, the FPFH was picked. The reason is that it has a better integration with the Point Cloud Library[1] which is used in the implementation of the practical part of this thesis.

[1]`http://www.pointclouds.org`

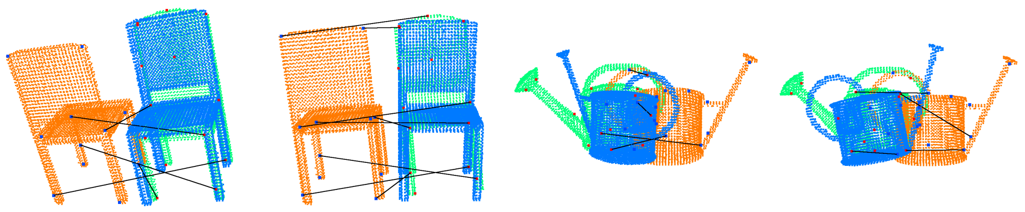


Figure 32: Alignment (blue) of two similar models (orange and green). Left FPFH and right SHOT respectively. The alignment error of the two methods between the two chairs is nearly the same. Whereas the error between the watering cans is larger with both methods, but the FPFH aligns it with a smaller error.

4.1.3 Alignment

Principle Component Analysis

A first approach to align a scan with a model was to take the Principle Component Analysis (PCA) [22]. The PCA computes the principal component of a point cloud. As result the eigen vectors of the point cloud is returned. In the case of a complete model the PCA will return the main components of an object, which can be used to align two objects. In the given case, however the scan is an incomplete representation of the real object. Therefore, the analysis of it will return a wrong result with respect to the real complete object (Figure 33). So it is not possible to retain a good alignment using the PCA method.

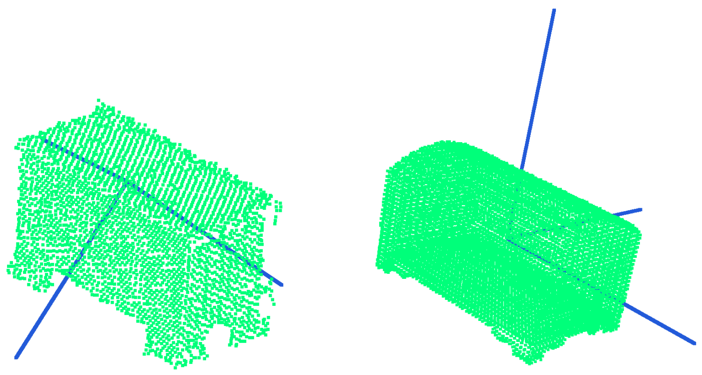


Figure 33: Results of the PCA applied on a partial scan left and a complete model right. Note that the computed principal component is exactly opposed to the scan direction.

Sample Consensus Initial Alignment
To get a correct alignment and especially to be robust against missing object parts, the Sample Consensus Initial Alignment (SAC-IA) introduced by Rusu et al. [26] was tested. It tries to align the matched features not only by optimizing the distance between two correspondences, but also by taking into account the ratio of overlapping between the two point clouds. Because of this SAC-IA is most often used to register two different and incomplete scans of the same object. During the tests, it was also possible to align even two rather dissimilar objects in a satisfying way. Therefore, SAC-IA provides a fast and robust way to align two diverse point clouds.
As shown in Figure 34 it is possible to align a scan and a model which are quite dissimilar acceptable.

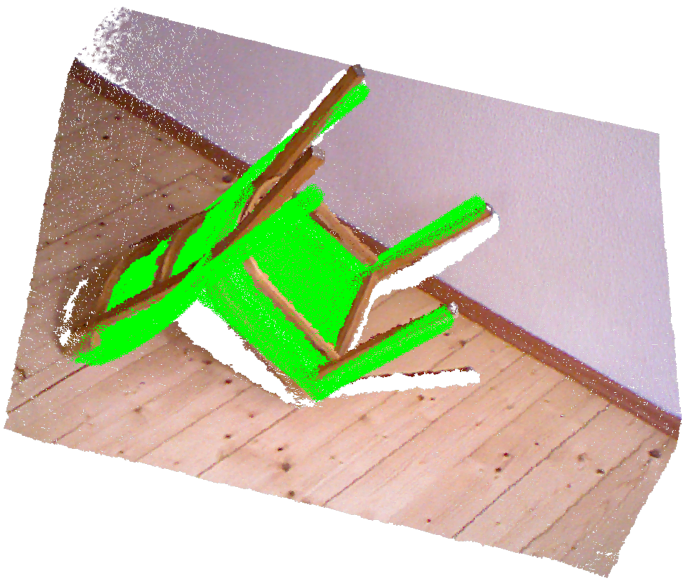


Figure 34: Scan and model, detected and aligned using the local recognition pipeline.

4.1.4 Conclusion

With the described methods, Harris key point detector, fast point feature histogram and the sample consensus initial alignment, the local pipeline approach has given some good results. It has been proven to work on scans, where a relatively similar model is available in the database. It is possible to match and align various objects as can be seen in Figure 35.

Despite this, the approach is unsuitable for a general object detection and matching, where the appearance of models are really different in comparison with the models known in the database. It also depends a lot on how much of the scanned object is visible in the scan and how much can be used to calculate the feature descriptors. Another weak point of this approach is that, due to the fact that radius parameter are needed for the key-point detection and also the feature description, the size of the scanned object has to be known, at least approximately, during the database creation. Therefore, it is also not possible to detect objects with different sizes than the ones present in the database.
For an application where all objects in the scene, are previously known and available in a CAD representation, this approach should bring good results and a high matching quote.

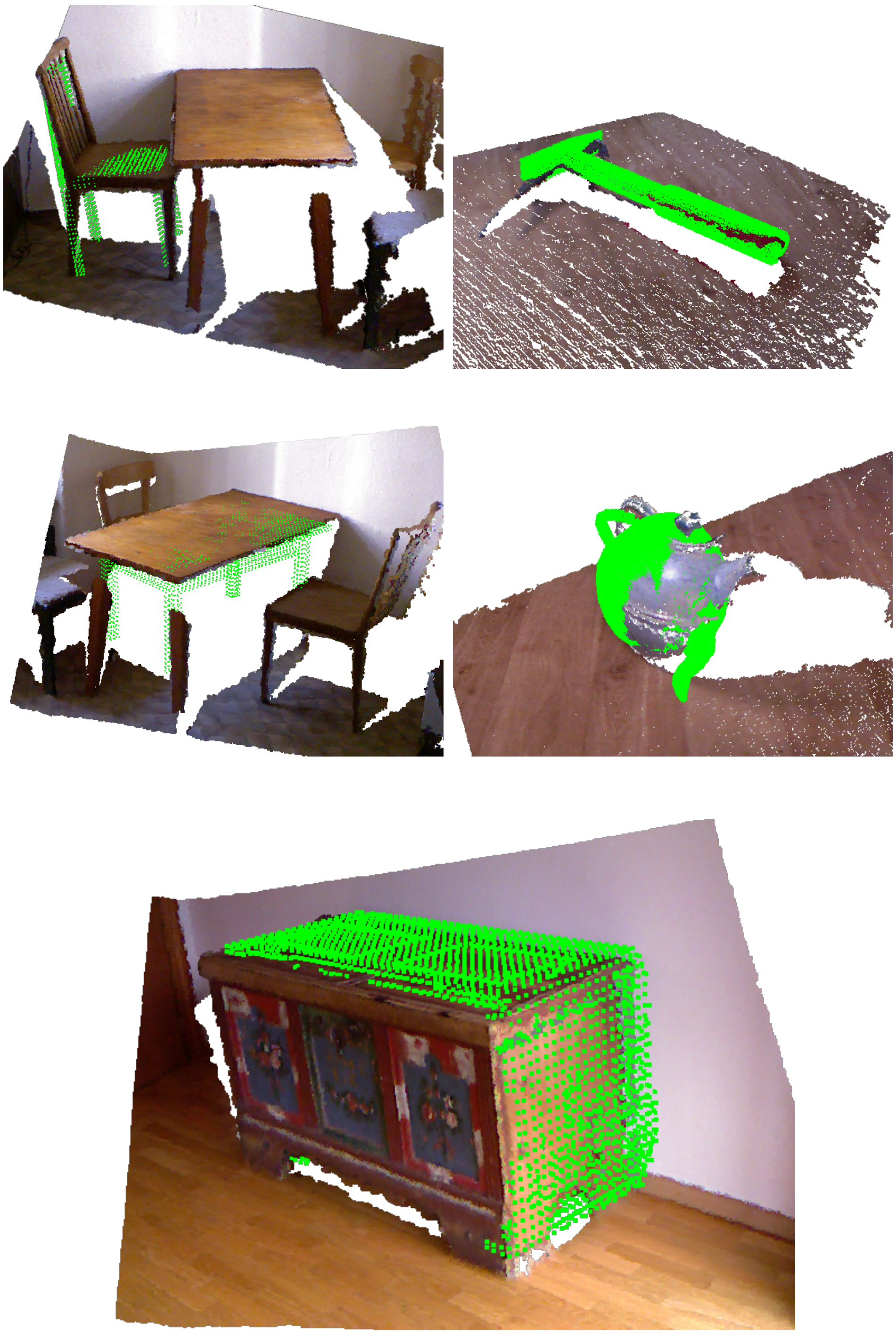


Figure 35: Various scenes with the aligned models (green). Due to the spherical body the teapot is not aligned with the right pose, whereas the chest is aligned 90 degrees rotated, as there is too little information about the correct pose in the feature descriptors.

4.2 Global Pipeline

This section covers the results of the global pipeline approach. Objects are first extracted from the scene, then they are matched against the models in the database. Finally, the matched objects are aligned accordingly.

4.2.1 Clustering

To extract an object from the scan and liberate it from the clutter three segmentation algorithms were tested. These are Euclidian, Region growing and Min Cut based segmentation. They all use different measures to decide if a point lies in the same cluster.
The Euclidian segmentation uses the flood fill algorithm. First a random point is chosen, then the distance of each neighbouring point is calculated. If it is less than a threshold, then the point is added to the cluster. If no new point can be added, a new cluster starting from a new random point is created. As can be seen in Figure 36 due to the fact that the algorithm only takes the distance between two points, it can not detect if a point belongs to another object and therefore the clusters returned by the Euclidian segmentation can span multiple objects.

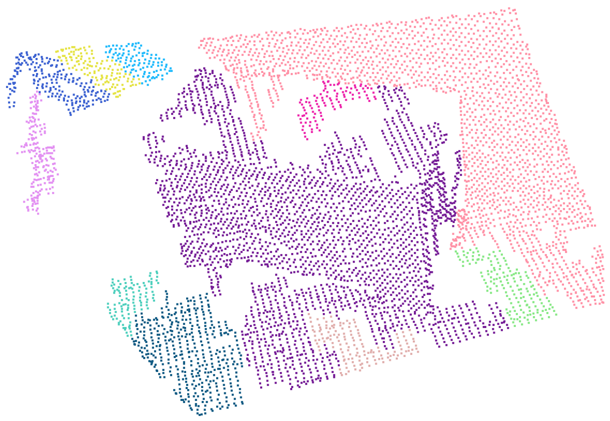
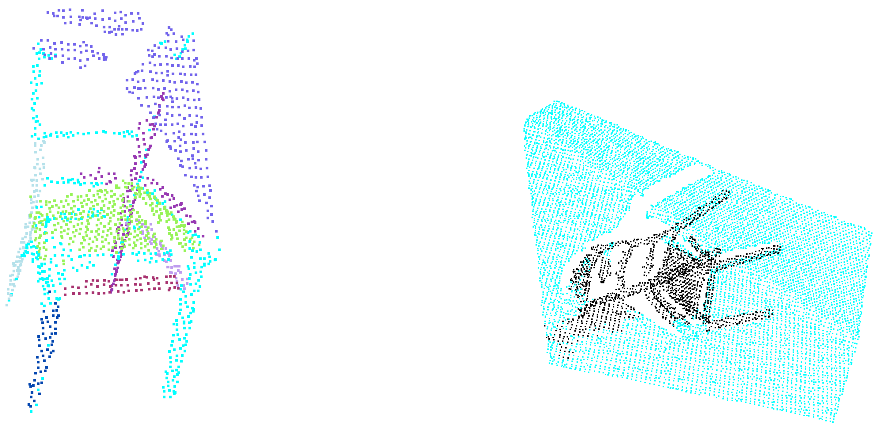


Figure 36: The Euclidian segmentation produces clusters which can span over multiple objects. Note how some parts of the scan are not clustered, as the distance threshold is too small.

The Region growing segmentation uses the same algorithm, but instead of only looking at the distance of the points, also the curvature of the points is taken into account. By using the curvature as measurement, this algorithm is able to return clusters with a cleaner segmentation, which respects object borders. However, it also segments the object itself, if the object is made of surfaces with different orientations and hard edges.



(a) The Region growing segmentation clusters the scene nicely into basic elements, the object is made of.

(b) The Min Cut based segmentation fails to extract the object, since in the scan the transition between object and background is seamless.

Figure 37: Region growing and Min Cut based segmentation.

The Min Cut based segmentation uses an algorithm introduced in [10]. This algorithm takes the centre point of an object and its radius as input and returns a binary set of foreground and background points. Where foreground points are likely to belong to the object and background points are not part of the object. During some tests it has shown, that a clean extraction of the object from a scan is not possible and that the returned foreground points will always return some clutter, which was wrongly classified.

The figures 36 and 37 visualize the results of each individual algorithm. As seen the region growing segmentation provides the best segmentation of the scene into separate patches. But since most objects are clustered into multiple patches an extraction of one object using only this method is not possible. In fact clustering of a scene is an active research area and a general solution does not exist yet. Therefore, the extraction of the object is performed with the help of the user. Giving him the right tools and a pre clustered scene, there is little interaction needed to define the objects bounding box and to remove all the clutter around the object (Figure 38).

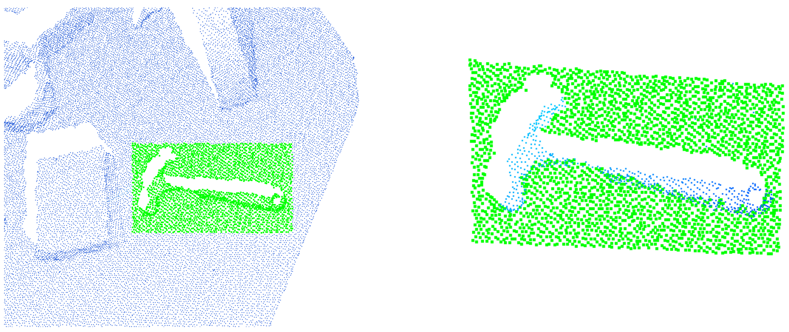


Figure 38: User helped segmentation: first the user draws a rectangle around the object in the scene, then the user can refine the selected rectangle, by removing unwanted clusters.

4.2.2 Classification

For the classification part the Ensemble of Shape Functions (ESF) descriptor is used. It is fast to compute and describes an object in a 640 dimensional vector. The good thing about this descriptor is that it does not describe the object precisely, but uses a collection of shape functions to create a good approximation of the shape. It is therefore perfect to compare two objects.

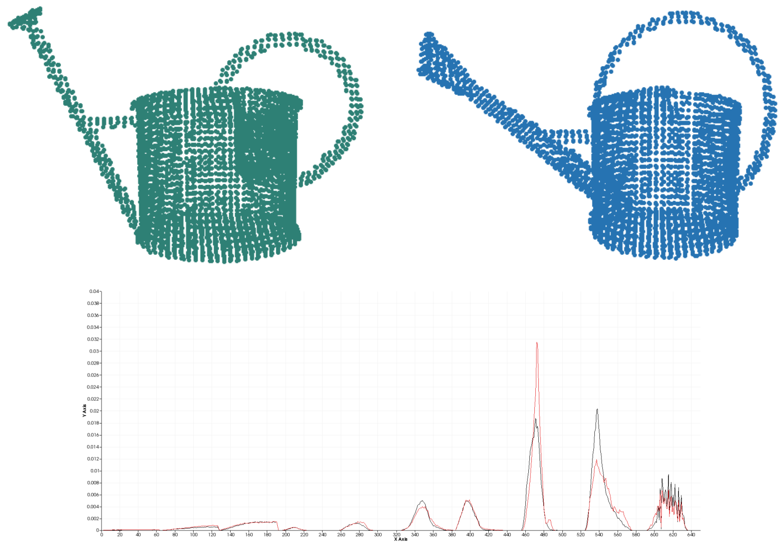


Figure 39: Two different watering cans and their respectively ESF descriptors. The difference between the two descriptors is minimal.

As seen in Figure 39, two objects of the same class return a similar descriptor. By comparing the descriptors it is easy to detect to which class an object belongs.
In some cases however, especially if object classes contain similar shapes the ESF returns the wrong match (Figure 40).

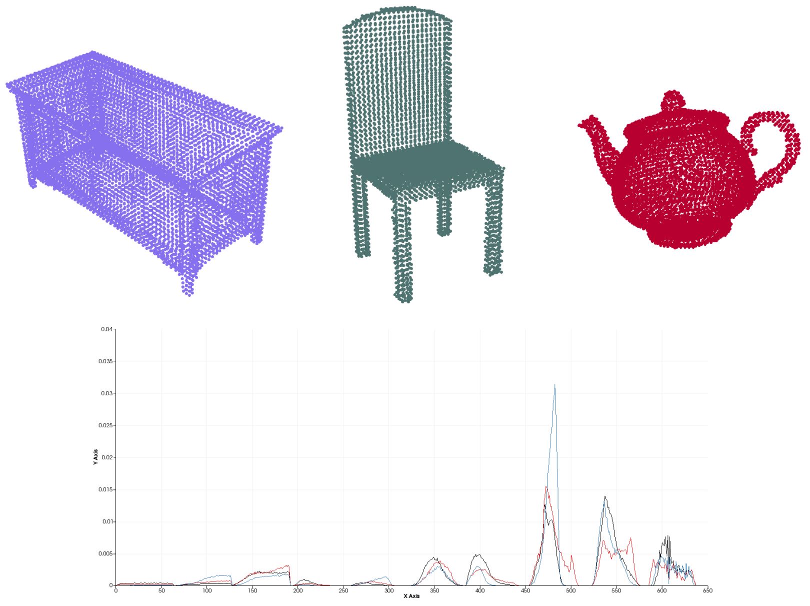
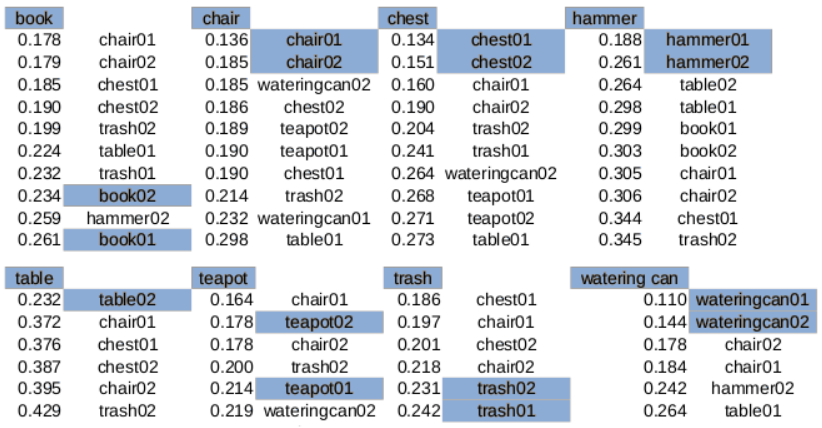


Figure 40: Some objects can have similar elements and therefore some shape functions can be equal. This leads to a smaller overall difference and a wrong classification.

