Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische
Friedrich-Wilhelms-Universität Bonn

Bachelor Thesis

Deep 3D Non-Rigid Registration for Novel Objects from RGB Images

Author:
Florian Huber

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Dr. Michael Weinmann

Supervisor:
Diego Rodriguez

November 10, 2019

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Place, Date

Signature

Abstract

3D object non-rigid registration for novel objects from RGB images aims to deform a given textured 3D model to match the geometry of an unknown object observed in only one RGB image. 3D object registration for given RGB-D data is a well known research area with several applications in robotics, but it is still an open problem when depth information is not available. The use of RGB over RGB-D cameras presents several advantages in terms of price, frame rate and resolution. Additional problems with inaccurate and complex depth calibrations are also avoided. In this thesis, we will present a novel idea to solve the 3D non-rigid registration problem from RGB images using convolutional neural networks. We will show how to represent non-rigid deformations to make them suitable for neural networks and how to build an appropriate dataset for learning. This approach relies on a neural network architecture based on the FlowNet architecture. Furthermore, we will address the problem of occluded parts, that occurs by using only one image for 3D registration. This is achieved by using a lower dimensional shape space that contains typical deformations of an object category. We will present experiments done on unseen objects and we will compare the results with other registration approaches.

Contents

1 Introduction

While humans are able to dexterously operate a tool, after they learned how to use it once, robots need to overcome a challenging generalization task to be able to achieve similar results in a dynamic environment. Non-rigid registration is a way of transferring knowledge to novel objects. After having information about what kind of tool should be used, non-rigid registration methods are capable of deforming a known object into a novel one, observed in the environment. In most cases, the observation is recorded with a RGB-D sensor to extract depth information of the unknown object. Furthermore, in a real world application a complete observation is often not possible, thus existing non-rigid registration approaches are aiming on deforming the known object towards the seen parts of the unknown object, while applying different constraints to conserve a consistent shape.
The method presented in this thesis introduces a way to perform non-rigid registration based on a single RGB image. The use of RGB over RGB-D cameras holds several advantages such as price, frame-rate and field of view. Additionally, using RGB cameras comes without calibrating the sensors before usage, as it has to be done with RGB-D cameras, where it is necessary to calibrate the depth channel to match the RGB channels. This is not only time consuming, but also a source of errors.
In this thesis, we present an image based representation for deformation fields similar to optical flow that can be estimated by Convolutional Neural Networks (CNNs) in an iterative fashion. We show how to capture a synthetic dataset from limited access to high quality textured 3D models, using the Coherent Point Drift (CPD) method to obtain ground truth deformations. We aim to use only a single image as input for the non-rigid registration, and overcome the problem of occluded parts. This thesis will describe how we use Principal Component Analysis (PCA) together with a variety of training objects to capture typical deformations of any object class, that is later used to solve the problem of occluded parts. Final experiments will demonstrate the capabilities for the non-rigid registration of the drill category. We compare the results of two state-of-the-art approaches for non-rigid registration on point clouds and reach a competitive accuracy, even though we are solving a harder problem because of the lack of depth information.
Our contributions include: (i) finding a representation for deformation fields, that

can be presented to neural networks, (ii) proposing a network architecture, capable of estimating a deformation from RGB images in an iterative fashion, (iii) building a synthetic dataset, consisting of images of rendered 3D meshes together with ground truth deformations from a canonical object towards the training objects calculated with CPD, (iv) presenting a solution to the problem of occluded parts via Principal Component Analysis, completing a partially defined deformation field, (iv) execution of experiments and evaluating our results against non-rigid registration approaches relying on RGB-D data.

2 Related Work

There are three papers that were very influential for this thesis. The first one to mention is the FlowNet network proposed in [1] and [2]. Dosovitskiy et al. [1] presented a neural network architecture for estimating optical flow. This approach was then used by Li et al. [3] to design a neural network capable of 6D pose estimation. The presented approach follows the idea of using a rendered image of a three dimensional model to match an observed image. By iterative pose refinement of the three dimensional model before rendering, the images will look more alike in each step.
The approach presented in this thesis is also influenced by [4]. Rodriguez and Behnke [4] are using a method based on CPD and PCA to perform non-rigid registration. The predicted information gets processed to not only register two objects in a non-rigid fashion, but also to transfer grasping skills from a known model to a previously unseen observed object.
Finally, we will briefly cover the state of the art on optical flow estimation, shape completion and non-rigid registration.

2.1 Learning Optical Flow with Convolutional Networks

The approach proposed by [1] proved for the first time in 2015 that CNNs are capable to achieve competitive accuracy on optical flow estimation on existing datasets like Sintel [5] and Kitti [6] at frame rates of 5 to 10 frames per second. The results are encouraging to apply CNNs in other image processing tasks that require precise per-pixel localization and correspondence finding between two input images, since both are requirements for successful optical flow estimation. The training of the FlowNet networks is done end to end. Since it is known to be a difficult task to obtain ground truth flow estimation data on video sequences, the training of the network relies on a synthetic dataset called ”Flying Chairs”. This dataset consists of random background images of Flickr on which segmented images of Chairs are overposed. Even though the dataset is synthetic and the pictures are quite different than actual real world scenes, the network is able to

generalize on real world data. One of the proposed and tested network structures, namely "FlowNetSimple" is presented in Figure 2.1. Figure 2.2 shows the optical flow estimation of the FlowNet networks from [1] on image pairs of the Sintel dataset. Based on [1] Ilg et al. [2] proposed FLowNet2.0 in 2017. The methods

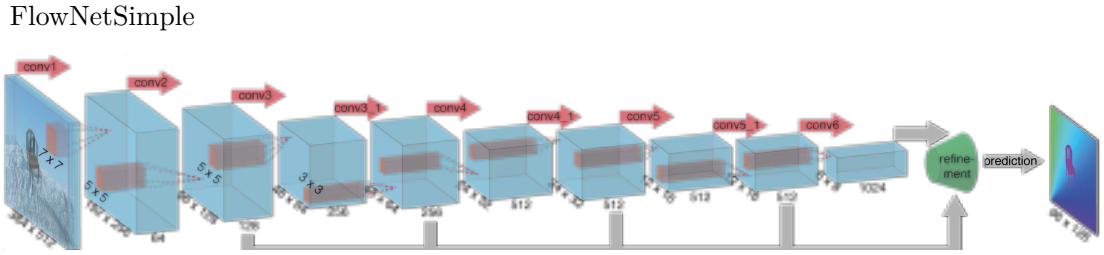

Figure 2.1: Network architecture of FlowNet Simple. Image taken from [1]

.

of [1] are tweaked in a way, that state of the art flow estimation is achieved by using CNNs. The FlyingChairs dataset gets adjusted to FlyingChairs3D. Instead of using two dimensional chair models now three dimensional chair models are used to reflect true motion and lighting effects on the models. The work shows, that a dataset schedule works best, starting with the old FlyingChairs dataset followed by fine tuning on FlyingChairs3D. This modification alone improved the results by up to 30%. A contribution introduced in [2] is the decaying schedule for the training of the network.
Another contribution is the proposal of a new stacked network structure. Stacked networks are used to mime the effect of an iterative flow estimating approach. After each network, one of the input images gets warped with the previously calculated flow to enable the next network to concentrate on improving the flow estimation. A schematic view of the complete architecture can be seen in Figure 2.3. One last contribution is a new CNN architecture specialized on small motions and therefore small values for optical flow. Additionally, a network architecture is presented, that is capable of merging the results from different optical flow estimation results. The entire FlowNet2.0 architecture is presented in Figure 2.3.
All those changes finally lead to better results as accomplished in [1]. Figure 2.4 shows example optical flow estimations on images from the Sintel dataset.