The table in Figure 41 shows our test set and the query results. Most of the time the ESF delivers the right class, whereas some objects return the wrong class. These cases can be dealt with, by comparing the query with more objects of the same class and taking the class with the best average results over all database entries.

Because there is always the possibility that an object will be misclassified, the user is given the chance to change the detected class into a more appropriate class after the classification step.



book		chair		chest		hammer	
0.178	chair01	0.136	chair01	0.134	chest01	0.188	hammer01
0.179	chair02	0.185	chair02	0.151	chest02	0.261	hammer02
0.185	chest01	0.185	wateringcan02	0.160	chair01	0.264	table02
0.190	chest02	0.186	chest02	0.190	chair02	0.298	table01
0.199	trash02	0.189	teapot02	0.204	trash02	0.299	book01
0.224	table01	0.190	teapot01	0.241	trash01	0.303	book02
0.232	trash01	0.190	chest01	0.264	wateringcan02	0.305	chair01
0.234	book02	0.214	trash02	0.268	teapot01	0.306	chair02
0.259	hammer02	0.232	wateringcan01	0.271	teapot02	0.344	chest01
0.261	book01	0.298	table01	0.273	table01	0.345	trash02

table		teapot		trash		watering can	
0.232	table02	0.164	chair01	0.186	chest01	0.110	wateringcan01
0.372	chair01	0.178	teapot02	0.197	chair01	0.144	wateringcan02
0.376	chest01	0.178	chair02	0.201	chest02	0.178	chair02
0.387	chest02	0.200	trash02	0.218	chair02	0.184	chair01
0.395	chair02	0.214	teapot01	0.231	trash02	0.242	hammer02
0.429	trash02	0.219	wateringcan02	0.242	trash01	0.264	table01

Figure 41: Retrieval results using the ESF descriptor on the test set. Most of the time the right result is returned, only on some specific objects, the returned results are incorrect.

4.2.3 Alignment

Voxel Grid

To get an alignment which is robust against variations a method described by Kim et al. [17] and some derivative methods were tested. In this publication a voxel grid representation of the point clouds (Figure 42a) is used to align two quite dissimilar objects. More precisely, a voxel grid representation of every possible orientation of the models in the database is made. During the alignment step, the voxel grid representation of the scan is then compared to all available voxel grids in the database, returning the orientation of the scan with respect to the model in the database.

While this method provides good results, it needs a lot of memory as for every orientation a voxel grid has to be stored in database.

Voxel Image

To improve this a similar approach was taken. Instead of taking a three dimensional voxel grid, only a two dimensional image representation was used (Figure 42b). The image representation was made by projecting each point of the point cloud down to the ground plane. The projected points are then rasterized into an image where each entry contains the normalized amount of points that are lying in the pixel.

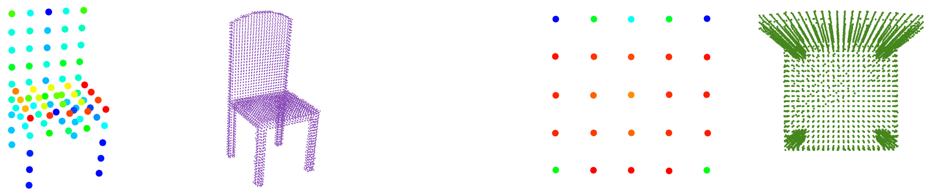

(a) 3D voxel grid (b) 2D image

Figure 42: Point cloud of a chair model with the voxel grid and image representation. The colours depict the various point densities in the voxels.

This improves the memory used but also the alignment itself (Figure 43).

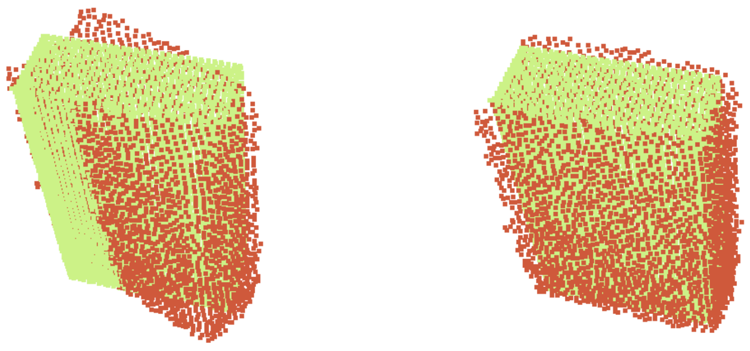

(a) 3D voxel grid (b) 2D image

Figure 43: Alignment of a model (green) to a scan (orange) using the voxel grid and image representation with a fixed upper axis.

The biggest drawback of both methods is that they need a fixed upper axis to align the objects. With the help of a little user interaction it would be possible to define a proper upper axis, but the user interaction should be minimized. Therefore, the next step was to get rid of this constraint and allow the alignment of arbitrary objects.

3D Voxel Grid

Three methods where tested. The first was to try to use the voxel grid without constraints to align two models. Since now the voxel grid needs to be made for every possible orientation, the memory usage is really high. Also, the alignment is getting worse compared to the previous case, as there are more degrees of freedom (Figure 44).

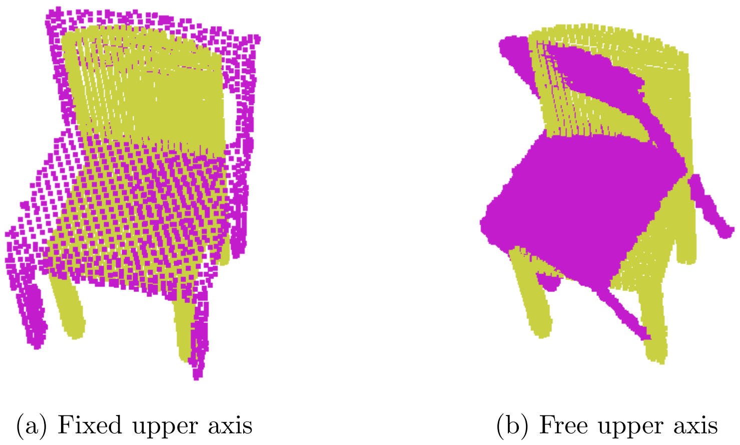
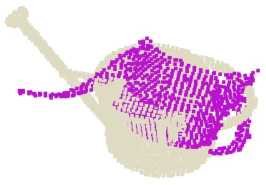

(a) Fixed upper axis (b) Free upper axis

Figure 44: Alignment of a model (green) to a scan (purple) using the voxel grid representation with fixed upper axis and without.

Depth Image
The next step is to use the depth images of an object and compare it with every possible depth image of another object. The resolution of the images can be really low, more precisely it is better if they are not too high, because we want to match different objects. By taking low resolution images, the alignment gets better because it gets not disturbed by small differences between the objects. The resulting alignment, as seen in Figure 45 is a bit better as the previous used voxel method. However, it is still not good enough than the methods with fixed upper axis.


(a) 3D voxel grid

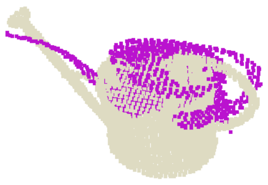
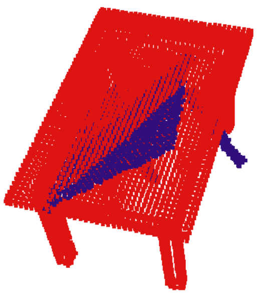

(b) depth image

Figure 45: Alignment of a model (grey) to a scan (purple) using the voxel grid and the depth image representation.

3 Shape Images

To reduce the alignment error further and to somewhat stabilize the results, another method using images, taken from front, side and top respectively, was tested. The images used are binary images, meaning that a pixel has the value of one, if it there are any projected points lying inside the boundaries of the pixel or zero if no point lies inside. This significantly improves the memory usage regarding the voxel grid and also the depth image approach. Even tough the alignment improves a little too, the alignment error is still too high, to be acceptable for the human eye. As can be seen in Figure 46, even the improvement made with this method is too small. The alignment does look better than the other results, but it is still not good.


(a) depth image

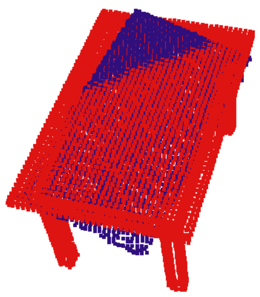

(b) 3 binary images

Figure 46: Alignment of a model (red) to a scan (blue) using the depth image and the three binary images representation.

Convolutional Neuronal Network
The comparison of different objects of the same class, or their representation, using standard measurements, like euclidean distances can be challenging. Therefore, the next step was to use a neuronal network to find the right alignment of a model to a scan of the same class.
Wohlhart et al. [36] used in their publication a neuronal network, which is able to classify and align a model with a scan. The network is trained using colour and depth image of different views of an object and their pose. By using a triplet cost function the network is trained to describe two similar poses of the same object in a similar descriptor, whereas a different pose or a pose from a different object should lead to a descriptor with a high difference. The trained network can then be used to generate descriptors from template images and query images. By comparing the descriptors the most similar template pose can be found. As training set synthetically generated images and real world images are used. In addition, background noise is added mimic different backgrounds. This way the network should learn to only consider the object and to ignore the background. Figure 47 shows some results of the described paper. The first row shows the colour images and the second row the corresponding depth images.

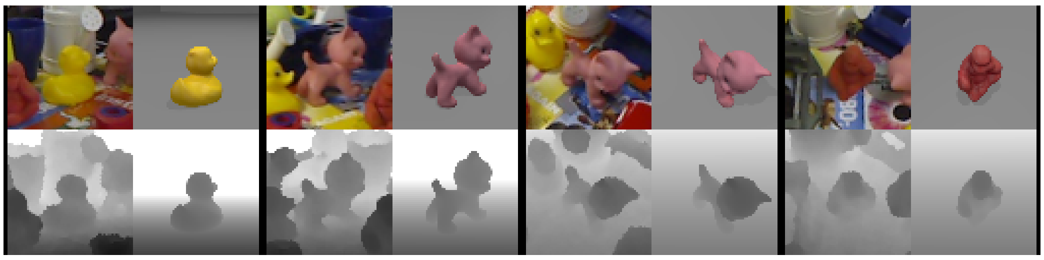


Figure 47: Retrieval results of the neuronal network presented in Wohlhart et al. [36]. The left column represents the query object, whereas the right column displays the detected pose respectively.

The method used in this thesis follows the idea of the paper. However, since the class of the object is already detected during the classification step and the extracted object is clean of any clutter, some simplification can be made. The network should only learn to generate a descriptor of the pose of one object or one class of objects.

To get to these results, the network is trained using a set of clean synthetic data generated from different objects of the same class. The difference of the objects are validated using the ESF descriptor, as seen in Figure 48. The selected objects all belong to the same class, meaning that in this special case they all can be classified as a chair. Nevertheless, each object differs in various ways from each other object, as can be visually be seen in the object models as well as in the ESF descriptor difference. This way the training objects can be selected generally as objects which are in the same class, but which ESF descriptor differ the most.

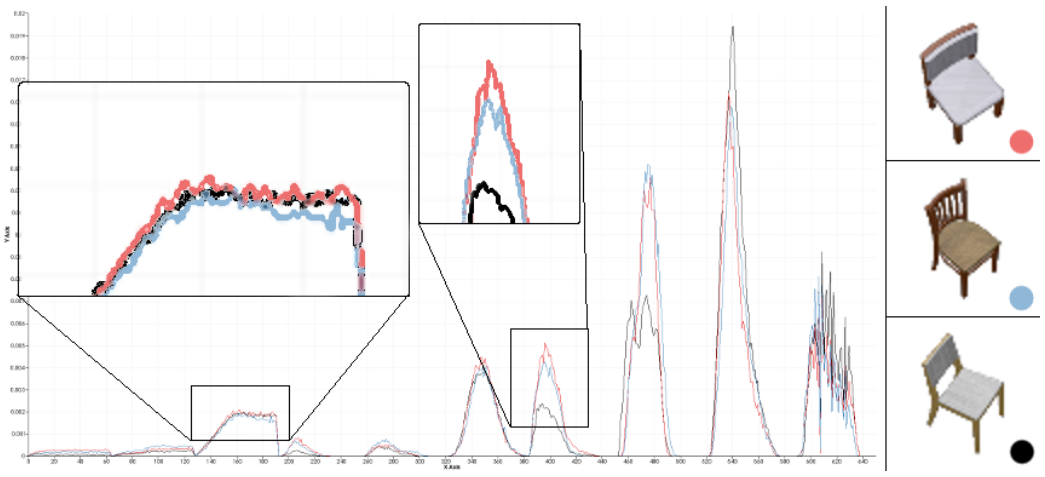
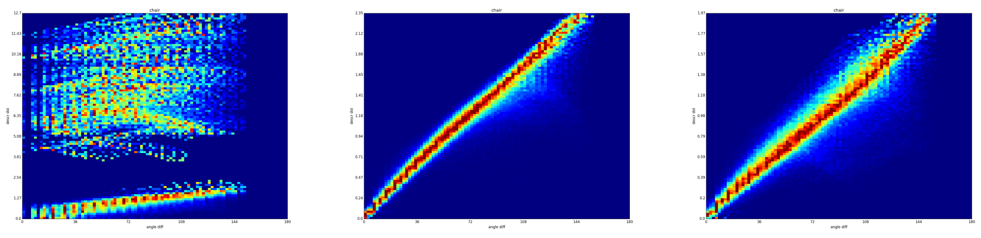


Figure 48: Objects used to train the neuronal network: Model representation and their corresponding ESF descriptors. (for more results see Appendix)

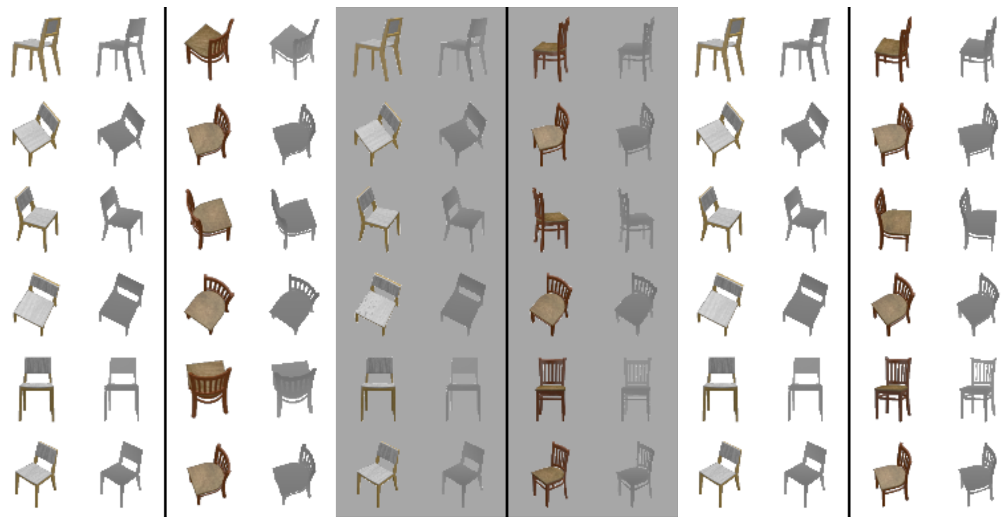
One main aspect of neuronal network is that, the more training data is available, the better the network can be trained and the better the results of the trained network will get. Of course the same is true of the convolutional neuronal network used in this implementation. The more variation of a class it is trained with, the better it will detect the pose of a query object.
In contradiction to that, during the tests it has shown, that using only two different objects as training data is enough to detect the pose of a big portion of the same class. Further it has been shown, that using an additional object as training data does not increase the overall detection of the pose. In fact, most of the time the network trained with two objects returns the same alignment as the network trained with three objects.
This can be seen in visually in Figure 50b and Figure 50c, where queries which lead to the best matching pose are shown. While in the case of two objects as training data, the best matching poses are nearly identical with the query poses, the case where three objects are used as training data, the alignment of the best matching poses are sometimes (e.g. in the fifth row) slightly of. Also in Figure 49b and Figure 49c, where the alignment error is plotted, showing the descriptor distance in correlation with the angle distance, the alignment

error increases in the case where two objects are used as training data. It can also be seen that in the case where two objects are used the correlation of the distance and error is nearly linearly. This is a wanted result, because if the distance between the angle of two poses is increase, also the distance between the descriptors should increase.



(a) One object (b) Two objects (c) Three objects

Figure 49: Correlation between the descriptor distance (Y-axis) and the similarity of the pose (X-axis) displayed as plot.



(a) One object (b) Two objects (c) Three objects

Figure 50: The best matching poses (right) using some test data (left). The network trained with two objects gives the best results, followed by the network trained with three objects. To train the network with only one object increases the alignment error dramatically.

The tests also showed, that using two objects is also the minimum to get a good pose estimation. If only one object is used to train the neuronal network, the resulting detected poses are visibly worse (Figure 49a and Figure 50a) than the results of the network trained with two objects.

In the end the trained network can output a descriptor of every pose of a template object, which can be used to match the scan data.
The following matches were made using a synthetically generated test set as well as a set of real world scans of different objects. Both show the query colour and depth images in the left column and the matched template images on the right column. In Figure 51 one finds that the network was able to exactly match all synthetic query images with the correct template images.