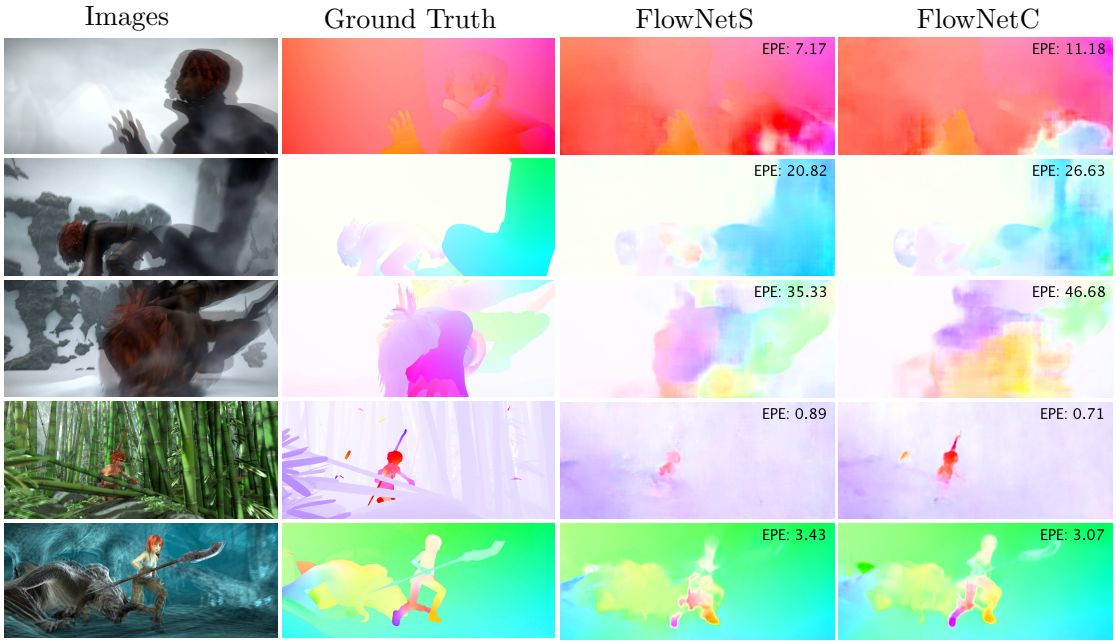

Figure 2.2: The overlayed input images on the left, followed by different colored visualizations of optical flows: Ground Truth, calculated by FlowNetS and calculated by FlowNetC. Image taken from [1].

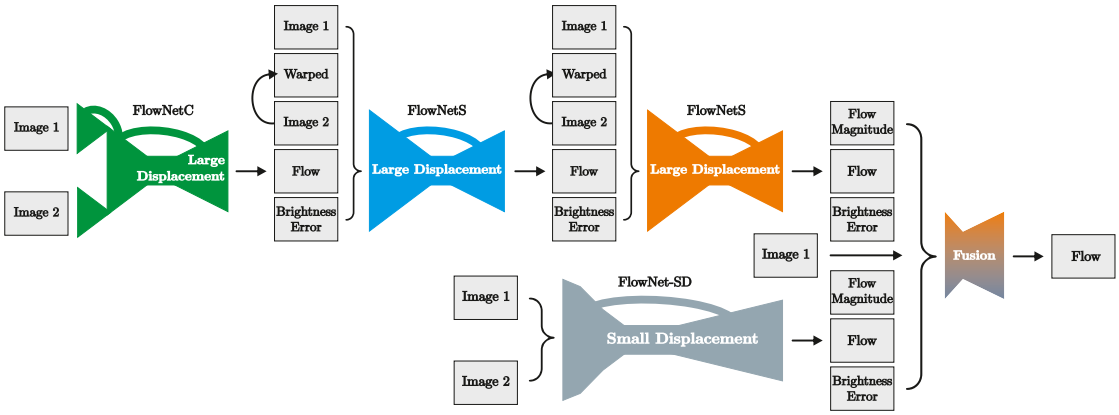

Figure 2.3: Schematic view of complete FlowNet2.0 architecture. Multiple FlowNet CNNs are combined. Curly braces means that the input gets concatenated. At the end, there is an additional new fusion network to calculate the final estimate. Image taken from [2].

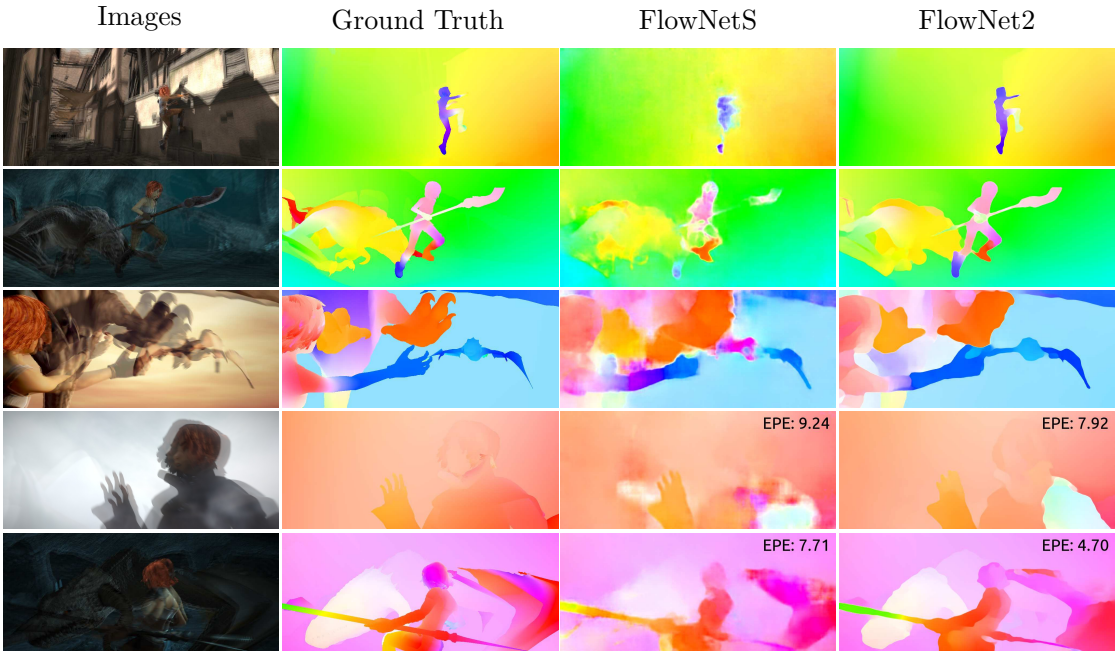

Figure 2.4: Example of different flow estimation results for the two overlayed images from the Sintel dataset on the left. The two most right images shows the results of the FlowNet2.0 network. Image taken from [2].

2.2 Learning 6D Pose Refinement with Convolutional Networks

Li et al. [3] used a CNN with a similar structure as FlowNetSimple from [1] to perform iterative matching for 6D pose estimation [3] (DeepIM). The motivation for this paper comes from the fact that recognizing the 6D pose, i.e., 3D location and 3D orientation of objects provides useful information for grasp and motion planning. Several recent techniques attack this problem by using 3D information of the object received by RGB-D cameras. Limitations of those cameras in comparison with RGB sensors are prevalent, such as frame rate, depth range, resolution and field of view.
A CNN for iterative pose refinement is introduced that automatically learns an internal refinement mechanism. The network is displayed in Figure 2.5. An 8 channel input consisting of the observed image, a bounding box for the observed object, a rendered image and a mask for the rendered object serves as input for the network. The network is used in an iterative approach, i.e., there are two alternating steps that are repeated. The first step is to render an image of a textured 3D model of the observed object in a pose that should be refined. The

second step is to feed the observed image together with the rendered image to the network to receive a small refinement for the previously applied pose. Afterwards, the first step is performed again with the refined pose estimation. The two steps are iterated, until the rendered and the observed image are highly similar.
The network architecture used for training differs from the one that is used for testing. For training, estimation of optical flow and mask prediction were added after deconvolutional layers of the network. This improves the stability of training process , but since deconvolutional layers are time consuming and the output is of no benefit for the pose estimation task, the network used for testing has no output of optical flow or mask images. The paper also covers an untangled representa-

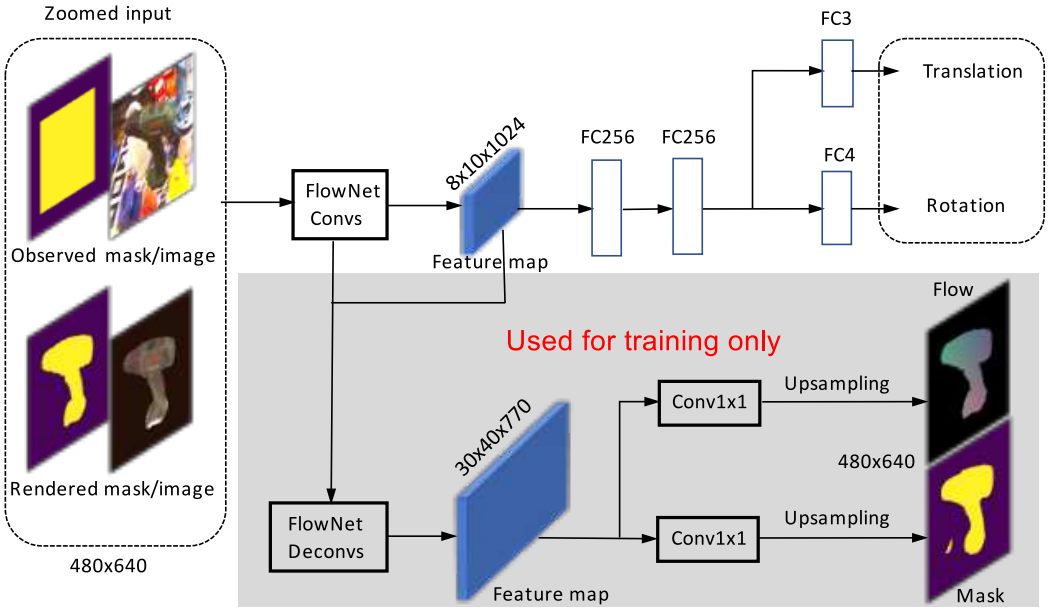

Figure 2.5: DeepIM network using the FlowNetSimple as backbone architecture. A pose estimation is predicted based on an observed and a rendered image. Image taken from [3].

tion of the SE(3) transformation between object poses. This representation is key to achieve accurate pose estimates. Experiments on several datasets, such that the LINEMOD [7] and the Occlusion dataset proved, that the DeepIM network improves over state of the art of RGB-only pose estimation in terms of accuracy.

2.3 Latent Space Non-Rigid Registration

Since objects within a category are often similar in shape and usage the approach from [8] and later on [4] uses CPD and PCA to define a search space for possible deformations from a known canonical model towards a novel observed object. The generated search space is called latent space. In this space a deformation from the canonical towards a novel object is found, which best matches the observed 3D points.
The method is divided in a learning and an inferring phase. In the learning phase, several three dimensional models of objects from the same category are used, where a category is defined as a class of objects with similar topology and extrinsic shape, like for example the category of drills. A canonical object is chosen heuristically and CPD is used, to generate a deformation from the canonical instance towards all the other instances. The deformation gets captured in a single matrix per training instance. By rewriting the matrices into vectors in a high dimensional vector space, it is possible to perform dimensionality reduction on the deformations vie PCA. This produces a low dimensional latent space. Because the dimensionality is low, it is possible to find a vector in the latent space that translates into a deformation that minimizes the difference between the deformed canonical and the observed 3D points, in a reasonable time. A schematic view of the learning process can be observed in Figure 2.6. The approach is proven to outperform CPD on shape

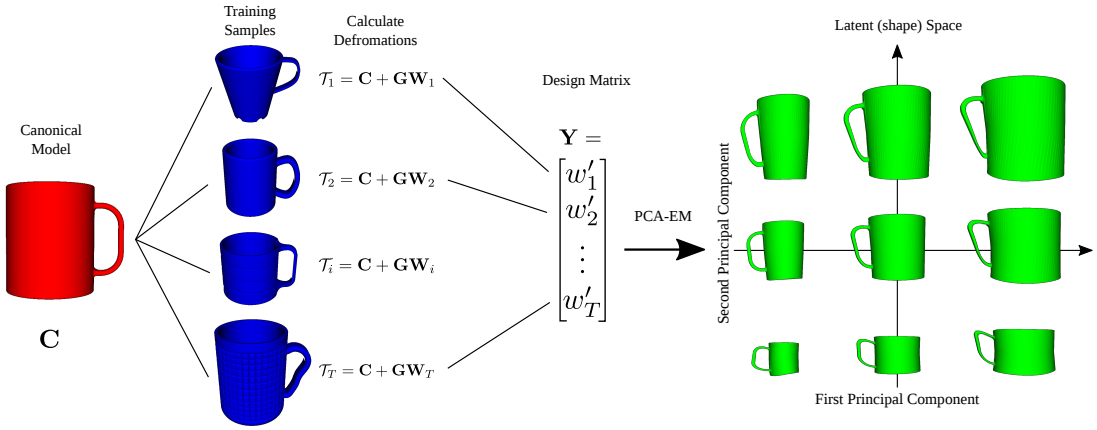

Figure 2.6: Latent space learning process. The deformations between the canonical model and the training instances are represented by $\mathbf{W}_i$. These are written into vectors and compressed by PCA to obtain the latent space. Image taken from [8].

completion tasks on partially observed objects. This comes from the fact, that deformations for similar shapes are already learned. This enables the approach to predict a possible shape even on parts where no information is available.

2.4 General Related Work

This section will cover over a wide field of related topics to this thesis. Starting by several approaches to perform optical flow estimation and motion extraction. The next part presents methods to complete 3D shapes. Finally, we introduce state-of-the-art approaches for non-rigid registration.

Optical Flow Estimation

Optical flow estimation using end-to-end training methods with CNNs was first presented in [1]. Following this approach Ranjan and Black [9] used a classical spatial-pyramid formulation for a deep learning concept to compute optical flow. The pyramid scheme takes care of the historically difficult problem of estimating large displacements within the optical flow. Another way to deal with large displacements in optical flow estimation via deep learning comes from [10]. A deep descriptor matching algorithm is blended with a variational approach for optical flow. Inspired from signal processing principles, Teney and Hebert [11] proposed a network architecture capable of end-to-end learning on motion extraction with a small number of training examples. Other approaches for estimating optical flow with CNNs are using patch matching together with Siamese networks [12] [13]. Two networks are used independently and in parallel to compute the descriptors of both images. To overcome the need of extensive training datasets, learning of optical flow estimation can be done in an unsupervised way. Unsupervised learning is applied by Ahmadi and Patras [14] for motion estimation. Another interpretation of learning optical flow unsupervised is given in [15].
Using neural networks for precise pixelwise estimation tasks often results in blurry estimations. Image denoising and image restoration in general can be applied in a post processing step to the network output. Chen and Pock [16] proposed a non-linear reaction diffusion model for image refinement with all parameters learned simultaneously through a loss based approach. In [2] the issue of blurry network output is solved by using a stacked network structure, to remove inaccuracy and refine the output in an incremental way.

3D Shape Completion

3D shape completion aims to infer unseen parts of an original shape given sparse or partial input observations. Radial basis functions are used to reconstruct smooth surfaces from point cloud data and to repair incomplete meshes [17]. Other approaches cast the reconstruction from oriented point clouds as a Poisson problem [18]. More recent methods use data-driven approaches for 3D completion tasks.

Choy et al. [19] use a recurrent neural network structure that learns mapping from images of objects to their underlying 3D shapes from a large collection of synthetic data. Volumetric deep neural networks and 3D shape synthesis are used in [20]. For a given partially observed shape, the network first infers a low-resolution output shape, that then gets passed through a 3D-encoder predictor network composed of 3D convolutional layers to infer the final shape. In [21], a singular depth image served as input to recover a complete 3D model. Objects from an exemplary database are used to transfer symmetries and surfaces. Furthermore, single view shape completion and reconstruction is approached by integrating deep generative models with adversarially learned shape priors [22] or by representing shapes as probability distributions of binary variables on a 3D voxel grid using a convolutional deep belief network [23]. Another way of representing a class of shapes are signed distance functions, that can be learned by neural networks [24].
Comparing our approach with the work presented in [22] and [19] is interesting, since both papers cover the topic of shape reconstruction from RGB images. While both approaches achieve good results when estimating shapes, they have no possibility to transfer knowledge over the estimated shape. Our approach meanwhile achieves a shape estimation by deforming a known object, such that it is possible to transfer knowledge.

Non-Rigid Registration

The variety of non-rigid registration methods differs mostly through the prior restrictions or through the regularization of the deformation that the points can undergo. The non-rigid registration surface matching problem can be formulated as a high order graph matching problem [25]. In a different approach, nearly isometric maps are blended together to obtain blended intrinsic maps between two surfaces [26]. Several methods for non-rigid registration rely on an isometry-invariant comparison of smooth surfaces [27] [28] [29]. Thin plate spines are used for for fitting high detailed meshes from human body scans [30] and the classical iterative closest points approach gets adjusted to work on non-rigid objects [31]. Additionally, to make use of expectation maximization algorithms the points of one point set can be understand as centroids of a gaussian mixture model [32].

3 Background

To explain our approach on solving the problem of non-rigid registration for novel objects from RGB images we need to explain two necessary methods. First we introduce the CPD method for point set registration. Afterwards we introduce PCA as a method to lower the dimension of data, while conserving as much information as possible.