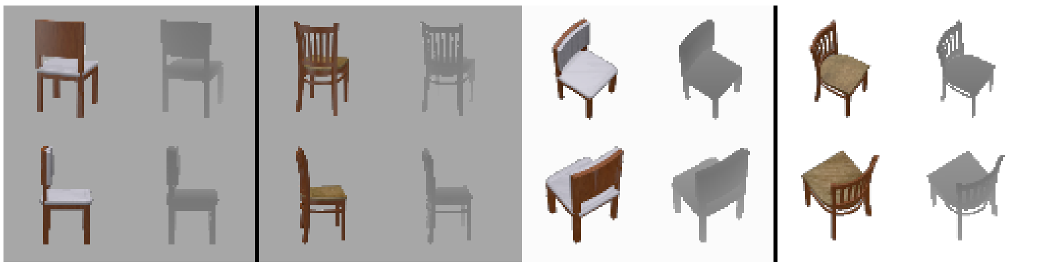


Figure 51: Retrieval results using the trained network on a synthetically generated test dataset. The first columns show the query pose and the second columns show the best matching pose of the template object respectively. (for more results see Appendix)

Figure 52 shows that the neuronal network provides good to very good results even on real world scan data. In some cases the alignment is a bit off (e.g. fourth row first and second column), which could come due to the fact that scan images are a bit rotated around the forward camera axis with respect to the template ones. To fix theses cases an additional step is introduced, where various rotated versions of the same query image are used to find the best alignment (3.2.4).
The biggest alignment error can be seen in row 2, last column. This is probably because even tough the template is rotated with respect to the query, the images still look pretty similar. As this happened only once during all the tests this error is seen as an exceptional case.
Since the template poses does not contain every possible orientation, the found pose can still be a bit of. This can be seen especially, when the aligned three dimensional representation is overlaid with the scanned object (Figure 53a). To get a better alignment, the found pose is refined by comparing the density voxel representation of the model rotated in a range of five degrees around each of the three rotational axes. This way the final pose (Figure 53b)

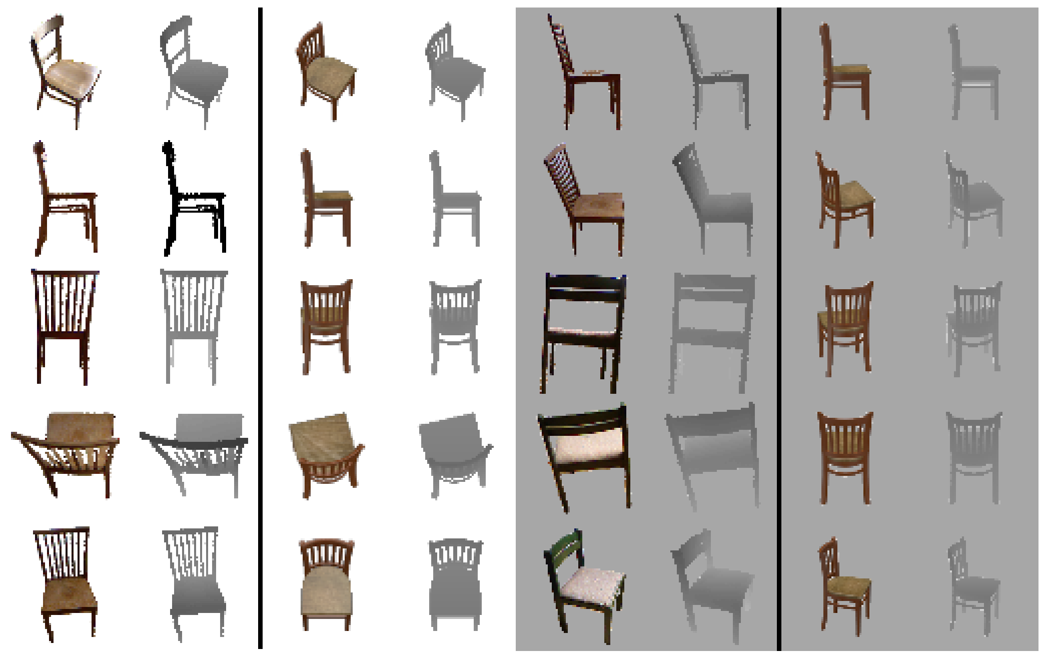
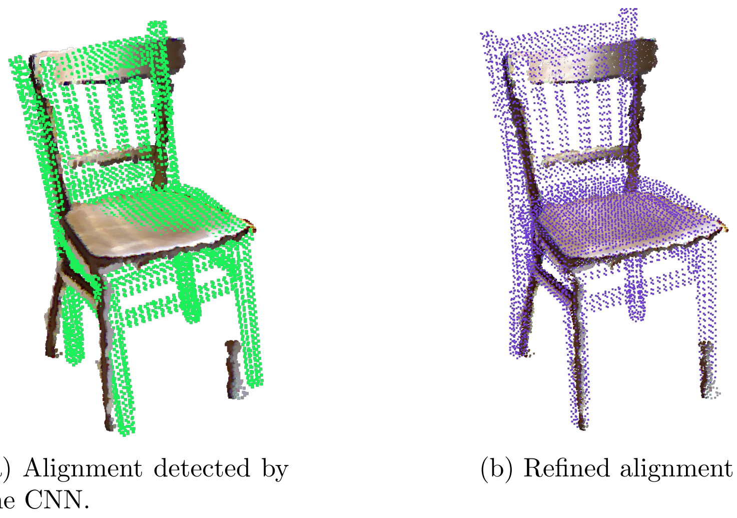


Figure 52: Retrieval results using the trained network on test data generated with a RGBD device. (for more examples see Appendix)

is a good approximation of the real orientation of the scanned object.



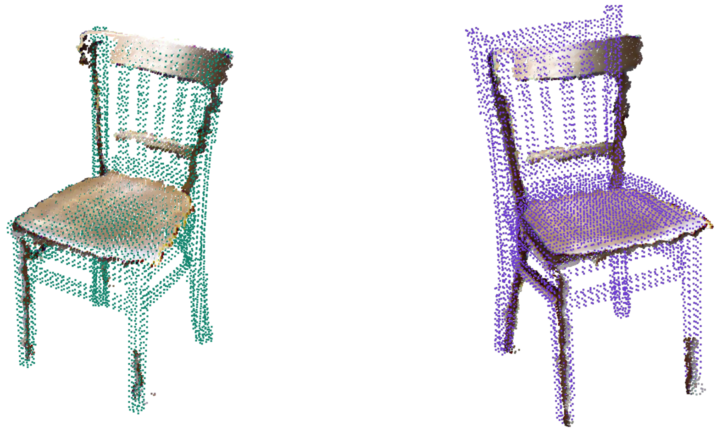
(a) Alignment detected by the CNN.

(b) Refined alignment

Figure 53: Alignment of a model to a scanned object using the neuronal network method. To get a good alignment the result of the network needs to be refined.

4.2.4 Comparison

Using the global descriptor pipeline approach it is possible to detect objects if the class of the object is in the database and therefore previously known. The described global descriptor pipeline brings two possibilities (Figure 54), one where the ground plane, or upper world axis has to be known and one with no such restrictions. Both however need a clean segmented object to work on.



(a) Alignment using the voxel image approach and a known upper axis.

(b) Alignment using the neuronal network approach.

Figure 54: Approaches using the global descriptor pipeline

The biggest advantage of the neuronal network method, is that it does not need as much storage as the voxel image one. It is enough to have the descriptors of one template object of each class in the detection step, whereas the other method needs several objects of each class to bring good results.
Both methods have their drawbacks, in the first it is the need of a fixed upper axis, which gives in return a better alignment compared to the second method. The second method has no such restriction, however it can be seen in Figure 54, that the alignment error increases therefore. Since the neuronal network is independent of any fixed axis, it is also possible to train the network such that object which are not in there upright position are found (Figure 55).

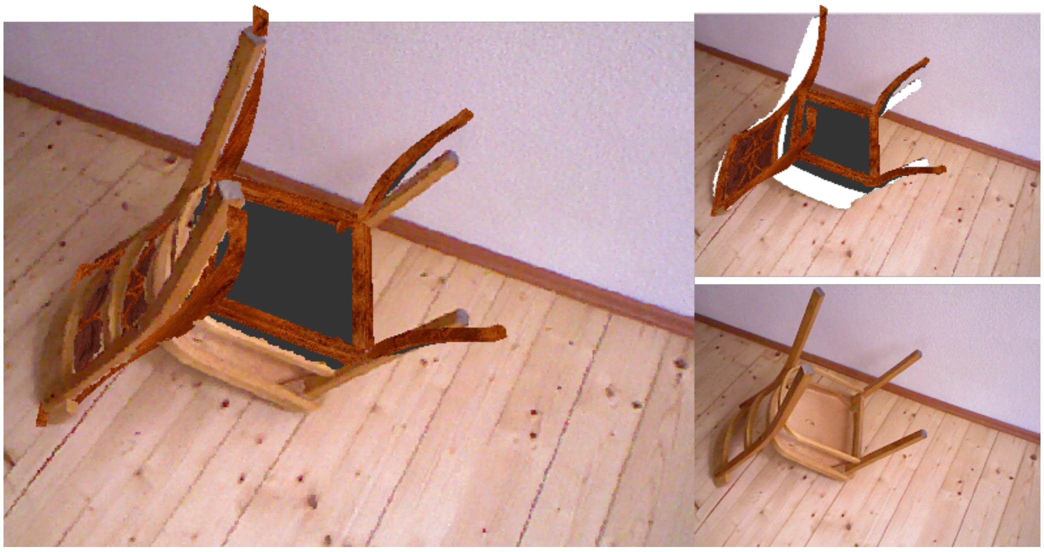


Figure 55: With the neuronal network approach it is also possible to find an alignment for objects, which are not in their usual position.

4.3 Performance

In the following the performance of each part of the system is analysed. The performance tests where made on a Intel Core i7-4500U CPU.

To scan a scene, two images (colour and depth) are created by the RGB-D device. The information of these images are then merged and a point cloud is constructed. The time needed to take the images and construct the point cloud is about 0.7 seconds, with an image resolution of 640x480 pixel. This time could be lowered to 0.4 seconds, by using a lower resolution (160x120 pixel). This reduction has no influence, since the constructed scenes are down sampled anyway (as described earlier). The only downside of using images with lower resolution is the visual appearance to the user. Therefore, in this thesis a higher resolution is used.

For the first method using the FPFH descriptor, the models need to be preprocessed by converting them to a point cloud with normals, scaling them to fit in a unit box and reducing their point count, these take about 11 seconds per model. The next step of the off-line part is to detect key points using the Harris key point detector. The time used to detect these key points depends on the complexity of the model, it lays however around one second in average. These key points are then used by the FPFH descriptor. The FPFH descriptor needs around 2 second to describe every key point, again the time depends on the complexity of the model. During the detection step, the key points of the scene needs to be detected and described (this takes about 3 seconds). Next the features need to be compared and matched against each other and the found object needs to be verified. This takes again 3 seconds. After an object has been found the scene must be researched for any other known objects. Therefore, the automatic detection of objects using this method can take a very long time, since these steps needs to be repeated for every model.

FPFH	chair	book	average
preprocess model	12.64	10.44	11.54 seconds
key point detector	0.72	0.78	0.75 seconds
FPFH descriptor	2.27	2.29	2.28 seconds
Total	0.261	0.225	0.243 minutes
Complexity			(0.243 * n) minutes

ZUP	chair	book	average
preprocess model	12.64	10.45	11.545 seconds
ESF descriptor	0.14	0.18	0.16 seconds
voxel images	9.52	9.45	9.485 minutes
Total	9.73	9.63	9.68 minutes
Complexity			(9.68 * n) minutes

CNN	chair	book	average
render templates	19	20	19.5 minutes
train network	148	151	149.5 minutes
ESF descriptor	0.14	0.18	0.16 seconds
Total	167.00	171.00	169.00 minutes
Complexity			(169 * k) + (0.00267 * n) minutes

(a) Off-line performance. n denotes the total model count in the database and k denotes the class count.

FPFH	chair	book	average
scene key points	1.100	1.000	1.050 seconds
scene fpfh descriptor	2.050	1.980	2.015 seconds
match align	3.000	2.790	2.895 seconds
Total	6.150	5.770	5.960 seconds

ZUP	chair	book	average
ESF lookup	0.098	0.090	0.094 seconds
voxel image lookup	0.180	0.130	0.155 seconds
Total	0.278	0.220	0.249 seconds

CNN	chair	book	average
ESF lookup	0.098	0.090	0.094 seconds
query CNN	2.490	2.480	2.485 seconds
CNN refine	4.250	4.180	4.215 seconds
Total	6.838	6.750	6.794 seconds

(b) On-line performance.

Figure 56: Performances of the three methods. The FPFH method is clearly the fastest in the off-line step, only with a high number of models per class the CNN method could be faster. In the on-line step the method with fixed upper axis is the fastest.

Similar to the previous method, also the method with fixed upper axis needs an off-line step, where the database is deployed. During this step each model is downloaded and transformed, leading to a point cloud of the model, which fits in the unit box. This step takes about 12 seconds depending on the size and the complexity of the model. Next the ESF descriptor of the model needs to be calculated, which takes about 0.1 seconds. For this method to work a voxel image of the model of every possible rotation around the upper axis is needed. The calculation of these voxel images takes about 10 minutes. After that the model is prepared and can be used in the on-line step.

The on-line step needs some user interaction, the up axis needs to be defined and the object needs to be freed from the surroundings. Then the class of the object is detected, by computing the ESF descriptor and looking at the ESF descriptors in the database. This lookup is a fast operation (0.1 second), as it uses a kd-tree data structure. Once the class has been found the corresponding model can be aligned by creating a voxel image of the object and comparing it with the voxel image of every rotation of the model. Also, this lookup is implemented using a kd-tree so it has about the same performance (0.1 second) as the previous lookup.
Overall the time needed to build the database is dependent on the amount of models used, while the time needed to detect an object and align a model is nearly constant, since the amount of models in the database has no influence.

Contrary to the other methods the method using a convolutional neuronal network does not need to do an off-line step for every model used. For the neuronal network to work, it needs to be trained. Therefore, two objects per class need to be rendered from several viewpoints, which takes about 20 minutes per object. With these rendered images the network is trained which takes about 2.5 hours to be completed.
The on-line step needs again an object which is free from any clutter. After the object is classified using the same approach like in the previous method, a descriptor of the object using the colour and depth image is computed. This descriptor is then matched against the descriptors of the template object. This part takes about 2.5 seconds to complete. As described earlier, the detected pose gets refined by using a voxel grid, which takes about 4 seconds.
Even-tough the time needed for the off-line step of this method seems to be much higher than the other methods at the first glance, it has to be pointed out, that the time spent is on a per class basis, rather than on a per model base like the other methods. Another point is that time reported here, is the time needed to train the convolutional network on a CPU. Training the network on a GPU leads to big acceleration (about 1h trainings time). The on-line step could also be improved, by using a GPU and another framework for the network.

4.4 Conclusion

This thesis provides three approaches to detect and align an object with an arbitrary model of the same class.

The first is the local pipeline approach, which can be seen as an automatic approach. The pipeline works without any user interaction and displays the detected objects and the aligned models right away. This method works pretty good for objects, for which a relatively similar model is available in the database. However, due to the use of local features this method has no chance to detect objects, which are very different from the models in the database. Therefore, two other methods using the global pipeline approach are implemented in the application as well.

The second method requires the user to define the ground plane of the scene and therefore simultaneously the upper axis. After that, the object needs to be cleared of any clutter. This is a relatively easy task and can be done fast if the object is standing on the ground plane, since the ground plane gets removed upon user selection and now the object should be standing free without any other objects nearby. With the box selection tool the free standing object can now be selected and the classification and alignment detection using the voxel image method can be started.

The third method uses the neuronal network approach to detect the pose of an object. Like in the previous method, also this one requires that the user extracts the object he wants to be classified and aligned from the scene. Even tough the two latter methods need user interaction, they were able to detect and align more objects and especially much better than the first method using the local pipeline approach.

Equipped with these the user can now easily use the implemented application to replace objects with arbitrary models and can therefore for example use it to create his own visually impression of how his home would look with different furniture.

5 Discussion

The result of this thesis is a system which allows to virtualize a real world scene, scanned with a RGB-D device. To achieve this the system uses three different methods. Each method works differently and has its own use case, but also its own drawbacks.
The first method uses the local pipeline. It uses local descriptors to describe the surface around key points. These can be matched and corresponding features and key points can be found. When looking at these correspondences a rigid transformation from one set of the points to the corresponding point set needs to be found. This is a known problem and various solution [26][32] exists. The problem arises if the transformation can be of non rigid nature. This can happen if the size of the query object differs hugely with the size of the object to be matched, but also for the case of this thesis, where a query object needs to be matched with a model of the same class. The geometric matching of such general objects using the local pipeline is impossible, as the degree of freedom is too high with respect of the known parameters. What would be need is a lot more information about the objects, which could be used to restrict the degree of freedom. A possibility would be to know the up vector of all points or the knowledge of some special properties, e.g. symmetry, which could be used to derive some geometrical information about the object.
Another drawback of this method lies in the computational complexity and in the storage needed. Since the verification if an object was found takes place after the alignment, each model in the database is searched and aligned in the scene. If a lot of objects want to be detected the database can be huge and therefore also the automatic detection and alignment step will take up a lot of time.
However, due to the use of local feature descriptor, the orientation of the object, the scanning device or the models in the database have no influence in the resulting detection and alignment.
For the simple case where only a small amount of objects, which are nearly identical to the one available in the database, should be detected and aligned, this approach is sufficient. It has been successfully used to find and align various object.