3.1 Coherent Point Drift

In this section, the main concepts of the CPD [32] method will be introduced. Our approach of estimating a non-rigid registration is based on the non-rigid version of CPD. Since the rigid version of CPD is not used throughout this thesis, it will not be covered in this chapter. CPD is a method to perform point set registration for two given sets of points. Its goal is to assign a deformation field to one of two sets of points, such that the deformed set of points is similar to the other set of points.
For two sets of D-dimensional points $X = (x_1, ..., x_N)^T$ and $Y = (y_1, ..., y_M)^T$ CPD outputs a deformation field mapping Y towards X. For this purpose the points of Y are considered as the centroids of a Gaussian Mixture Model (GMM) and the points of X are considered as samples drawn from the GMM Y. Equal isotropic covariances σ^2 and equal membership probabilities $P(m) = \frac{1}{M}$ are used for all GMM components. The desired deformation field maximizes the probability of the points from X being drawn from the GMM Y, by moving the centroids. Therefore the point set registration problem is broken down to an expectation maximization (EM) problem.
At the same time, there are limitations on the possible movement of the centroids. Under the assumption, that centroids near each other move in a similar way, the calculated deformation field should be smooth. To achieve this CPD imposes a smoothness constraint in the form of motion coherence. This idea is based on Motion Coherence Theory [33].
For the non-rigid approach, a deformation field $\mathcal{T}(Y, v)$ is described by the initial

position of the points Y together with a displacement function v:

$$\mathcal{T}(Y, v) = Y + v(Y). \tag{3.1}$$

For any D-dimensional set of points $Z \in \mathbb{R}^{N \times D}$ the displacement function v is defined as:

$$v(Z) = \mathbf{G}(Y, Z) * \mathbf{W}. \tag{3.2}$$

Where $\mathbf{W} \in \mathbb{R}^{M \times D}$ is a weight matrix for the movement of the centroids. This can be interpreted as an offset or D-dimensional deformation vector that should be applied to the points of Y. The coherent movement is controlled by the Gaussian kernel matrix $\mathbf{G}(Y, Z)$. Given β, a free parameter, that controls the strength of interaction among different points, the matrix $\mathbf{G}(Y, Z)$ is defined element-wise as:

$$g_{ij} = \mathbf{G}(y_i, z_j) = \exp(-\frac{1}{2\beta^2} \left\| y_i - z_i \right\|^2). \tag{3.3}$$

For our convenience in the notion, $\mathbf{G}(Y, Y)$ will be denoted as $\mathbf{G}$. The in Equation (3.1) stated definition of a deformation field allows us to formulate an objective function $E(Y, \phi)$. Minimizing $E(Y, \phi)$ will maximize the likelihood that the points of X are drawn from the GMM Y. It is defined as:

$$E(Y, v) = -\sum_{n=1}^{N} log \sum_{m=1}^{M} \exp(-\frac{1}{2\sigma^2} \left\| x_i - \mathcal{T}(Y, v) \right\|^2) + \frac{\lambda}{2} \psi(v(Y)). \tag{3.4}$$

Whereas λ represents the trade-off between the precision of the maximum likelihood fit and a regularization term. The regularization is expressed with $\psi(v)$ and briefly summarized it is described by a norm in the Hilbert space $\mathbb{H}^m$ that decreases with the smoothness of $v(Y)$. The first part of Equation (3.4) penalizes the distance between the deformed points from Y and the points from X and the second part motivates coherent movement from the points of Y.
CPD uses an EM algorithm derived from [34] to minimize the energy function from Equation (3.4). The EM-algorithm is split in an estimation and a maximization step. Both steps build one iteration of the algorithm. Within the estimation-step a posterior probabilities matrix $\mathbf{P} \in \mathbb{R}^{M \times N}$ is calculated. It is defined element wise as:

$$p_{mn} = \frac{\exp(-\frac{1}{2\sigma^2} \left\| y_n - \mathcal{T}(y_m, v) \right\|^2)}{\sum_{k=1}^{M} \exp(-\frac{1}{2\sigma^2} \left\| y_n - \mathcal{T}(y_k, v) \right\|^2) + \frac{\omega}{1-\omega} \frac{(2\pi\sigma^2)^{0.5D}}{N}}. \tag{3.5}$$

The entry p_{mn} describes the a posteriori probability that the point with index n from X is drawn from the GMM with index m. The free parameter ω works as a weight in front of a regularization term, that takes care of noise in the input point set. The maximization-step of the EM-algorithm is used to estimate the matrix $\mathbf{W}$ from Equation (3.2). It is done by solving the following equation:

$$(\mathbf{G} + \lambda\sigma^2 d(\mathbf{P1})^{-1})\mathbf{W} = d(\mathbf{P1})^{-1}\mathbf{P}X - Y. \tag{3.6}$$

We will refer with $\mathbf{1}$ to a column vector of ones and with $d(.)$ to the function that maps a column vector to a quadratic diagonal matrix in the natural way.
Note that a deformation can be applied to a point set with knowledge about $\mathbf{G}$ and $\mathbf{W}$. While the $\mathbf{G}$ matrix only depends on the point set that is deformed, the $\mathbf{W}$ matrix depends on the target point set as well. This can be used to develop a method to learn deformation fields from a source point set to a target point set. As long as the source point set is well known, learning a deformation field is broken down to the task of estimating a fitting $\mathbf{W}$. We will use this idea later on for developing our non-rigid registration approach. An example for successful non-rigid registration via CPD can be observed in Figure 3.1

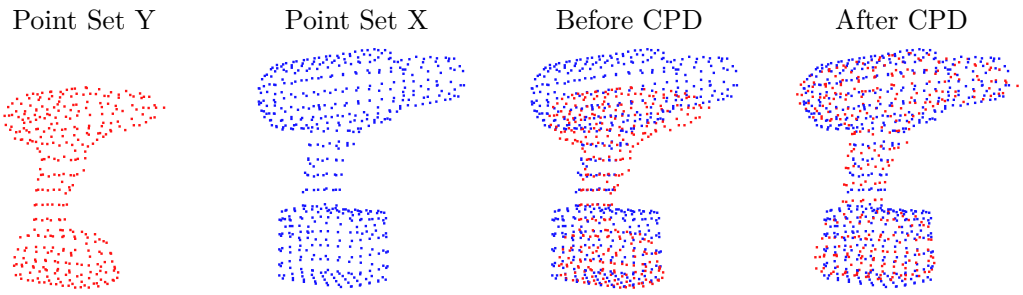

Figure 3.1: Example of a CPD execution on two point sets Y and X.

3.2 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure to map a set of high dimensional points to a set of points with a lower dimensionality. The goal is to reduce the complexity of the data without loosing too much information. This means that it should be possible to restore the original data points precisely. The main idea is, that the individual dimensions of the pre-PCA data are correlated to each other. PCA then calculates a change of basis to a lower dimensional vector

space, that is applied to the data, yielding in a post-PCA representation with less correlation between the decimated dimensions.
Let the pre-PCA data be represented by a vector of data points $X = (x_1^T, ..., x_n^T)^T$ with $x_i \in \mathbb{R}^m$. Each of the m dimensions will be denoted as d_i with $i \in (0, ..., m-1)$. For two given dimensions d_i, d_j we denote the sample covariance on X between them as $cov(d_i, d_j)$.
The first step for performing PCA is to normalize the data by subtracting the mean $\bar{x}$ from all points of the data set. $\bar{x}$ is a vector holding the mean value over X from each individual dimension. The next step is to calculate the covariance matrix $\mathbf{C}$. It is defined element-wise as:

$$c_{ij} = cov(d_i, d_j). \tag{3.7}$$

Afterwards the unit eigenvectors of $\mathbf{C}$ are calculated and sorted by the size of their according eigenvalues. Picking the first l of them leaves us with l eigenvectors $e_1, ..., e_l$ with $l << m$ and $e_i \in \mathbb{R}^m$. These vectors will serve as basis for a vector space. We will denote to this space as *latent space* throughout this thesis. All together the l eigenvectors build the matrix $\mathbf{L} \in \mathbb{R}^{m \times l}$ with one vector in each of the columns.

$$\mathbf{L} = \left(e_1 ... e_l\right). \tag{3.8}$$

$\mathbf{L}$ can be used to map data points between the original- and the latent space. By multiplying either a point in original coordinates with $\mathbf{L}$ from the right side, or by multiplying a point in latent coordinates with $\mathbf{L}^T$ from the right side. The mean $\bar{x}$ has to be either subtracted or added accordingly. Multiple data points can be converted simultaneously by putting them together in a matrix. For example, we can convert our original (pre-PCA) data points by rewriting the vector of vectors X as the matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$. We then obtain the latent space coordinates for all n points. They are stored in the according rows of the matrix $\mathbf{Y} \in \mathbb{R}^{n \times l}$ calculated as:

$$\mathbf{Y} = \mathbf{X} * \mathbf{L} + \bar{x}. \tag{3.9}$$

To go from points $\mathbf{Y}$ in the latent space to points $\mathbf{X}$ in the original space, we can perform the inverse operation:

$$\mathbf{X} = (\mathbf{Y} - \bar{x}) * \mathbf{L}^T. \tag{3.10}$$

Note, that we can transpose the Equations (3.9) and (3.10) if we multiply from the left with $\mathbf{L}$ or $\mathbf{L}^T$. Furthermore we note, that $\mathbf{L}$ is an orthogonal matrix and therefore $\mathbf{L}^T = \mathbf{L}^{-1}$.
Since the computation of the principle components is very costly, when it is done in an analytical way, we use the PCA Expectation Maximization (PCA - EM) algorithm [35]. This method is not only faster than the analytical approach, but it is also very stable in a numerical way. As usual for every expectation maximization algorithm the matrix $\mathbf{L}$ is found by alternating between an expectation and a maximization step. The expectation step is given by:

$$\mathbf{Y} = \mathbf{X}\mathbf{L}^T(\mathbf{L}\mathbf{L}^T)^{-1}. \tag{3.11}$$

And the maximization step by:

$$\mathbf{L} = (\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{Y}^T\mathbf{X}. \tag{3.12}$$