The second and the third method use the global descriptor pipeline. Since global descriptors work on the whole object, the object needs to be free of clutter. This is a huge disadvantage regarding the previous method. The segmentation of an arbitrary scene is a difficult task, actively researched and until now no general solution has been found. Even-tough an automatic seg-

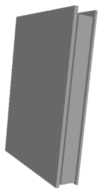
mentation of a scene would be really nice, the interactive solution used in this thesis provides a good compromise, which leads to a good segmentation with little user interaction.
To classify the object the ESF descriptor is used. It returns a good classification for most of the model and it is really fast to compute the descriptor from a point cloud. An enhancement of the usage of the ESF could be done, by using some learning algorithm to classify an object. In the current implementation the ESF of a query object is matched against the whole database. A better approach would be to create a template descriptor of each class using some specific descriptor and then match the query object against these template descriptors.
The second method uses a voxel grid to find the alignment of a model. This is efficient and accurate. There are two drawbacks using these methods. The up vector of the scanned object needs to be known. This could be defined automatically, by detecting the ground plane, however the detection is not always right and in case of a miss detection the user is left with a misaligned model. Therefore, to reliable align the model the up vector needs to be defined by the user. The other drawback is, that the voxel grid representation of each orientation of every model has to be stored in database, this increases the storage usage of the database.
The third method uses a convolutional neuronal network. Unlike the other methods, these methods finds the alignment of an object, by matching the scanning device pose against template poses. Using a neuronal network it is possible, due to the use of training data with different instances of the same class, to match a big spectrum of object variants in a class. A drawback of this method is the need of training data and especially the time needed to train the network. In particular the time needed to make the neuronal network robust against camera rolling was too high. Therefore, another approach is used to find the correct pose including the camera rolling. It uses again voxel grid, which have the drawback of being computational expensive during execution time.

Overall this thesis tries to explain the current state-of-the-art in object detection and alignment. It does that by implementing several approaches which can then be used in the resulting user application. This application provides a simple way to create a virtualized scene from a real world scene.
This thesis can be seen as groundwork on which further experimentation and research can happen.

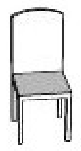
APPENDIX

A ESF Test Models

(a) book01


(b) book02

(c) chair01


(d) chair02

(e) chest01

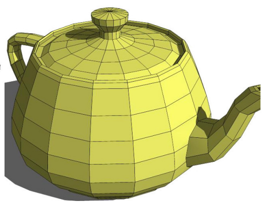
(f) chest02

(g) hammer01

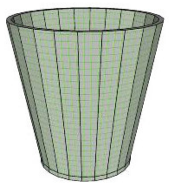
(h) hammer02

(i) table01

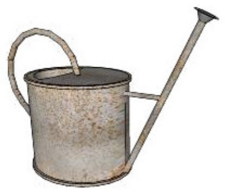
(j) table02


(k) teapot01

(l) teapot02


(m) trash01

(n) trash02


(o) wateringcan01

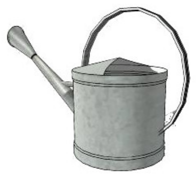

(p) wateringcan02

Figure 57: Models used to test the ESF descriptor (Figure 41).

B Scan Results

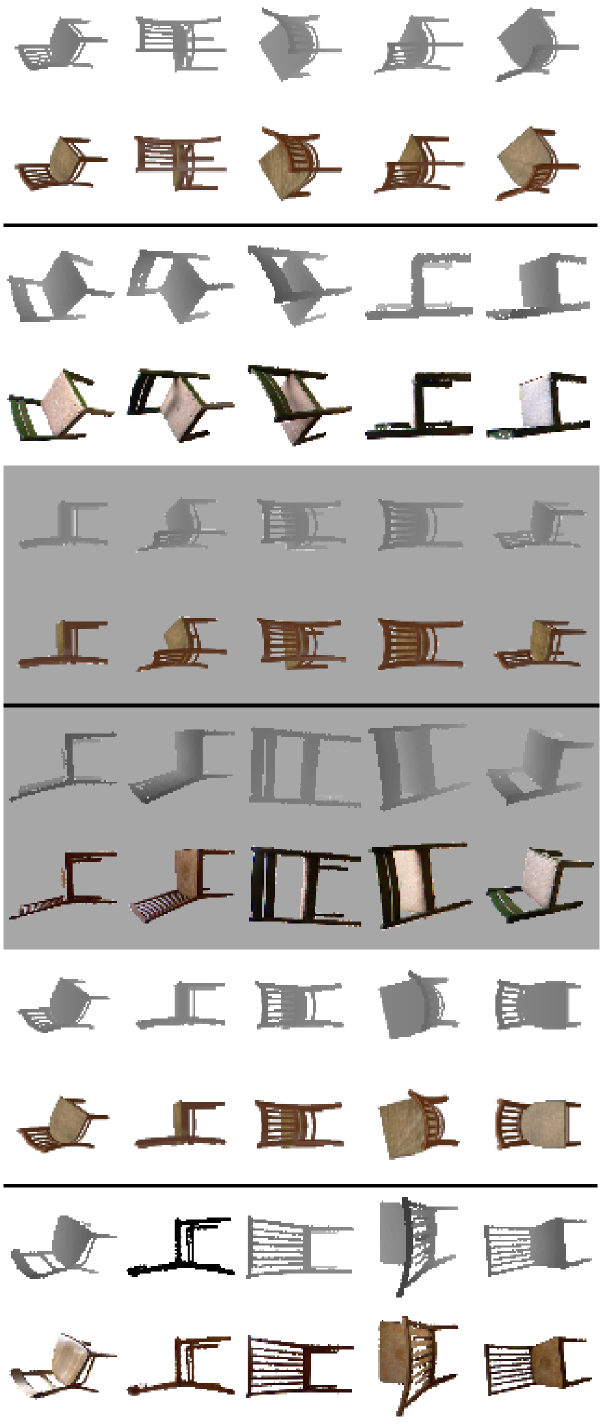


Figure 58: Results of a real scan query (first image pair respectively) and the returned template poses found by the CNN trained with two chairs (second image pair respectively).

C Chair Single Trained

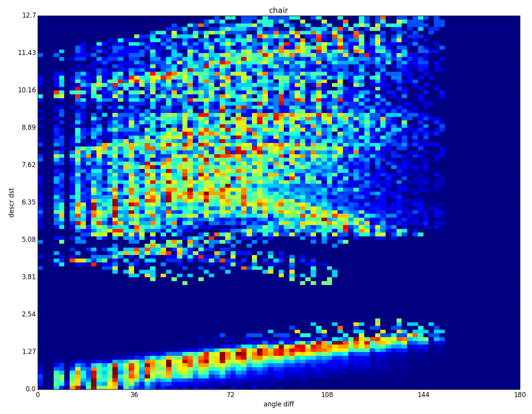


Figure 59: This figure visualizes the correlations between the descriptor distance (Y-axis) and the similarity of the pose (X-axis). The correlation shows clearly, that training the CNN with one objects is not enough, to get a good descriptor.

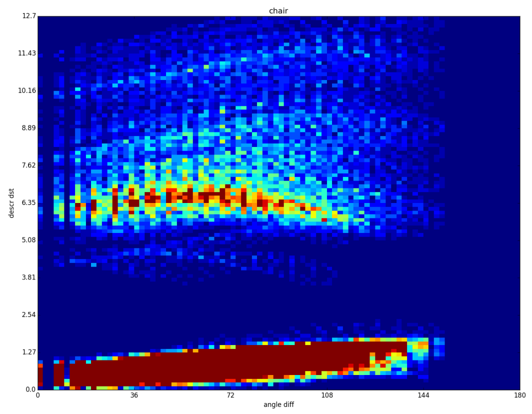


Figure 60: This figure (clipped colour scale) shows that the main part of the correlation forms a nearly horizontal line.

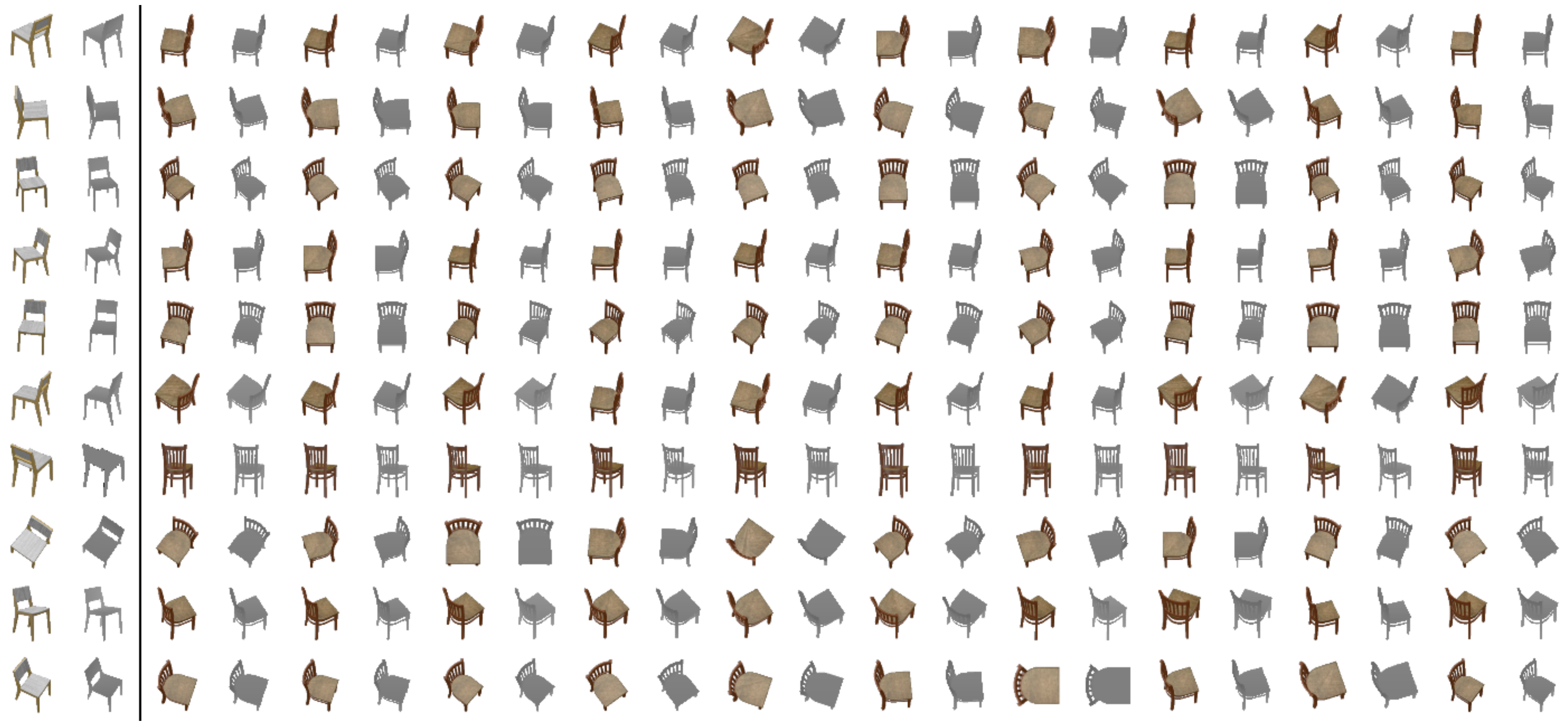


Figure 61: Results of a query (first image pair) and the returned template poses found by the CNN trained with one object. The results are sorted starting from the best match on the left side. Most of the queries return a wrong first match and also the further matches do not contain the right pose.

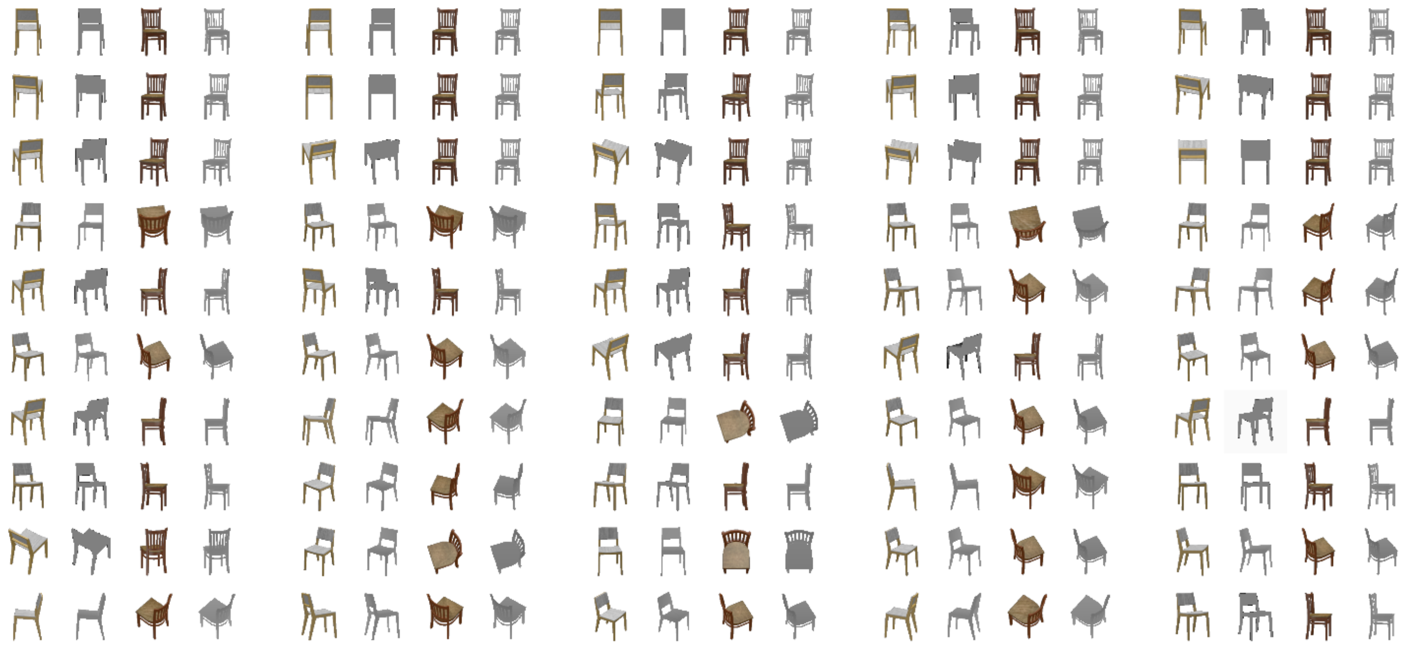


Figure 62: This Figure shows the queries (left image pair) for which the CNN trained with one object returned a false match (right image pair).

D Chair Two Trained

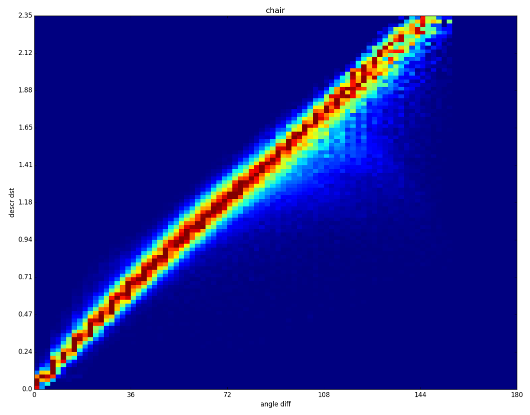


Figure 63: This figure visualizes the correlations between the descriptor distance (Y-axis) and the similarity of the pose (X-axis). The correlation forms nearly a line with 45 degree pitch, which means that it would map the similarity of the pose.

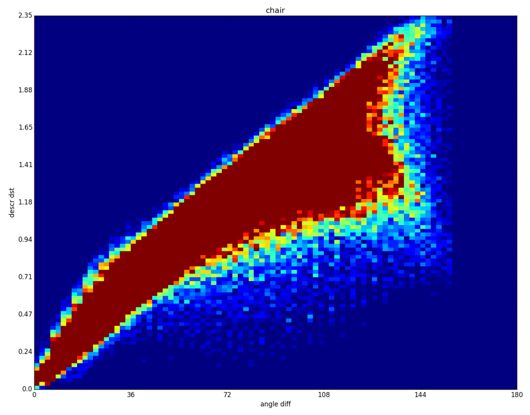


Figure 64: This figure (clipped colour scale) shows that there exists a few outliers, which break out from the line.

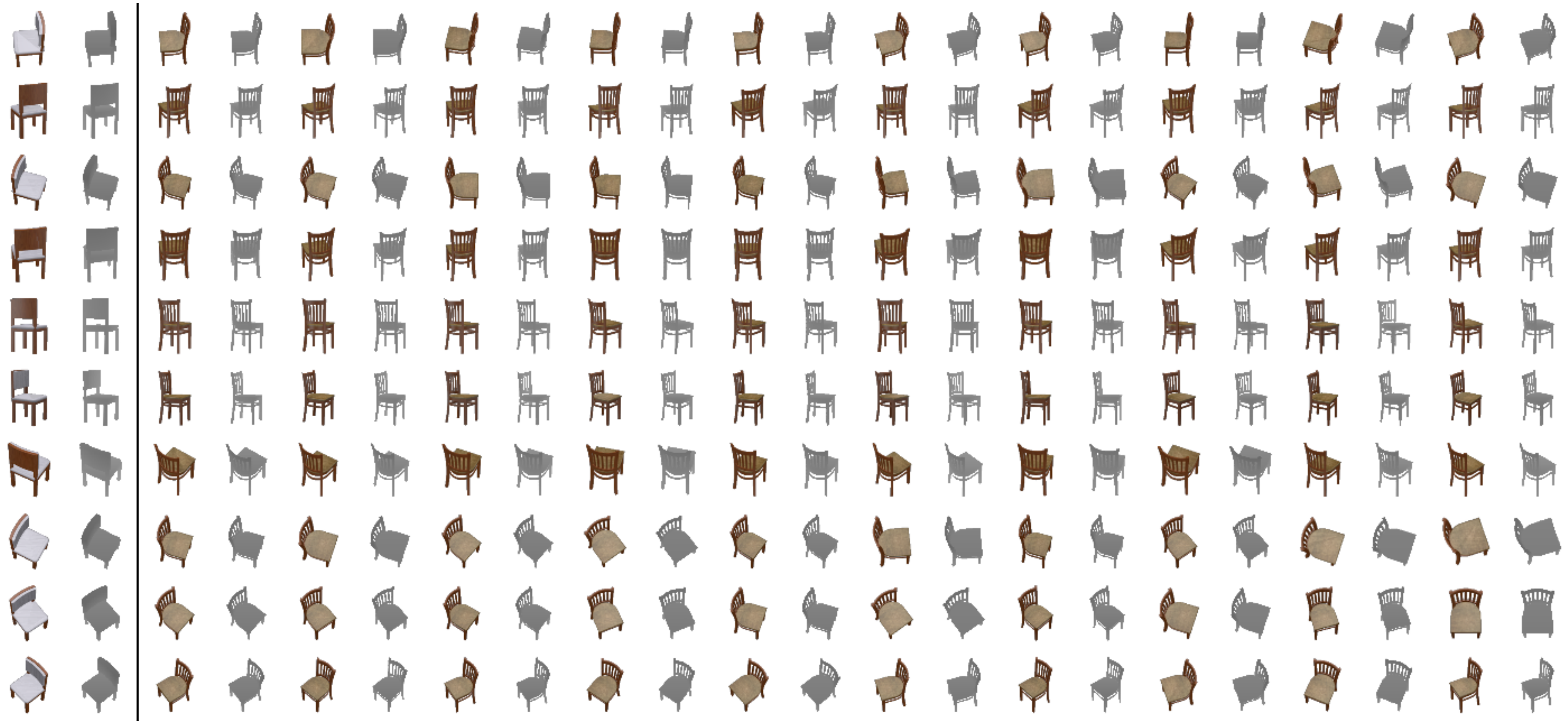


Figure 65: Results of a query (first image pair) and the returned template poses found by the CNN trained with two objects. The results are sorted starting from the best match on the left side. It can be seen, that the first match is not always the best matching pose (e.g. sixth row). But since the pose returned by the CNN will be refined, it has no influence if the returned pose is not the perfect matching pose.

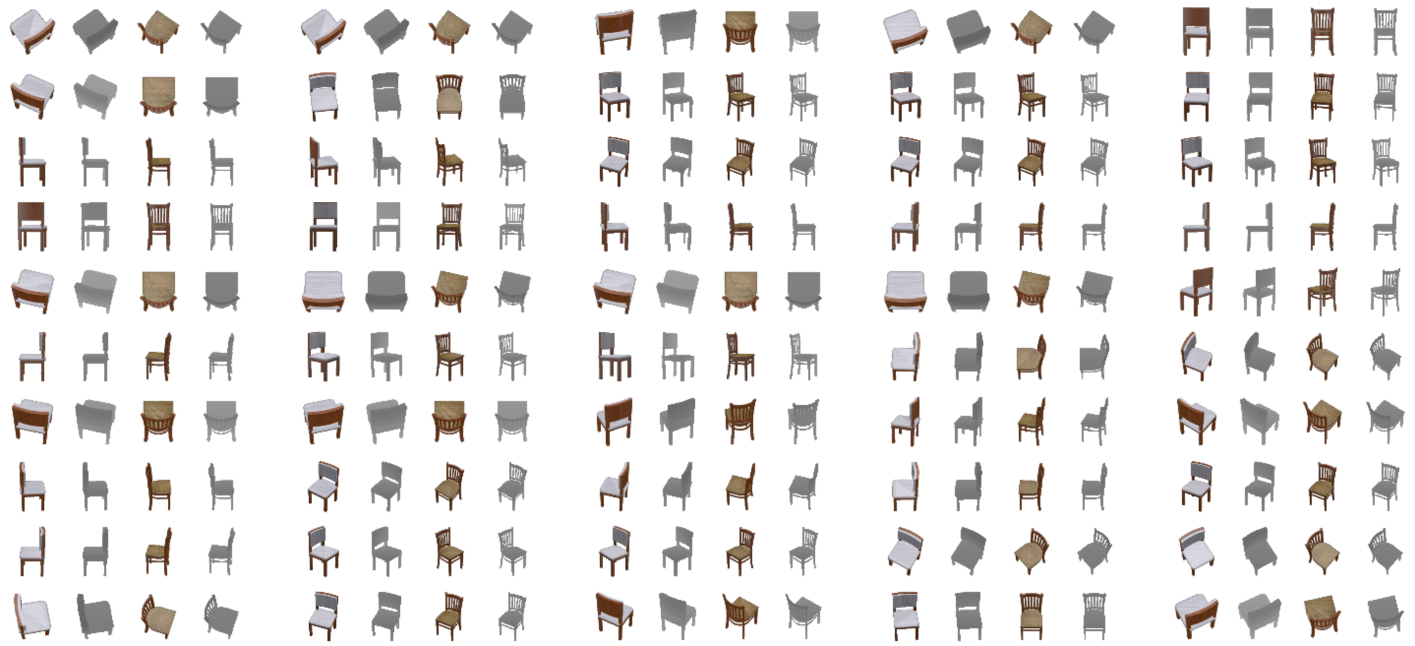


Figure 66: This Figure shows the queries (left image pair) for which the CNN trained with two objects returned a false match (right image pair).

E Chair Three Trained

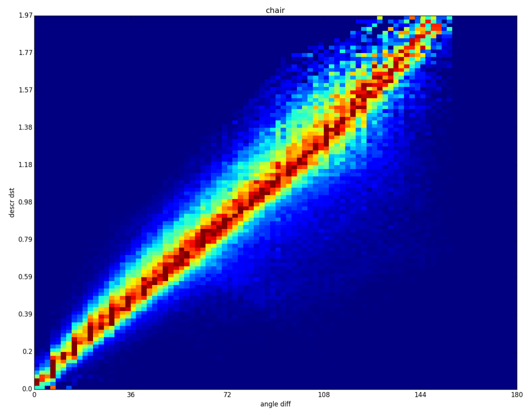


Figure 67: This figure visualizes the correlations between the descriptor distance (Y-axis) and the similarity of the pose (X-axis). The correlation forms nearly a line with 45 degree pitch, which means that it would map the similarity of the pose.

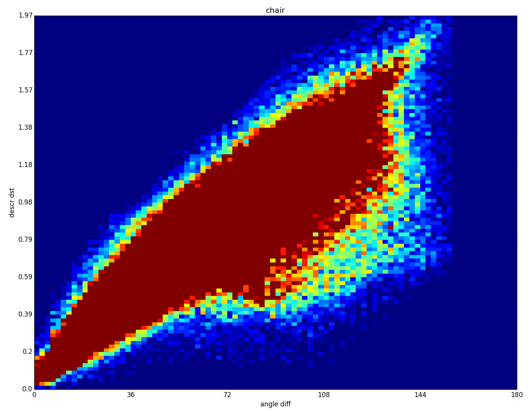


Figure 68: This figure (clipped colour scale) shows that there are more outlier, which break out from the line, than in the previous case, where the CNN was trained with two objects.

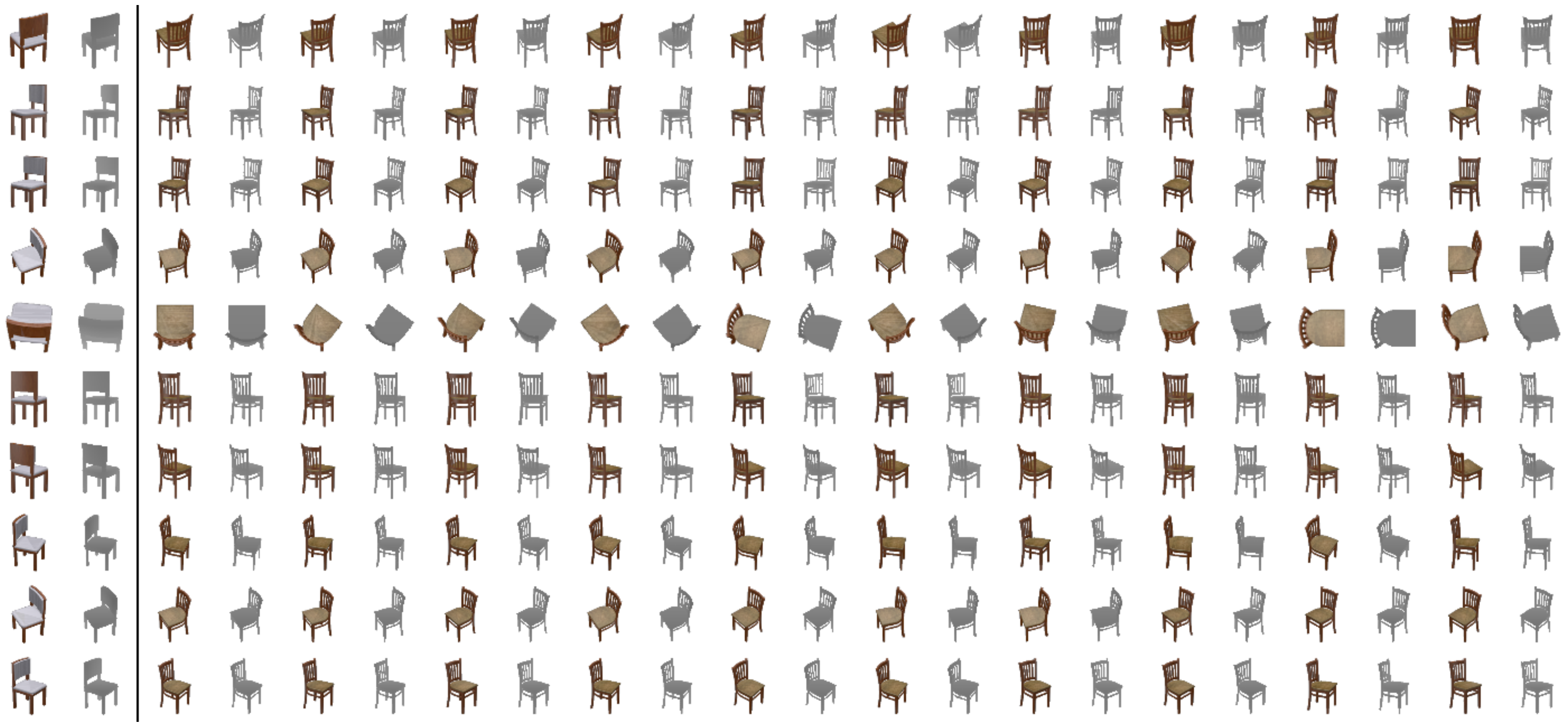


Figure 69: Results of a query (first image pair) and the returned template poses found by the CNN trained with three objects. The results are sorted starting from the best match on the left side. It can be seen, that the first match is not always the best matching pose.

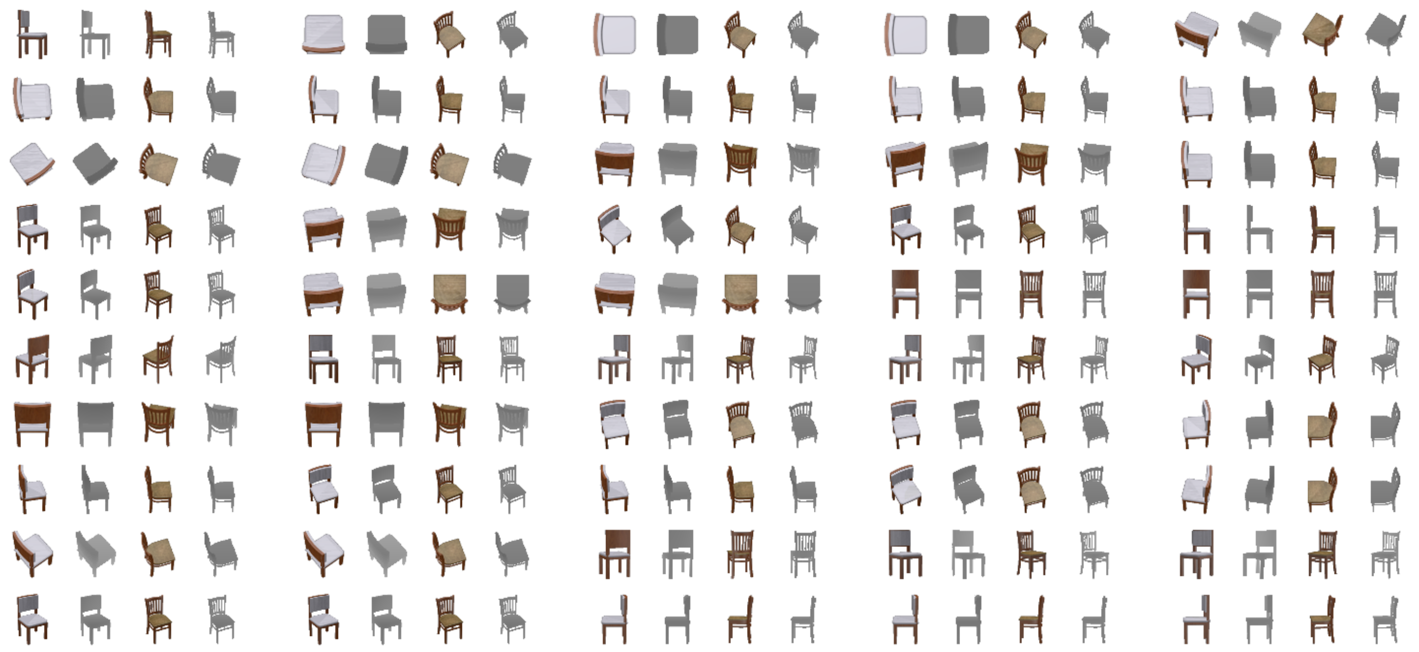


Figure 70: This Figure shows the queries (left image pair) for which the CNN trained with three objects returned a false match (right image pair).

F Chair Upside Down

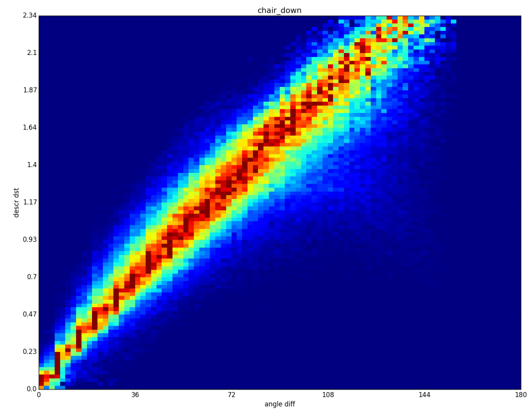


Figure 71: This figure visualizes the correlations between the descriptor distance (Y-axis) and the similarity of the pose (X-axis). The correlation forms nearly a line with 45 degree pitch, which means that it would map the similarity of the pose.

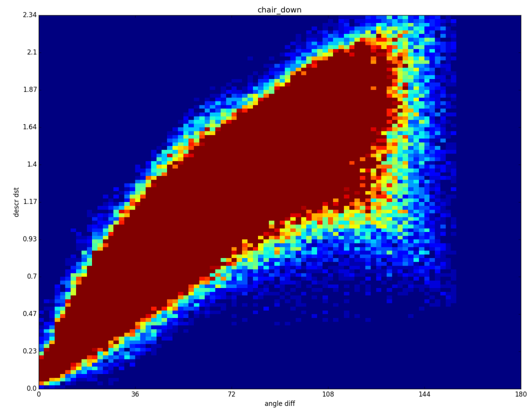


Figure 72: This figure (clipped colour scale) shows that there exists a few outliers, which break out from the line.

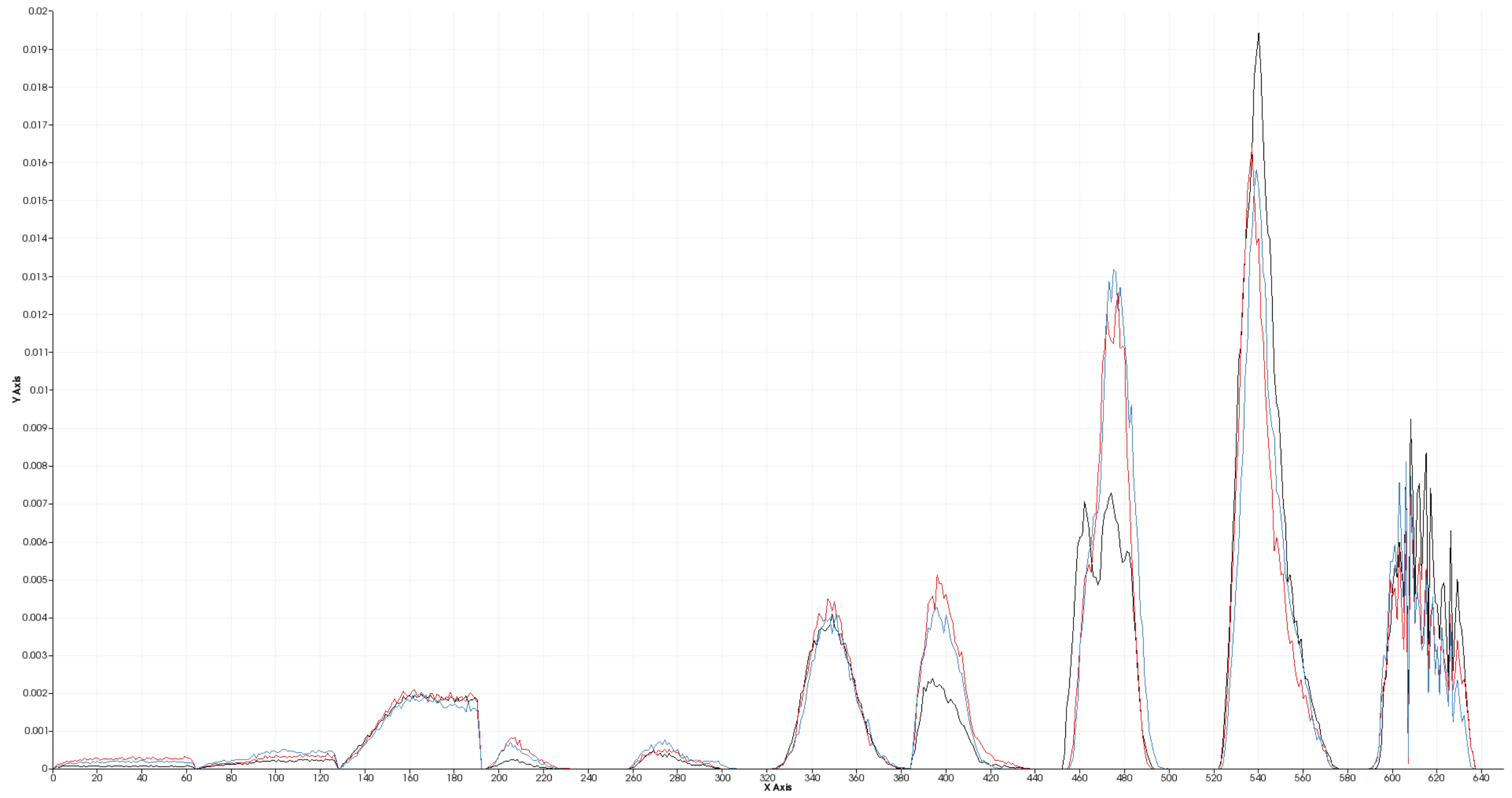


Figure 73: ESF histogram of the three models used to train the network. It can be seen, that even-tough every histogram has the same appearance typical for this class, the histograms are slightly different in some portions (e.g. 390-440) of the histogram.