In these equations, $\mathbf{X} \in \mathbb{R}^{n\times m}$ denotes the original data vector in its matrix form and $\mathbf{Y} \in \mathbb{R}^{n\times l}$ is the matrix of the previously unknown conversion from $\mathbf{X}$ into latent coordinates, as it will be calculated by Equation (3.9). The columns of $\mathbf{L}$ will span the vector space of the first l principal components. By standard EM convergence proofs, it is shown that PCA - EM converges to a local maximum [34]. It can be shown additionally, that the only stable extrema is the global maximum [36] [37]. Therefore the algorithm will be optimal for a large enough number of iterations.
As a general insight from this section note, that the matrix $\mathbf{L}$ together with $\bar{x}$ stores a lot of information about the input dataset. Since the transfer from original into latent coordinates happens by a matrix multiplication, we can assume that points that are similar in the higher dimensional space are also similar in the lower dimensional one.

4 Methodology

This chapter describes how we solved the task of non-rigid registration from RGB data using deep learning. It describes how we use CPD to obtain ground truth deformations for our network to learn. The way these deformations are presented to a neural network will be described as well. Afterwards, the architecture of the neural network that is used to infer three dimensional deformations from a RGB image is described. The problem of occluded points that appears when an object is observed from a single view is addressed. We use PCA to generate a latent space for valid deformations of the object category and extend the deformations on the visible points by a least squares method. The last section is devoted to building a training dataset to learn object deformations. It is described how we used CPD to obtain additional training objects, and thus to augment the training dataset.

4.1 Deformation Representation

The task we aim to solve in this thesis is to calculate a deformation, that will deform a known 3D model into a novel object, observed on a RGB image. We refer to the model that is known and should be deformed as the canonical model. The canonical model consists of a textured and structured three dimensional mesh and an underlying point cloud. The mesh consists of m points or vertices. The vertices are connected by triangular cells. We infer the underlying point cloud for a given mesh by ray-casting from several viewpoints on a tessellated sphere. Afterwards, we apply a voxel grid filter that leaves us with n points given a defined resolution. An example of this process is shown in Figure 4.1. The underlying point cloud is described through a matrix where we assign each row to a 3D point. The matrix belonging to the underlying point cloud of the canonical model is referred as $\mathbf{C} \in \mathbb{R}^{n\times 3}$. By exchanging the coordinates of the point cloud points with the 3D coordinates of the vertices of the mesh, we define a matrix to represent the mesh, $\mathbf{C}_m \in \mathbb{R}^{m\times 3}$. The underlying point cloud is used to define a deformation on the whole mesh structure. To deform the mesh structure we have to move the vertices of the mesh without changing the topology of the cells. The matrix belonging to

the deformed mesh vertices is defined as $\mathbf{C}'_m \in \mathbb{R}^{m\times 3}$ and calculated as follows:

$$\mathbf{C}'_m = \mathbf{C}_m + \mathbf{G}(\mathbf{C}, \mathbf{C}_m) * \mathbf{W}(\mathbf{C}, \mathbf{O}). \tag{4.1}$$

In this equation, $\mathbf{W}(\mathbf{C}, \mathbf{O}) \in \mathbb{R}^{n\times 3}$ describes the offsets that should be applied to the points of the canonical point cloud $\mathbf{C}$ to deform it towards a point cloud, underlying an observed instance represented by $\mathbf{O} \in \mathbb{R}^{k\times 3}$. In this manner, each line of $\mathbf{W}(\mathbf{C}, \mathbf{O})$ can be assigned to a point of the canonical point cloud. The offsets $\mathbf{W}(\mathbf{C}, \mathbf{O})$ get multiplied by $\mathbf{G}(\mathbf{C}, \mathbf{C}_m) \in \mathbb{R}^{m\times n}$ to convert offsets given on the underlying point cloud into offsets on the vertices of the mesh. $\mathbf{G}(\mathbf{C}, \mathbf{C}_m)$ is calculated as described in Equation (3.3) and serves to ensure coherent movement of the vertices. Since the movement is smooth we use the old topology together with the new mesh vertices $\mathbf{C}'_m$. Note that $\mathbf{W}(\mathbf{C}, \mathbf{O})$ is the only part of the equation that depends on the observed instance. Therefore learning a deformation to apply to the canonical model is reduced to learning $\mathbf{W}(\mathbf{C}, \mathbf{O})$.

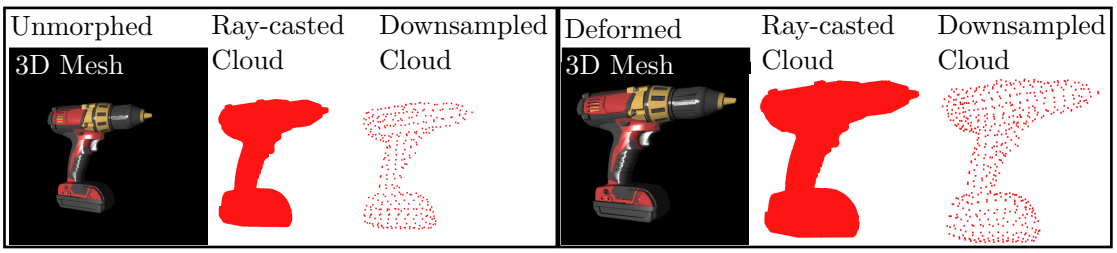

Figure 4.1: 3D mesh to low resolution point cloud. The mesh gets ray-casted and the resulting point cloud gets downsampled to a lower resolution. On the left side we see the canonical model in its original shape and on the right side we see the canonical model after it was deformed with an arbitrary deformation.

4.2 Rendering for Deep Learning

In this section, we briefly overview the definition of rendering or image synthesis. Since it is a major sub-topic of computer graphics, it is out of the scope of this thesis to give a depth analysis on rendering, but we can refer to further literature [38] [39] [40]. Rendering refers to the process of generating a photo-realistic image from a given 3D-model. The model contains information about the geometry and the texture of itself. The program to perform the rendering, called a renderer, then gets further information from the user. This includes a point of view and a source of light. The point of view determines the parts of the model that are seen on the picture and the source of light defines the lighting and the shadows.

The renderer outputs an image of the object on a black background. For most purposes it is useful to choose a real life image to substitute the background. Exemplary rendered images can be observed in Figure 5.2.
Todays rendering techniques are capable of producing convincing and photo-realistic rendering of highly complicated themes. In the concept of deep learning, rendering is used to make three dimensional data accessible to CNNs. Furthermore, rendering is used to generate synthetic datasets, that are on a high qualitative standard and are highly customizable for different applications. In this thesis we use rendering to render a frame of a canonical mesh to provide a RGB image as input for our neural network. Additionally, we train the network with a synthetic dataset, consisting of rendered views from several meshes from several viewpoints.

4.3 The Network Architecture

We approach the task of learning a deformation from two RGB images by applying a deep neural network based on the architecture introduced in [2]. For a given observed image, we assume to know the type of object we want to register. For this purpose, we define an object category as a set of objects with the same topology and a similar extrinsic shape. In other words, we assume that the category of the object of interest on the observed image is known. Furthermore, we assume that the position and rotation from the object of interest is known. These assumptions can be met by, e.g., commonly used methods for semantic segmentation [41] and pose estimation approaches [3]. With the given information we render a frame of a canonical three dimensional model from the known category in the known pose. At the same time, we produce a matching mask for the rendered image. We additionally compute a bounding box for the observed image. This leaves us with four images to serve as input for our network —the observed RGB image and mask, together with the rendered RGB image and mask. Depending on the larger mask, we zoom into the images to cut out unnecessary information. The zoomed images are then upsampled to a fixed size of 256×192 pixels. An example for the zooming process is shown in Figure 4.2. Note that both images are handled the same way for all operations to ensure that the aspect ratio between both objects does not change. We used bilinear upsampling to obtain the final images. The low resolution is necessary to compensate for the size of the network. We can use a low resolution representation without losing important information, since the deformation is defined on the low resolution point cloud of the canonical model.
We decided to use a network architecture based on [2] since we assume a connection between the optical flow among two objects and the offset that should be applied