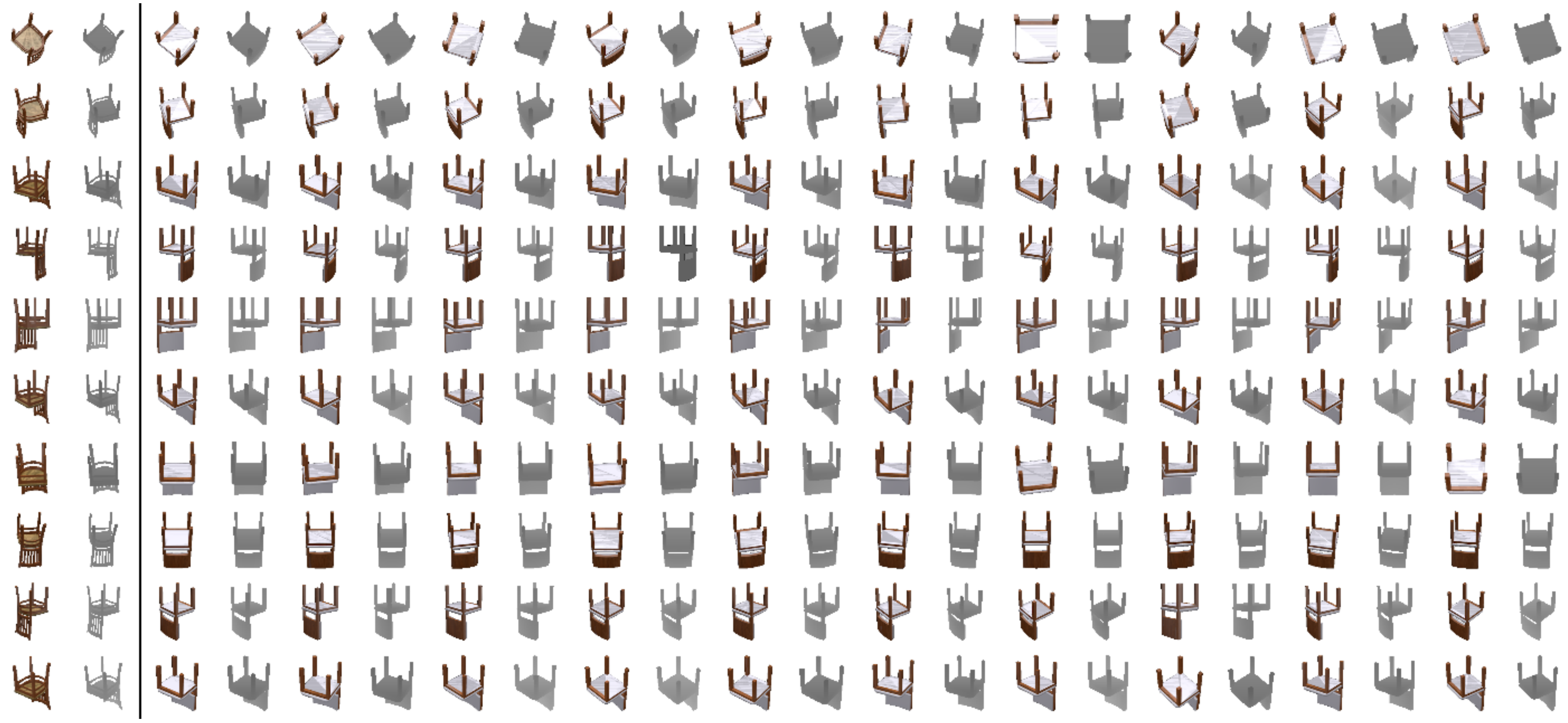


Figure 74: Results of a query (first image pair) and the returned template poses found by the CNN trained with two objects. The results are sorted starting from the best match on the left side. It can be seen, that the first match is not always the best matching pose. But since the pose returned by the CNN will be refined, it has no influence if the returned pose is not the perfect matching pose.

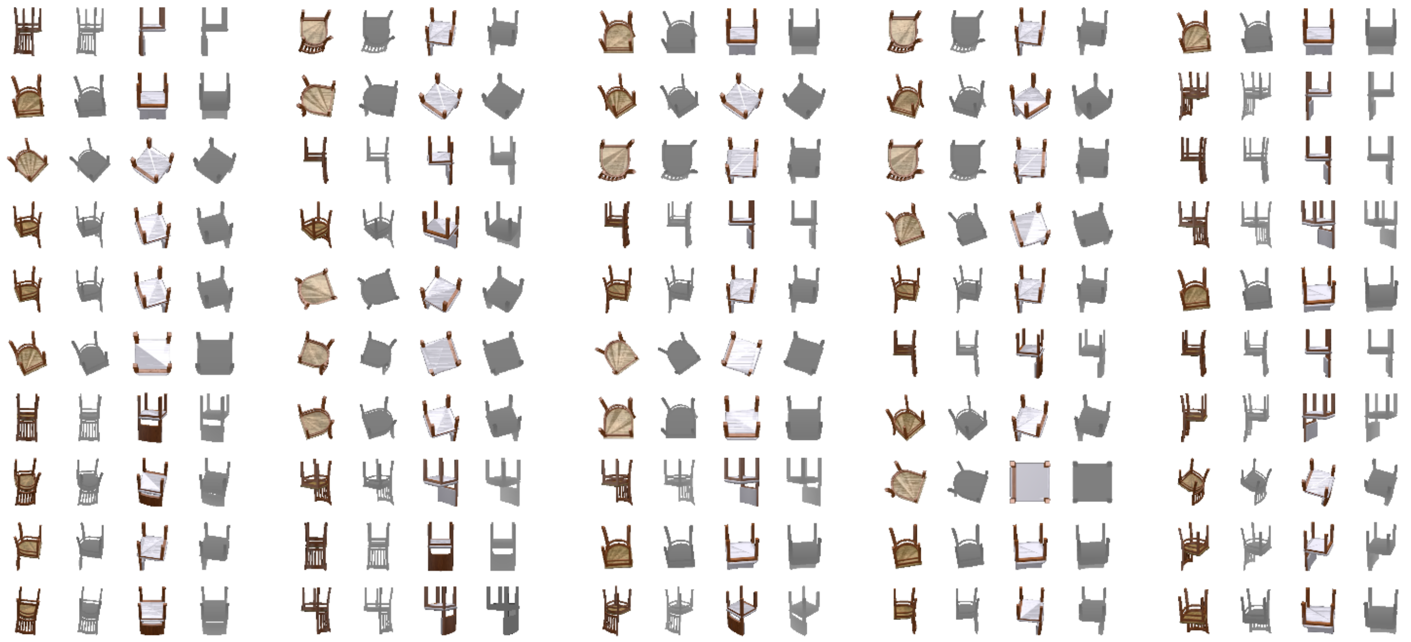


Figure 75: This Figure shows the queries (left image pair) for which the CNN trained with two objects returned a false match (right image pair).

G Hammer

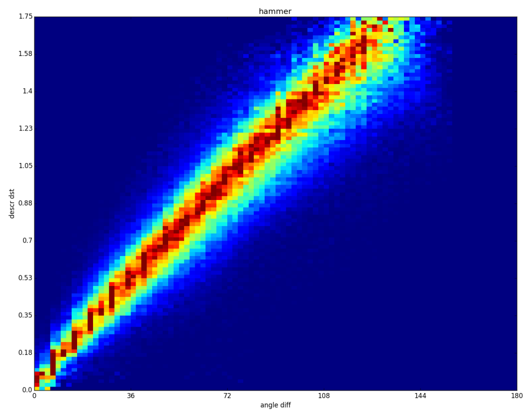


Figure 76: This figure visualizes the correlations between the descriptor distance (Y-axis) and the similarity of the pose (X-axis). The correlation forms nearly a line with 45 degree pitch, which means that it would map the similarity of the pose.

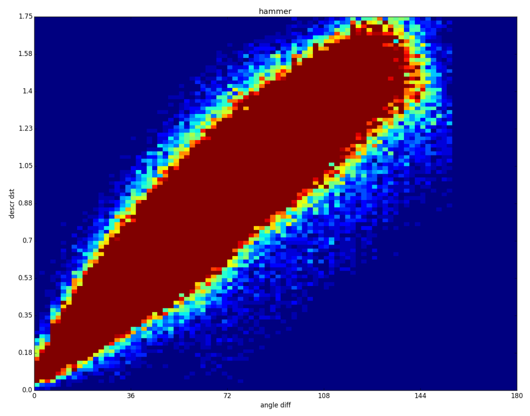


Figure 77: This figure (clipped colour scale) shows that there exists a few outliers, which break out from the line, but with are still in the surrounding of the line.

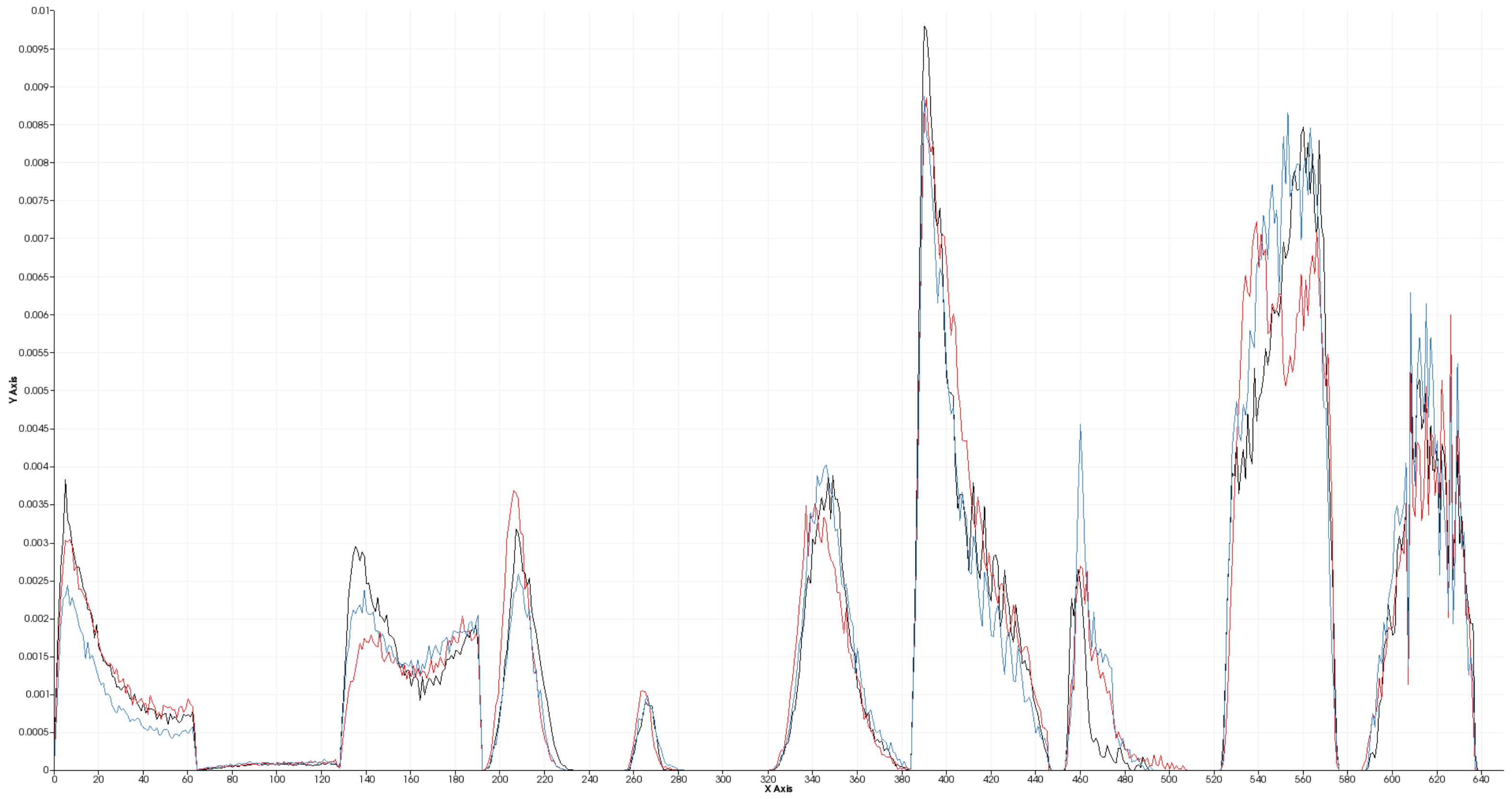


Figure 78: ESF histogram of the three models used to train the network. It can be seen, that even-tough every histogram has the same appearance typical for this class, the histograms are slightly different in some portions (e.g. 130-190) of the histogram.

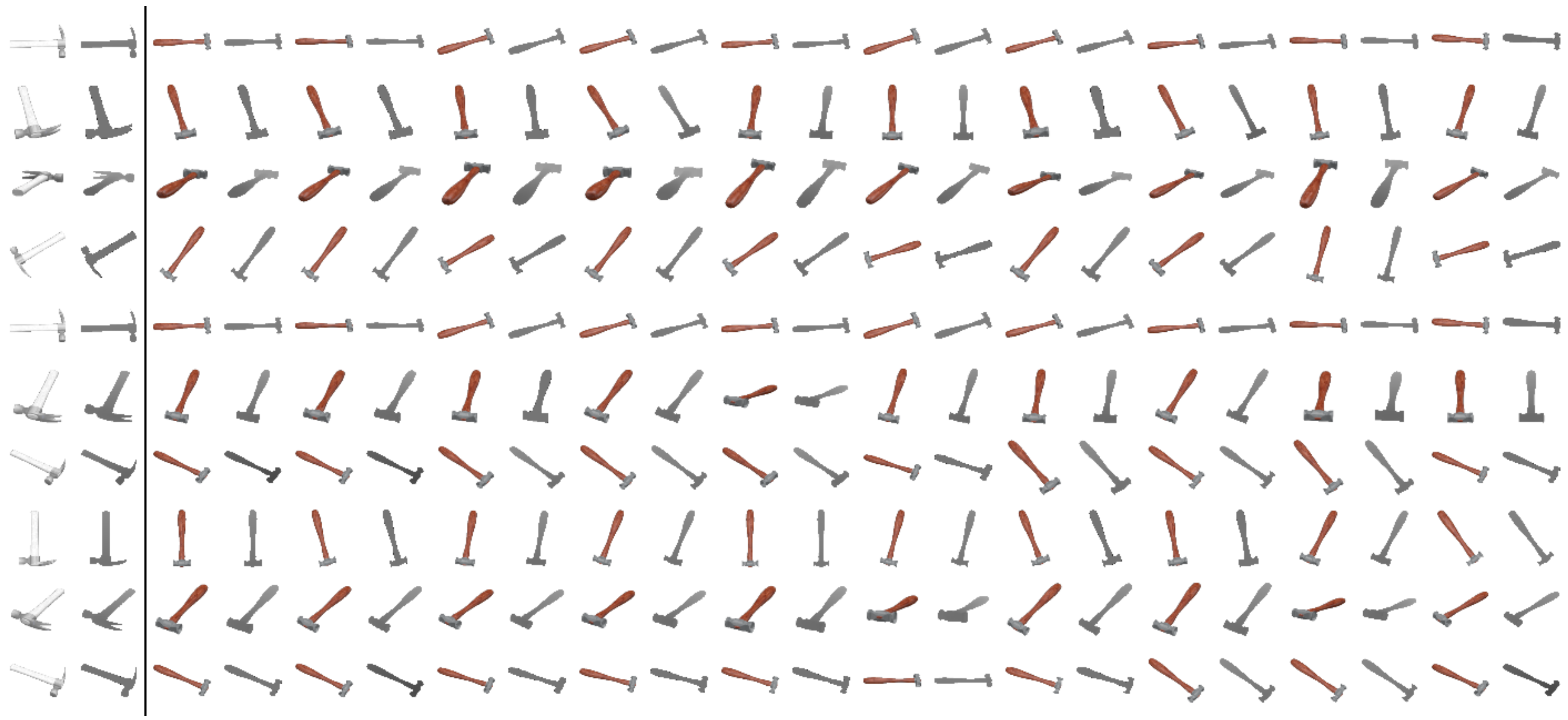


Figure 79: Results of a query (first image pair) and the returned template poses found by the CNN trained with two objects. The results are sorted starting from the best match on the left side. It can be seen, that the first match is not always the best matching pose (e.g. fourth row). But since the pose returned by the CNN will be refined, it has no influence if the returned pose is not the perfect matching pose.

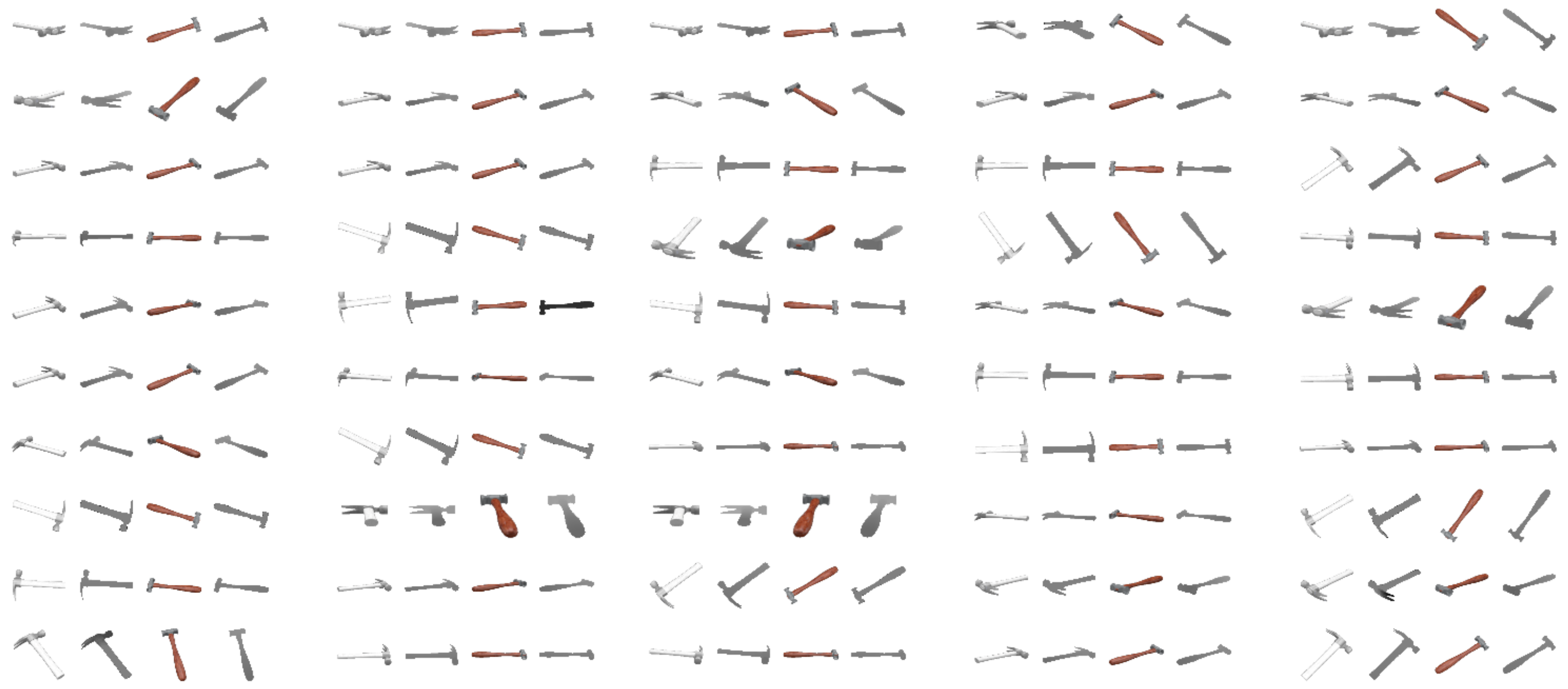


Figure 80: This Figure shows the queries (left image pair) for which the CNN trained with two objects returned a false match (right image pair).

H Table

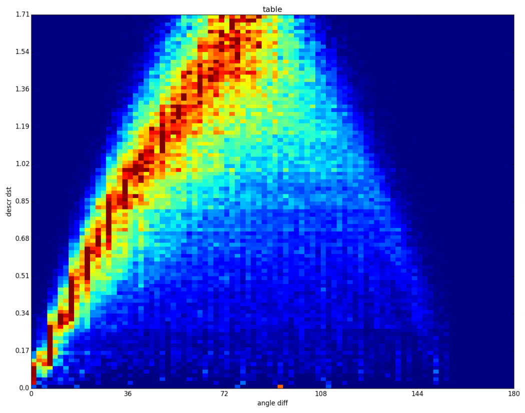


Figure 81: This figure visualizes the correlations between the descriptor distance (Y-axis) and the similarity of the pose (X-axis). The main part of the drawn correlation forms a line, although there are many outliers.

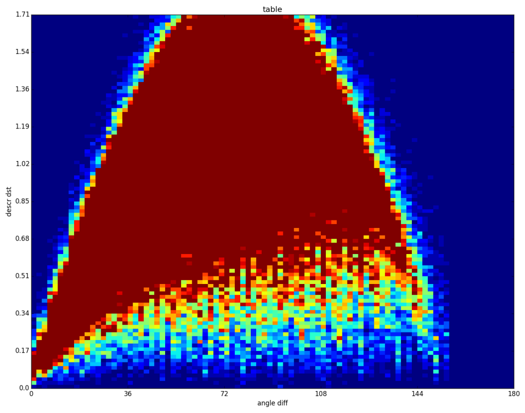


Figure 82: This figure (clipped colour scale) shows that there exists a lot of outliers, which spread far away from the line.

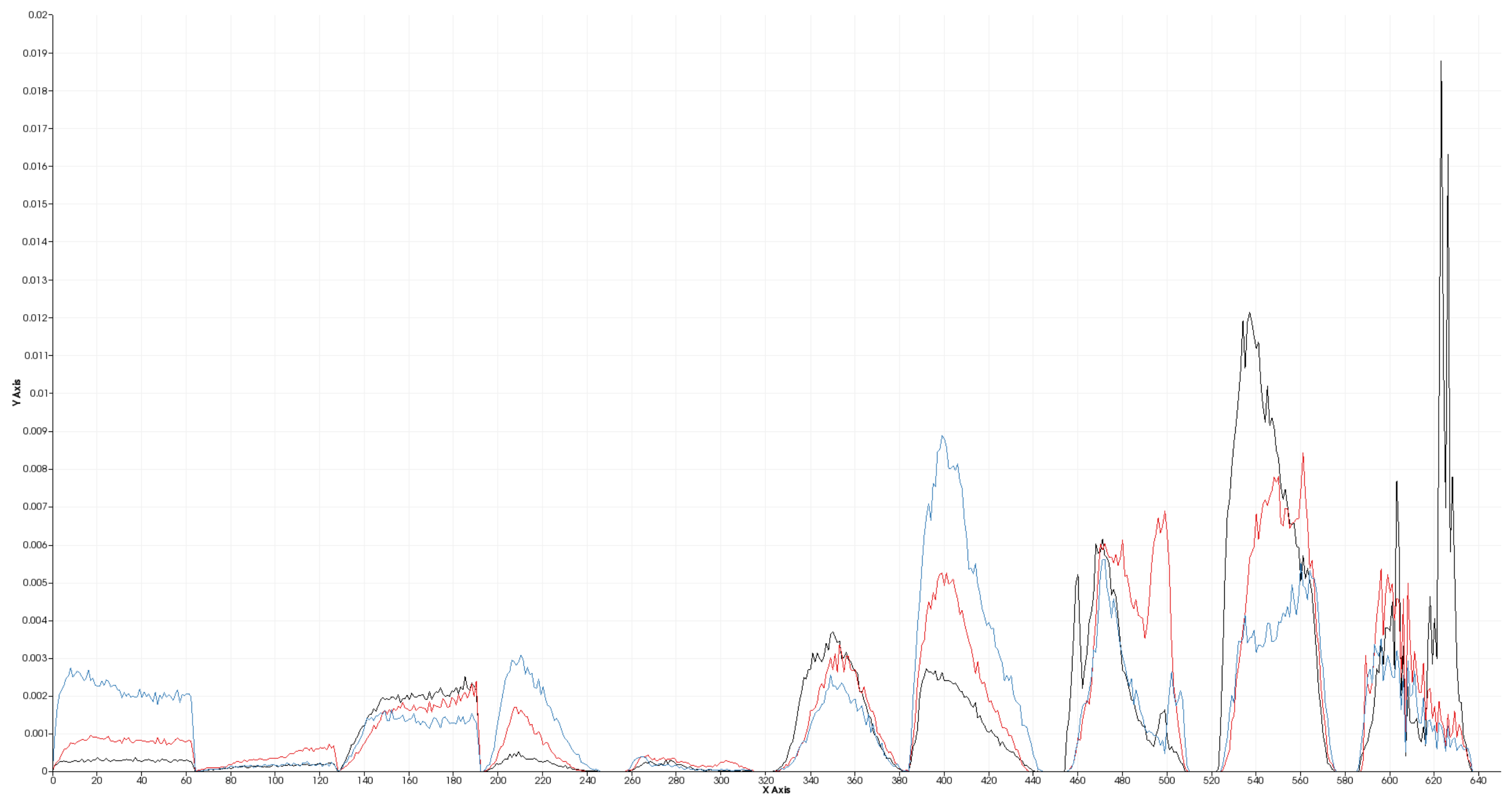


Figure 83: ESF histogram of the three models used to train the network. It can be seen, that even-tough every histogram has the same appearance typical for this class, the histograms are slightly different in some portions (e.g. 455-500) of the histogram.

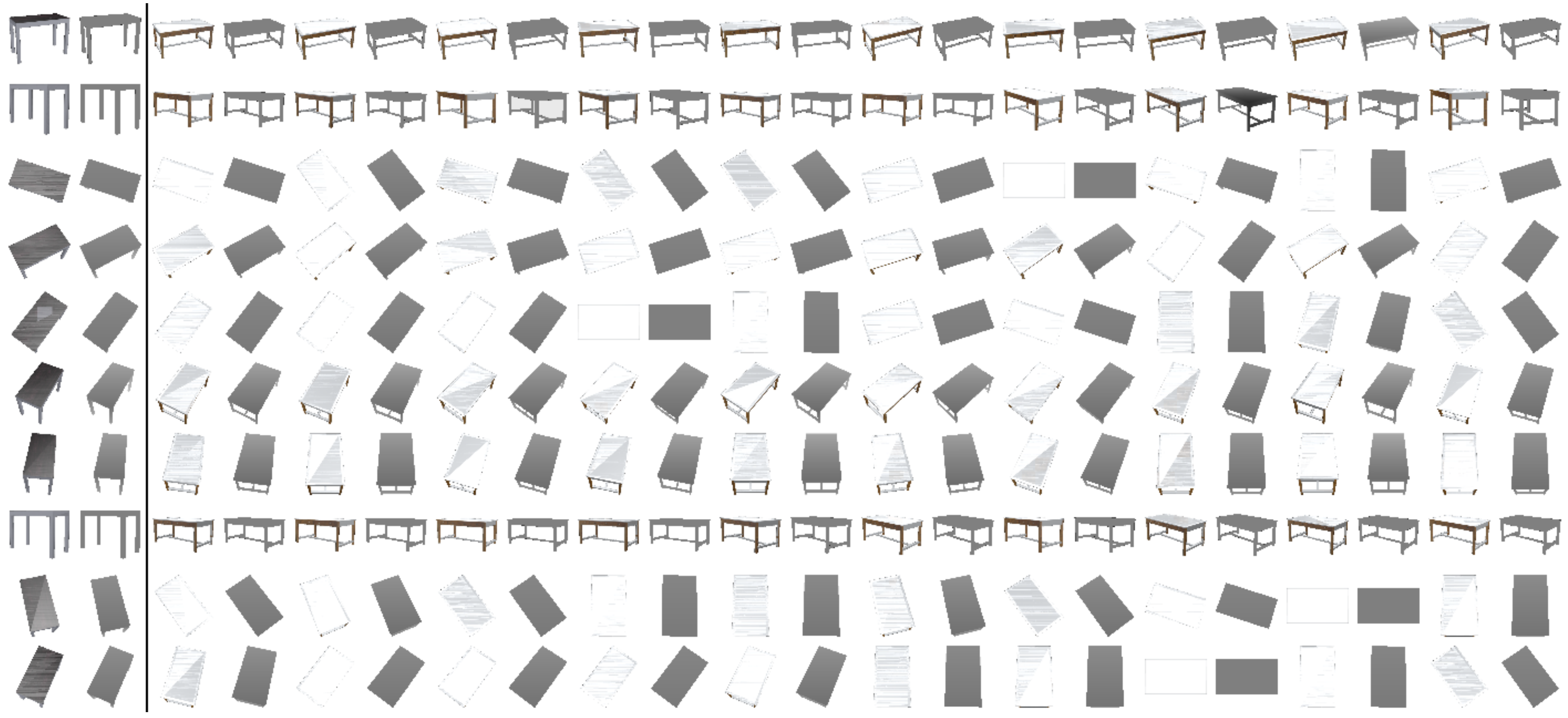


Figure 84: Results of a query (first image pair) and the returned template poses found by the CNN trained with two objects. The results are sorted starting from the best match on the left side. It can be seen, that the first match is not always the best matching pose (e.g. ninth row). But since the pose returned by the CNN will be refined, it has no influence if the returned pose is not the perfect matching pose.

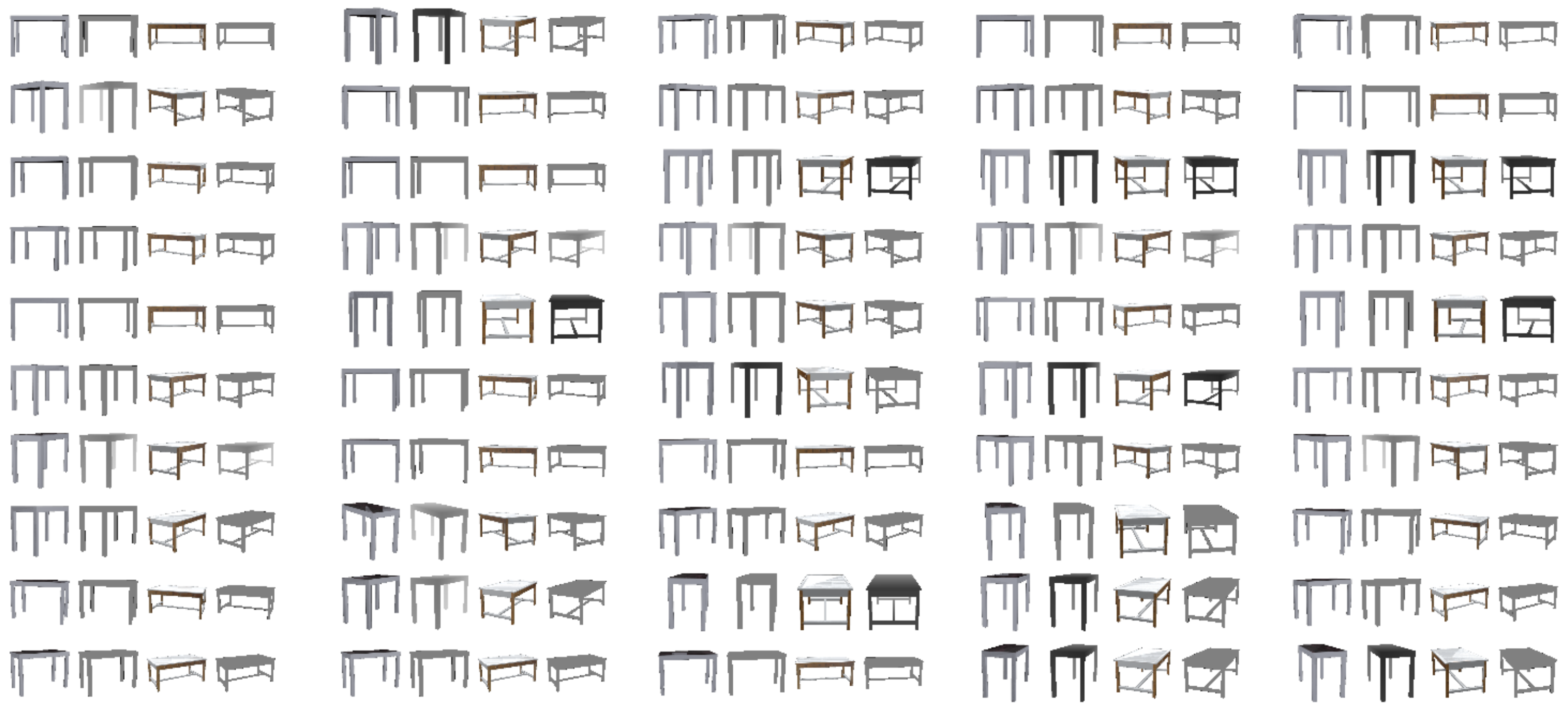


Figure 85: This Figure shows the queries (left image pair) for which the CNN trained with two objects returned a false match (right image pair).

I Watering Can

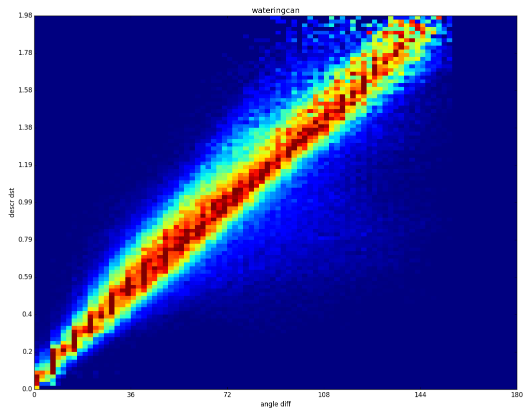


Figure 86: This figure visualizes the correlations between the descriptor distance (Y-axis) and the similarity of the pose (X-axis). The correlation forms nearly a line with 45 degree pitch, which means that it would map the similarity of the pose.

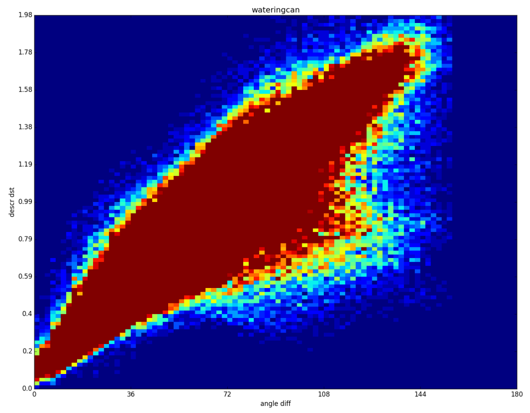


Figure 87: This figure (clipped colour scale) shows that there exists a few outliers, which break out from the line.