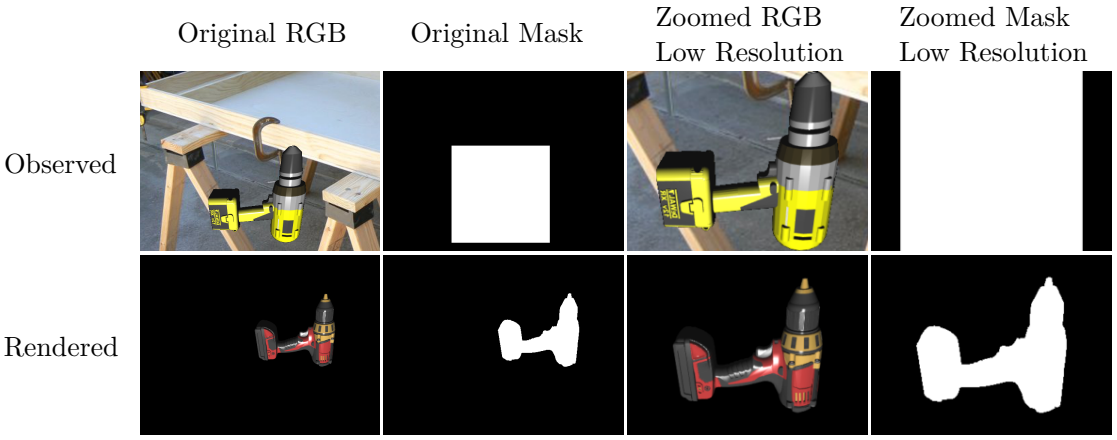

Figure 4.2: An observed image and the according rendered image are cropped and zoomed according to the biggest bounding box.

for a matching deformation. The mask can bee seen as a fourth channel additional to the RGB channels of the two input images. For our approach, the FlowNet2 architecture is changed in two major ways. First, we substitute every use of the original three channel input images with four channel images including the mask. Second, we change the network to stop estimating a two dimensional flow, but rather estimates a three dimensional offset. The final network architecture can be seen in Figure 4.3.

4.4 Image Coding for Deformation Estimation

To define a target for the network to learn, we use the deformation representation described in Equation (4.1) and keep a deformation in a three channel image. Each channel will represent one of the coordinate axis. We will use the term x-, y-, and z-channel throughout this thesis.
There are two ways of representing targets we tried within our work. The first way is to store the values from $\mathbf{W}(\mathbf{C}, \mathbf{O})$ directly. As stated above, each line in $\mathbf{W}(\mathbf{C}, \mathbf{O})$ belongs to exactly one point of the canonical point cloud. Given the zoomed rendered image of the canonical model we determine a closest point image of the same size (256×192 pixel). This image tells whether a pixel is part of the 3D model and its corresponding closest point of the canonical point cloud. The closest point image can now be used to create the target image for training. We search for the row in $\mathbf{W}(\mathbf{C}, \mathbf{O})$ that belongs to each pixel where the model is seen on. We then fill the first, the second and the third value of this row into the x-, y-, and z-channel of the target image on the same pixel. Pixels, where no model is

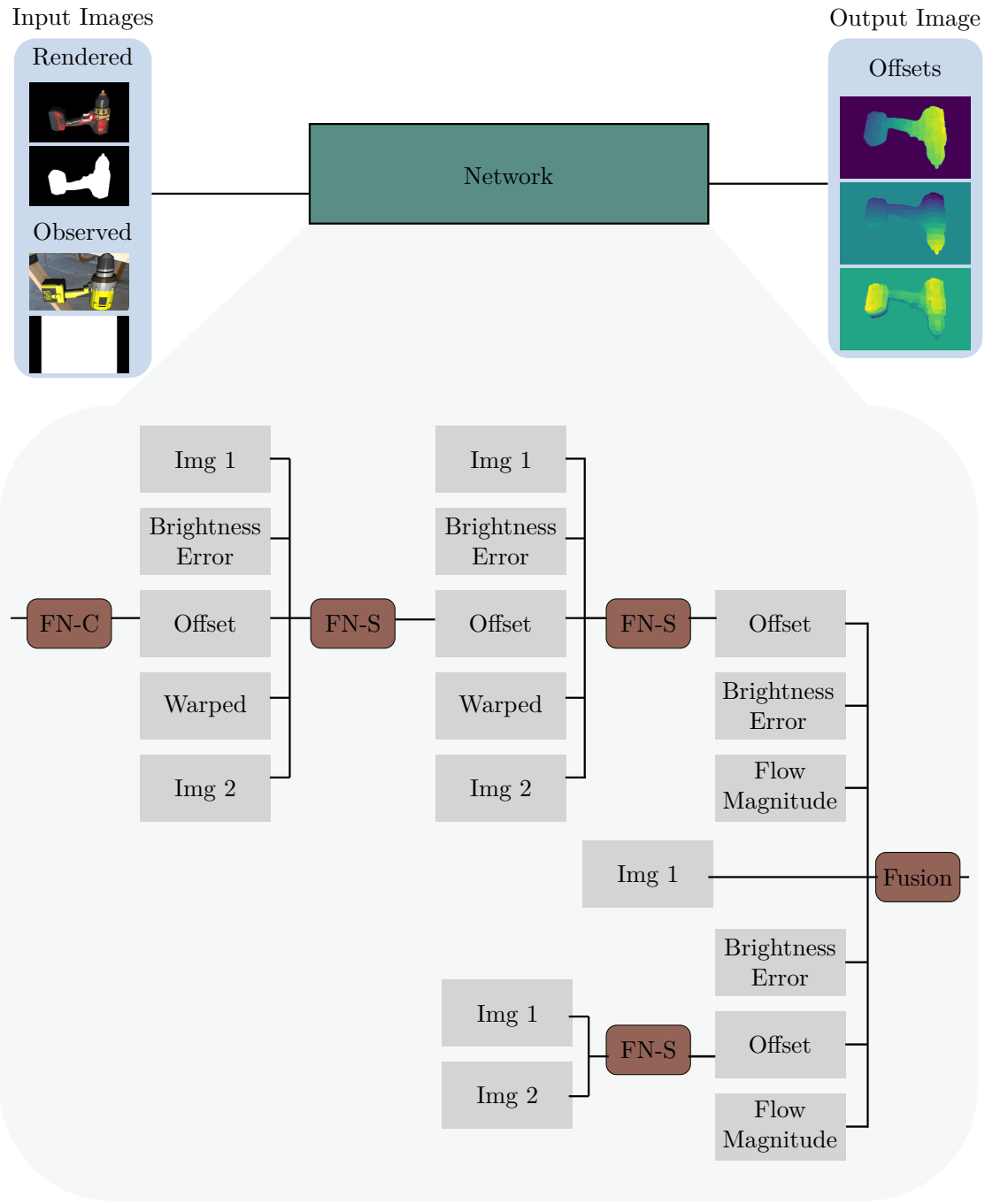

Figure 4.3: Starting from an 8 channel input, architectures derived from FlowNetSimple (FN-S) and FlowNetCorrelation (FN-C) are stacked, until receiving a three channel output, encoding an offset in x-, y- and z-direction. Img1 and Img2 refer to the concatenation of the rendered image and mask such as the observed image and mask. The Brightness error is the difference between the observed image and the rendered image warped with the previously estimated offsets (Warped). The output image is represented through heatmaps for each channel.

seen in the rendered and zoomed image are filled with zeros in the target image. This method holds the theoretical advantage that only necessary information gets stored, since $\mathbf{W}(\mathbf{C}, \mathbf{O})$ is the only part of Equation (4.1) that depends on the observed instance. However, the resulting pictures are not smooth and initial experiments showed, that the network had difficulties learning this representation in a sense of unstable behavior during training and bad generalization.
As a solution to this problem, we can use a coherence matrix $\mathbf{G}(\mathbf{C}, \mathbf{C})$ as calculated in Equation (2.3) to get a target picture that satisfies the condition of smoothness. We calculate the target offset matrix $\boldsymbol{\delta}(\mathbf{C}, \mathbf{O})$ depending on the canonical point cloud and the underlying point cloud of the observed model as follows:

$$\boldsymbol{\delta}(\mathbf{C}, \mathbf{O}) = \mathbf{G}(\mathbf{C}, \mathbf{C}) * \mathbf{W}(\mathbf{C}, \mathbf{O}). \tag{4.2}$$

We can now use $\boldsymbol{\delta}(\mathbf{C}, \mathbf{O})$ instead of $\mathbf{W}(\mathbf{C}, \mathbf{O})$ in the first method described above to obtain smoother images. A comparison between both representations can be seen in Figure 4.4.

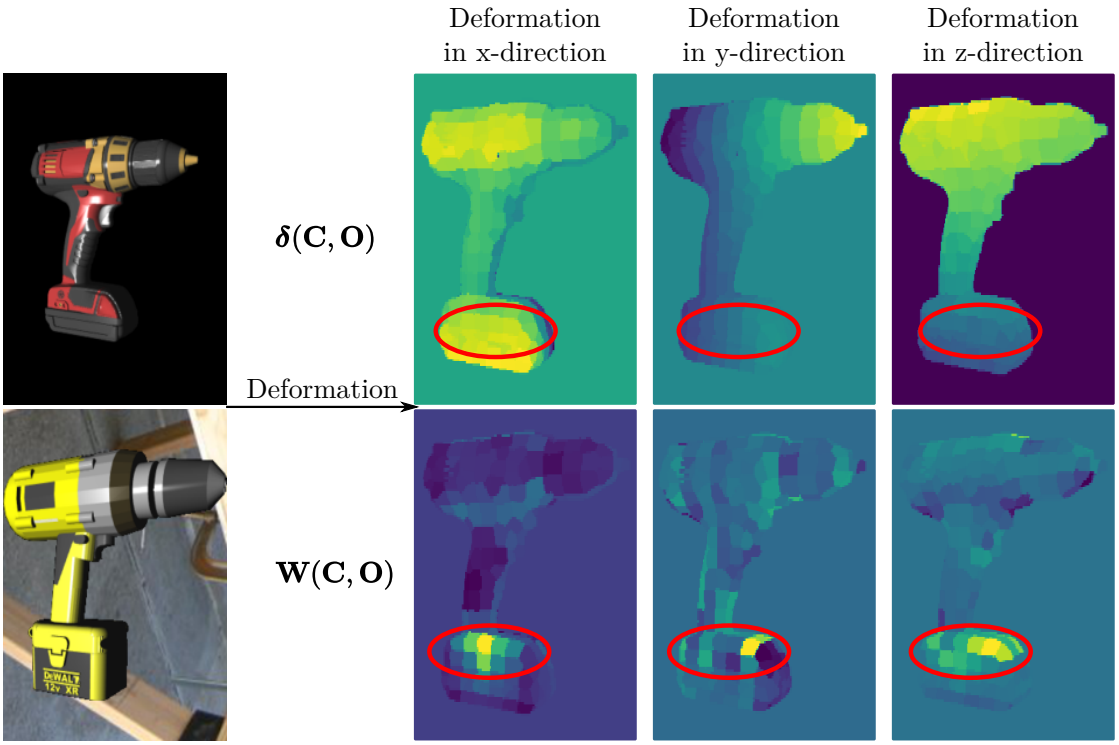

Figure 4.4: An observed and rendered image is the left, followed by the two different image codings for deformation estimation presented as heatmaps. The red circles highlight an area, that is irregular and difficult to learn using $\mathbf{W}(\mathbf{C}, \mathbf{O})$, but smooth when using $\boldsymbol{\delta}(\mathbf{C}, \mathbf{O})$.

4.5 The Problem of Occluded Parts

One problem to overcome when performing a three dimensional registration based on a two dimensional image is, that we are not able to fully observe the object. Because we use exactly one image as starting point for our deformation estimation, we need to find first an optimal deformation on the observable parts of the picture and then extend it to a deformation of all parts of the canonical model. Estimating optimal deformations on the observable parts, is handled by the network. Additionally we use multiple objects of an object category and calculate a latent space for inferring deformations of the occluded parts of the canonical model. We then calculate a deformation in the latent space, that is similar to the already estimated deformation on the observed parts but with additional information for the hidden parts as well. A similar procedure has proven to work well with partially observed three dimensional data [8] [4].
We start generating a latent space, by choosing k different objects from the same object category, that capture the geometric variability of the category. The point cloud of each object is denoted by $\mathbf{T}_i$ with $i \in (0, ..., k-1)$. The point cloud could be either directly given or could be generated with ray-casting from a three dimensional mesh, similar as in Section 4.1. To control that the deformations will only capture the different shape of the objects, we align all objects to the same coordinate frame. So for the drills category, for example, the middle of the base point is located at the origin of the coordinate system, the handle is aligned with the z-axis and the tip points in the direction of the y-axis.
We now use CPD to calculate k deformations, deforming the canonical point cloud towards each of the k category objects. The transformation of the canonical point cloud towards object i will be denoted as $\mathcal{T}_i$:

$$\mathcal{T}_i(\mathbf{C}, \mathbf{W}_i) = \mathbf{C} + \mathbf{G}(\mathbf{C}, \mathbf{C}) * \mathbf{W}_i. \tag{4.3}$$

Section 2.1 explains how we obtain $\mathbf{G}(\mathbf{C}, \mathbf{C})$ and $\mathbf{W}_i$. Note, that the only part of the representation, that is not relying on the canonical model and therefore capturing the uniqueness of the deformation is $\mathbf{W}_i$. In addition, we observe that the dimensionality of each $\mathbf{W}_i$ is the same. This allows us to define a feature vector w_i for each of the k objects. w_i is obtained by converting the matrix $\mathbf{W}_i \in \mathbb{R}^{n\times 3}$ into a row vector $w_i \in \mathbb{R}^{1\times 3n}$. Using each w_i as a data point we perform PCA as described in Section 2.2. This gives us the matrix $\mathbf{L} \in \mathbb{R}^{l\times 3n}$ of principle components. We can now transform any point given by a vector $x \in \mathbb{R}^l$ in the

latent space into a feature vector $w_x \in \mathbb{R}^{3n}$. This is given by the Equation:

$$w_x = \mathbf{L} * x + \bar{w}. \tag{4.4}$$

Where $\bar{w}$ is the mean of all feature vectors. This equation shows that the canonical matrix $\mathbf{C}$ together with the principle components $\mathbf{L}$ represents the deformation model for an object category. We observe, that since PCA normalizes the data points, the feature vectors w_i are corresponding to points in the latent space near the origin.
The latent space can now be used to obtain a deformation feature vector from network output. As described the network output stores information about the offsets, that should be applied to the canonical point cloud. We need to assign for every pixel of the zoomed rendered image the point in the underlying canonical point cloud, that is closest to the part of the 3D model that is seen in the pixel. Since the resolution of the images is higher than the resolution of the point cloud, multiple pixel are assigned to the same point. For every of the three channels of the network output, we take the mean of all pixels that are assigned to the same point of the point cloud. This results in a pointwise offset, that describes how the canonical points have to be moved, to match the shape of the observed object. Since we used only a single view image for this operation, there are points left, we do not have offset information about. These are the occluded points, while we denote to the points we have offset information on as visible points. For the occluded points we assume an initial offset of zero. Using all offset values, we build the matrix $\boldsymbol{\delta}_{vis} \in \mathbb{R}^{n\times 3}$ by filling row i with the x, y and z coordinate of the calculated offset of point i. This matrix contains a lot of zeros. We will use $\boldsymbol{\delta}_{vis}$ as starting point to obtain a full deformation of the canonical model.
The task of obtaining a full deformation is equal to the task of calculating a point x in the latent space. This holds, since a point in the latent space can be translated into a feature vector from a deformation and therefore in an inferred offset matrix $\boldsymbol{\delta}_{inf}$. This can be written as:

$$\boldsymbol{\delta}_{inf} = \tilde{\mathbf{G}}(\mathbf{C}, \mathbf{C}) * (\mathbf{L} * x + \bar{w}). \tag{4.5}$$

Where $\tilde{\mathbf{G}}(\mathbf{C}, \mathbf{C}) \in \mathbb{R}^{3n\times 3n}$ is a rearranged form of $\mathbf{G}(\mathbf{C}, \mathbf{C}) \in \mathbb{R}^{n\times n}$ with additional zeros. $\tilde{\mathbf{G}}(\mathbf{C}, \mathbf{C})$ is build in a way, that the offset obtained by rearranging vector $\tilde{\mathbf{G}}(\mathbf{C}, \mathbf{C}) * w_x$ to an offset matrix has the same result as rearranging the feature vector w_x to a matrix $\mathbf{W}_x$ and multiplying $\mathbf{G}(\mathbf{C}, \mathbf{C}) * \mathbf{W}_x$ to directly obtain the offset in matrix form. The goal is to minimize the loss on indices belonging to the

visible points between $\boldsymbol{\delta}_{vis}$ and $\boldsymbol{\delta}_{inf}$ defined as:

$$L(\boldsymbol{\delta}_{vis}, \boldsymbol{\delta}_{inf}) = \sum_{v \in i_{vis}} \|\boldsymbol{\delta}_{vis}(v,.) - \boldsymbol{\delta}_{inf}(v,.)\|^2 . \tag{4.6}$$