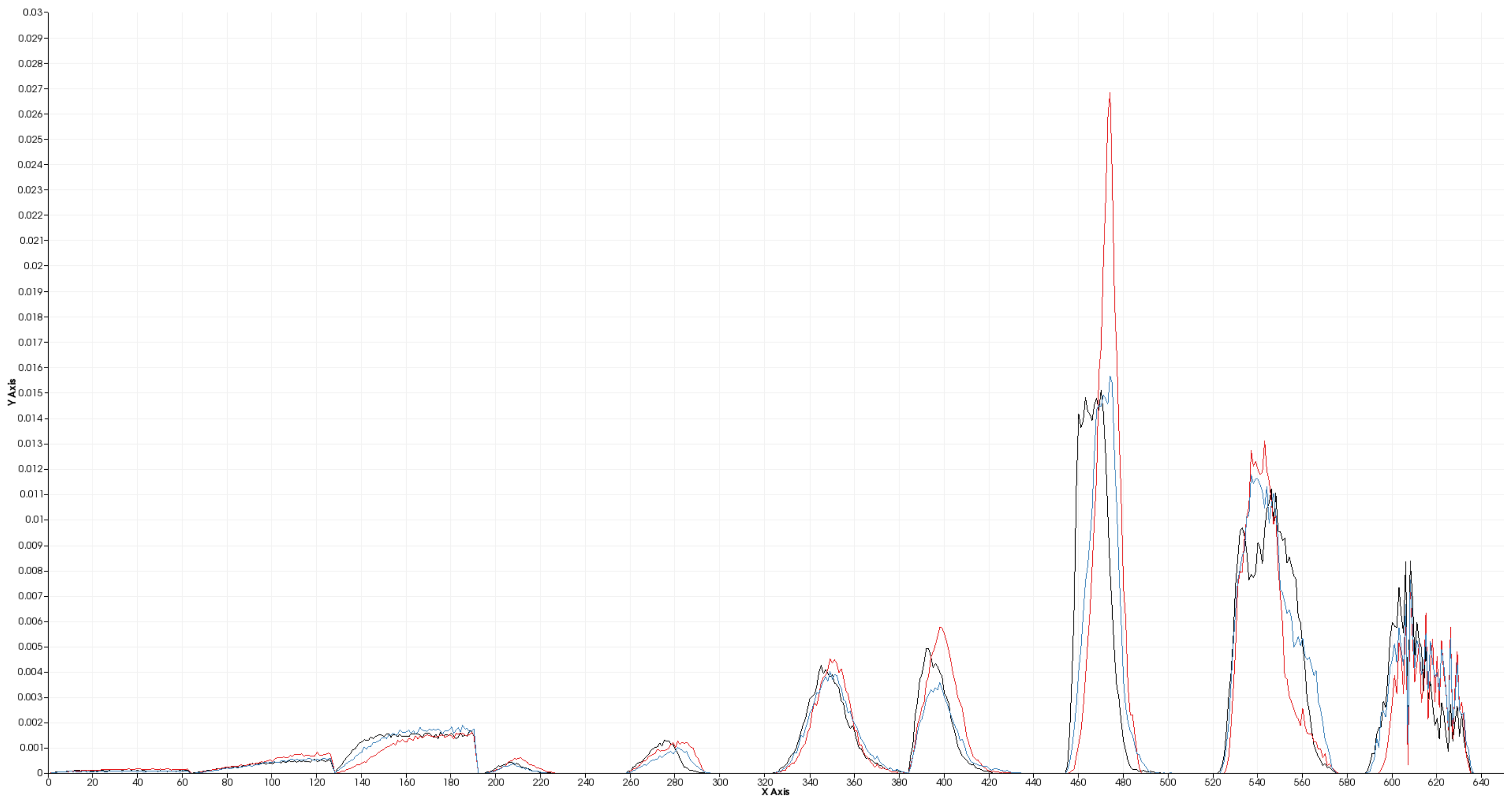


Figure 88: ESF histogram of the three models used to train the network. It can be seen, that even-tough every histogram has the same appearance typical for this class, the histograms are slightly different in some portions (e.g. 455-600) of the histogram.

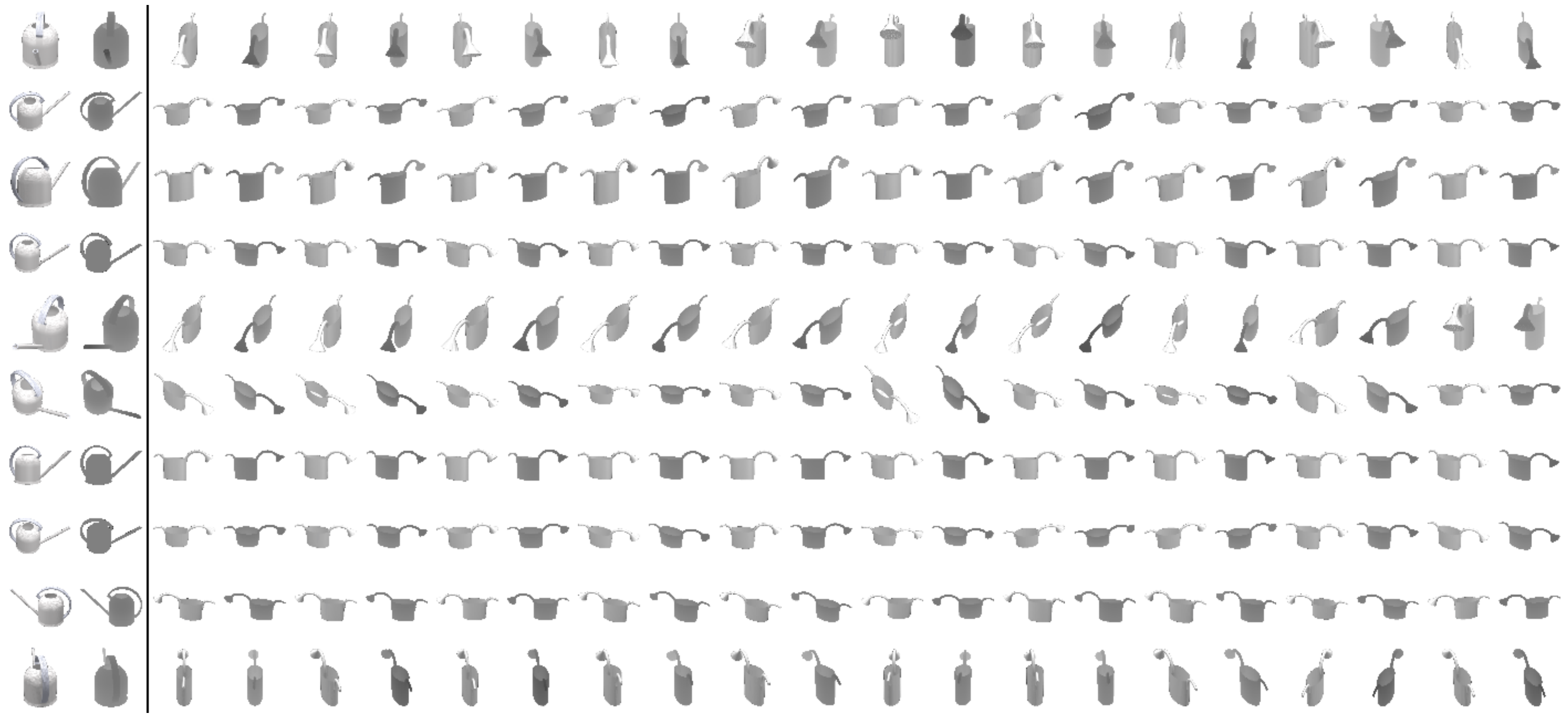


Figure 89: Results of a query (first image pair) and the returned template poses found by the CNN trained with two objects. The results are sorted starting from the best match on the left side. It can be seen, that the first match is not always the best matching pose (e.g. fifth row). But since the pose returned by the CNN will be refined, it has no influence if the returned pose is not the perfect matching pose.

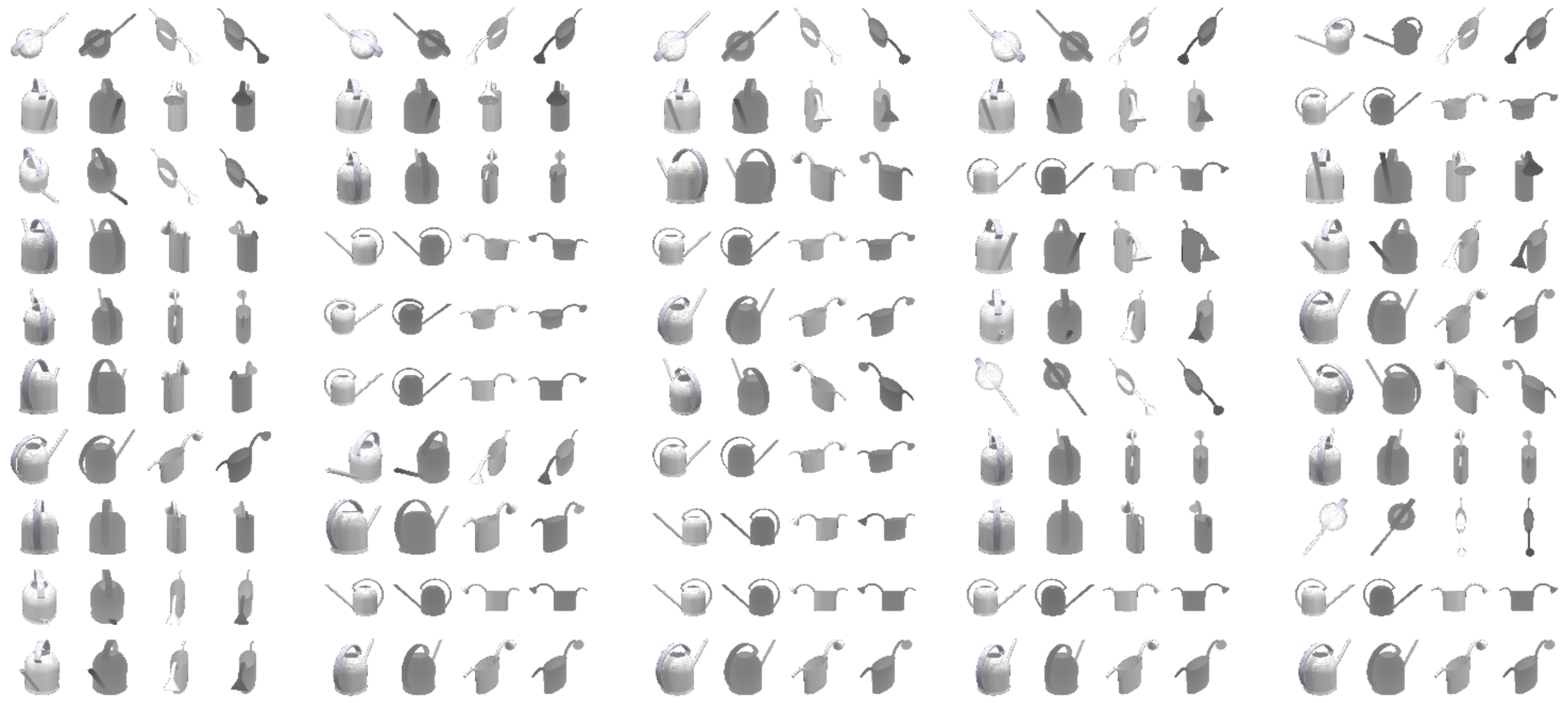


Figure 90: This Figure shows the queries (left image pair) for which the CNN trained with two objects returned a false match (right image pair).

References

[1] Aldoma, Aitor ; Marton, Zoltan-Csaba ; Tombari, Federico ; Wohlkinger, Walter ; Potthast, Christian ; Zeisl, Bernhard ; Rusu, Radu ; Gedikli, Suat ; Vincze, Markus: Tutorial: Point Cloud Library: Three-Dimensional Object Recognition and 6 DOF Pose Estimation. (2012). http://dx.doi.org/10.1109/MRA.2012.2206675. – DOI 10.1109/MRA.2012.2206675 7

[2] Aldoma, Aitor ; Vincze, Markus ; Blodow, Nico ; Gossow, David ; Gedikli, Suat ; Rusu, Radu B. ; Bradski, Gary: CAD-model recognition and 6DOF pose estimation using 3D cues 22, 24, 25

[3] Alexandre, Luís A: 3D descriptors for object and category recognition: a comparative evaluation, 2012 11, 15

[4] Dalal, N. ; Triggs, B.: Histograms of Oriented Gradients for Human Detection, 20-26 June 2005 25, 26, 29

[5] Dorai, C. ; Jain, A. K.: COSMOS-A representation scheme for 3D free-form objects. (1997). http://dx.doi.org/10.1109/34.625113. – DOI 10.1109/34.625113 21

[6] Drost, Bertram ; Ulrich, Markus ; Navab, Nassir ; Ilic, Slobodan: Model globally, match locally: Efficient and robust 3D object recognition. (2010). http://dx.doi.org/10.1109/CVPR.2010.5540108. – DOI 10.1109/CVPR.2010.5540108 10, 14, 15

[7] Dyer, Charles R.: *Feature Detection and Matching*. http://pages.cs.wisc.edu/~dyer/cs766/slides/matching/matching-4up.pdf 8

[8] Fischler, Martin A. ; Bolles, Robert C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. (1981). http://dx.doi.org/10.1145/358669.358692. – DOI 10.1145/358669.358692 17

[9] Frome, Andrea ; Huber, Daniel ; Kolluri, Ravi ; Bülow, Thomas ; Malik, Jitendra: Recognizing Objects in Range Data Using Regional Point Descriptors. (2004). http://dx.doi.org/10.1007/978-3-540-24672-5{_}18. – DOI 10.1007/978–3–540–24672–5_18 8, 10

[10] Golovinskiy, Aleksey ; Funk, Thomas: Min-cut based segmentation of point clouds 20, 21, 51

[11] Guo, Yulan ; Sohel, Ferdous ; Bennamoun, Mohammed ; Lu, Min ; Wan, Jianwei: Rotational Projection Statistics for 3D Local Surface Description and Object Recognition. (2013). http://dx.doi.org/10.1007/s11263-013-0627-y. – DOI 10.1007/s11263–013–0627–y 16

[12] Harris, C. ; Stephens, M.: A Combined Corner and Edge Detector 8

[13] Hinterstoisser, Stefan ; Holzer, Stefan ; Cagniart, Cedric ; Ilic, Slobodan ; Konolige, Kurt ; Navab, Nassir ; Lepetit, Vincent: Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes 25, 26, 27, 29

[14] Holzer, Stefan ; Hinterstoisser, Stefan: *PCL :: Object Detection – LINEMOD*. http://www.pointclouds.org/assets/cvpr2012/PCL_Object_Detection_LINEMOD.pdf 27

[15] Ichim, Alexandru-Eugen: *PFHRGB Feature results!* http://www.pointclouds.org/blog/gsoc/aichim/index.php 14

[16] Johnson, A. E. ; Hebert, M.: Using spin images for efficient object recognition in cluttered 3D scenes. (1999). http://dx.doi.org/10.1109/34.765655. – DOI 10.1109/34.765655 7, 8, 9, 10

[17] Kim, Young M. ; Mitra, Niloy J. ; Huang, Qixing ; Guibas, Leonidas: Guided Real-Time Scanning of Indoor Objects. (2013). http://dx.doi.org/10.1111/cgf.12225. – DOI 10.1111/cgf.12225 5, 23, 25, 34, 38, 54

[18] Lazebnik, Svetlana ; Schmid, Cordelia ; Ponce, Jean: A sparse texture representation using local affine regions. (2005). http://dx.doi.org/10.1109/TPAMI.2005.151. – DOI 10.1109/TPAMI.2005.151 12

[19] Li, Yangyan ; Dai, Angela ; Guibas, Leonidas ; Niessner, Matthias: Database-Assisted Object Retrieval for Real-Time 3D Reconstruction. (2015). http://dx.doi.org/10.1111/cgf.12573. – DOI 10.1111/cgf.12573 5, 23

[20] Marton, Z. ; Pangercic, D. ; Blodow, N. ; Kleinehellefort, J. ; Beetz, M.: General 3D modelling of novel objects from a single view 4, 14

[21] Marton, Zoltan-Csaba ; Pangercic, Dejan ; Rusu, Radu B. ; Holzbach, Andreas ; Beetz, Michael: Hierarchical object geometric categorization and appearance classification for mobile manipulation 22

[22] Pearson, Karl: LIII. On lines and planes of closest fit to systems of points in space. (1901). http://dx.doi.org/10.1080/14786440109462720. – DOI 10.1080/14786440109462720 46

[23] Rusu, R. B. ; Blodow, N. ; Marton, Z. C. ; Beetz, M.: *Aligning point cloud views using persistent feature histograms: [IROS 2008] ; Nice, France, 22 - 26 September 2008*. 2008. http://dx.doi.org/10.1109/IROS.2008.4650967. http://dx.doi.org/10.1109/IROS.2008.4650967 13

[24] Rusu, R. B. ; Bradski, G. ; Thibaux, R. ; Hsu, J.: Fast 3D recognition and pose using the Viewpoint Feature Histogram 22

[25] Rusu, Radu B.: *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*, Technische Universität München, Diss., 01.01.2009 19, 20

[26] Rusu, Radu B. ; Blodow, Nico ; Beetz, Michael: *Fast Point Feature Histograms (FPFH) for 3D registration*. 2009. http://dx.doi.org/10.1109/ROBOT.2009.5152473. http://dx.doi.org/10.1109/ROBOT.2009.5152473 17, 18, 47, 68

[27] Rusu, Radu B. ; Holzbach, Andreas ; Beetz, Michael ; Bradski, Gary: Detecting and segmenting objects for mobile manipulation 22

[28] Sipiran, Ivan ; Bustos, Benjamin: Harris 3D: A robust extension of the Harris operator for interest point detection on 3D meshes. (2011). http://dx.doi.org/10.1007/s00371-011-0610-y. – DOI 10.1007/s00371–011–0610–y 6, 43

[29] Steder, Bastian ; Rusu, Radu B. ; Konolige, Kurt ; Burgard, Wolfram: Point feature extraction on 3D range scans taking into account object boundaries 15

[30] Tombari, Federico: *Keypoints and Features*. http://www.pointclouds.org/assets/uploads/cglibs13_features.pdf 7

[31] Tombari, Federico ; Salti, Samuele ; Di Stefano, Luigi: *Unique shape context for 3d data description*. 2010. http://dx.doi.org/10.1145/1877808.1877821. http://dx.doi.org/10.1145/1877808.1877821 8, 11

[32] Tombari, Federico ; Salti, Samuele ; Di Stefano, Luigi: Unique Signatures of Histograms for Local Surface Description. Version: 2010. http://dx.doi.org/10.1007/978-3-642-15558-1{_}26. 2010. – DOI 10.1007/978–3–642–15558–1_26 8, 11, 68

[33] Tombari, Federico ; Salti, Samuele ; Di Stefano, Luigi: *A combined texture-shape descriptor for enhanced 3D feature matching: 11 - 14 Sept. 2011, Brussels, Belgium*. 2011. http://dx.doi.org/10.1109/ICIP.2011.6116679. http://dx.doi.org/10.1109/ICIP.2011.6116679 11

[34] Wikipedia: *Flood Fill*. https://en.wikipedia.org/wiki/Flood_fill 19

[35] Wikipedia: *RANSAC*. https://en.wikipedia.org/wiki/RANSAC 17

[36] Wohlhart, Paul ; Lepetit, Vincent: Learning descriptors for object recognition and 3D pose estimation 27, 28, 29, 39, 58

[37] Wohlkinger, Walter ; Vincze, Markus: Ensemble of shape functions for 3D object classification 23

[38] Yulan Guo ; Ferdous Ahmed Sohel ; Mohammed Bennamoun ; Min Lu ; Jianwei Wan: TriSI: A Distinctive Local Surface Descriptor for 3D Modeling and Object Recognition on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications, Barcelona, Spain, 21-24 February, 2013, 2013 16

[39] Zhong, Yu: Intrinsic shape signatures: A shape descriptor for 3D object recognition 6, 43