Where i_{vis} describes the set of indices belonging to visible points and $\boldsymbol{\delta}(v,.)$ describes the v-th line of the matrix $\boldsymbol{\delta}$. We can rewrite this minimization problem as a simple least squares problem. Let n_v denote the amount of visible points. We then define $\overline{\boldsymbol{\delta}}_{vis} \in \mathbb{R}^{1\times 3n_v}$ as $\boldsymbol{\delta}_{vis}$ after we removed every row containing zeros and rearranged it to a row vector. This means we removed every row with an index belonging to an occluded point. To overcome the problem of different dimensions from $\overline{\boldsymbol{\delta}}_{vis}$ and $\boldsymbol{\delta}_{inf}$ we introduce a matrix $\mathbf{D} \in \mathbb{R}^{n_v \times n}$. The matrix $\mathbf{D}$ is filled with zeros and ones in a way, that multiplication with $\mathbf{D}$ will remove exactly the same indices from $\boldsymbol{\delta}_{inf}$ as the ones that were removed from $\boldsymbol{\delta}_{vis}$. Putting all information together, minimizing Equation (4.6) results in the same as minimizing the following:

$$L(\overline{\boldsymbol{\delta}}_{vis}, x) = \left\| \overline{\boldsymbol{\delta}}_{vis} - \mathbf{D}(\tilde{\mathbf{G}}(\mathbf{C},\mathbf{C}) * (\mathbf{L} * x + \bar{w})) \right\|^2 , \tag{4.7}$$

which can be rewritten as:

$$L(\overline{\boldsymbol{\delta}}_{vis}, x) = \left\| \overline{\boldsymbol{\delta}}_{vis} - \mathbf{D}\tilde{\mathbf{G}}(\mathbf{C},\mathbf{C}) * \bar{w} - \mathbf{D}\tilde{\mathbf{G}}(\mathbf{C},\mathbf{C}) * \mathbf{L} * x \right\|^2 . \tag{4.8}$$

Defining:

$$\mathbf{A} := (\mathbf{D}\tilde{\mathbf{G}}(\mathbf{C},\mathbf{C}) * \mathbf{L}), \tag{4.9}$$

and

$$\mathbf{B} := (\overline{\boldsymbol{\delta}}_{vis} - \mathbf{D}\tilde{\mathbf{G}}(\mathbf{C},\mathbf{C}) * \bar{w}), \tag{4.10}$$

we write the minimization of Equation (4.8) as a linear least squares problem:

$$\arg\min_x L(\overline{\boldsymbol{\delta}}_{vis}, x) = \arg\min_x \|\mathbf{A}x - \mathbf{B}\|^2 . \tag{4.11}$$

Solving Equation (4.11) with an off-the-shelf linear solver gives us a point x_* in the latent space. We deform x_* into the feature vector w_{x_*} by Equation (4.4). By rearranging w_{x_*} we obtain a matrix $\mathbf{W}_{x_*}$. Finally by using Equation (4.1) we receive a deformation of the whole canonical model and the non-rigid registration process is finished. A schematic overview of the process can be seen in Figure 4.5.

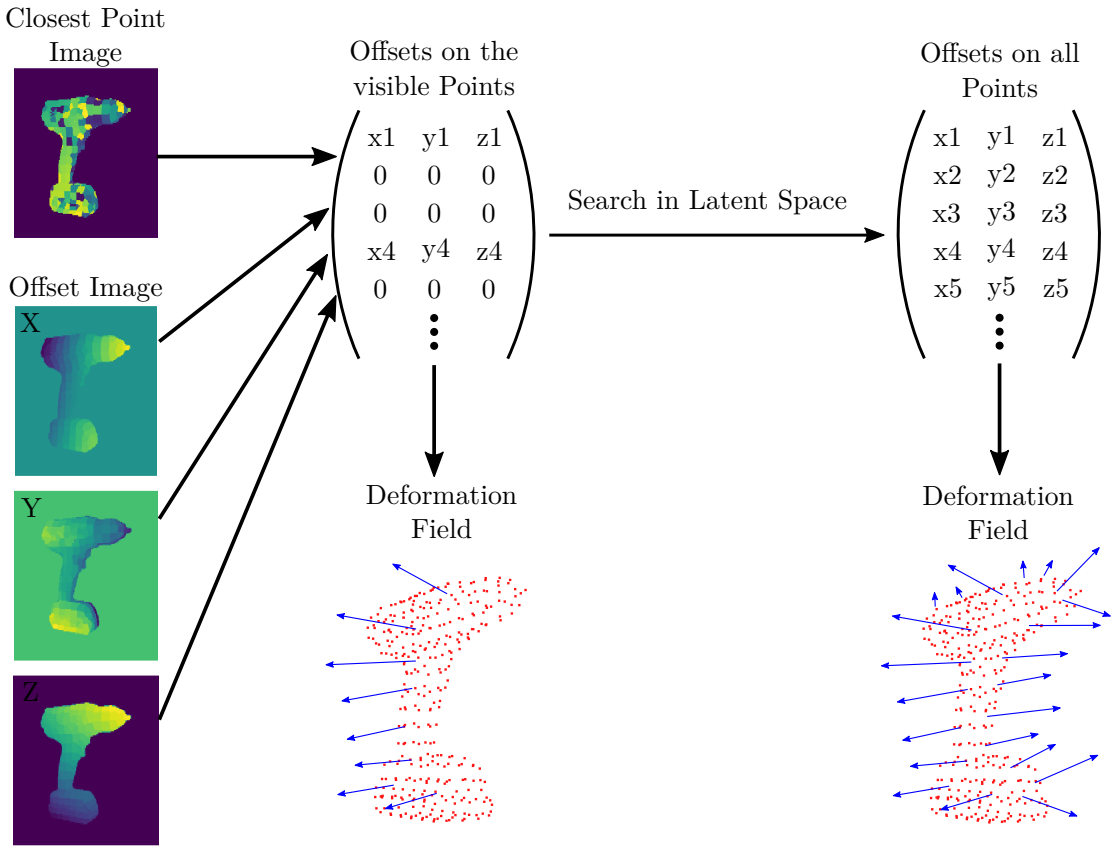

Figure 4.5: The network output is used to infer offsets for the visible points. The resulting deformation field gets expanded to all points with help of the latent space of the object category

4.6 Iterative Matching

Estimating a deformation that will match two objects from a single view RGB image is a challenging task for neural networks to learn. One idea to to improve the accuracy is to split the work, that has to be done by the network in smaller sections. To achieve this, we decided to try an iterative approach for non-rigid registration from RGB images. We already observed, that the deformation that should be applied to the canonical instance is described by the matrix $\mathbf{W}(\mathbf{C}, \mathbf{O})$ of Equation (4.1). We aim now on improving the estimation of $\mathbf{W}(\mathbf{C}, \mathbf{O})$ in a stepwise approach. We initialize the deformation field to be zero and therefore $\mathbf{W}_0(\mathbf{C}, \mathbf{O}) = 0$, additionally we will track $\tilde{\boldsymbol{\delta}}_{vis}$ as the accumulated offset of the

observed points within the registration process.
We define one iteration i for a given observed image and a previously computed $\mathbf{W}_{i-1}(\mathbf{C}, \mathbf{O})$. The first step is to render a frame of the canonical model in the position of the observed object. The canonical model is deformed before rendering using $\mathbf{W}_{i-1}(\mathbf{C}, \mathbf{O})$ and Equation (4.1). Now both images are zoomed together with their corresponding masks and presented to the network. The network computes an offset on the visible points $\boldsymbol{\delta}_{vis,i}$ that is used to update $\tilde{\boldsymbol{\delta}}_{vis}$ in the following way:

$$\tilde{\boldsymbol{\delta}}_{vis} = \frac{1}{i} * \boldsymbol{\delta}_{vis,i} + \tilde{\boldsymbol{\delta}}_{vis}. \tag{4.12}$$

We use the fraction of $\boldsymbol{\delta}_{vis,i}$ at the beginning of this equation considering, that the approximation task gets harder for the network. Later, the update $\tilde{\boldsymbol{\delta}}_{vis}$ is used to infer $\mathbf{W}_i(\mathbf{C}, \mathbf{O})$ as described in Section 4.5 and we finished the iteration. A schematic overview of one iteration can be observed in Figure 4.6.
This approach follows the idea, that if the network predicts a deformation that will lead to an erroneous match of the deformed canonical and the observed model, this will be reflected by the images that will serve as network input in the next iteration. This means, that our method has a chance to detect this errors and adjust the deformation accordingly. We assume that the network is able to generalize on estimating deformations for the deformed canonical model, since the size of the canonical model in the rendered frame is always relative to the observed model, because of the way we handle the zooming process. So the training dataset will include different versions of the canonical model.

4.7 Dataset generation

Since we have very detailed requirements for a dataset that will be used to train our network and there is no dataset available to meet the requirements, we had to build our own. The requirements for a given object category include:

- A 3D model of a canonical object from the category
- A large set of images including varying objects from the category with different textures
- 3D data for each object seen on the images to generate a ground truth non-rigid registration and rendered images
- Different points of view to render the 3D objects

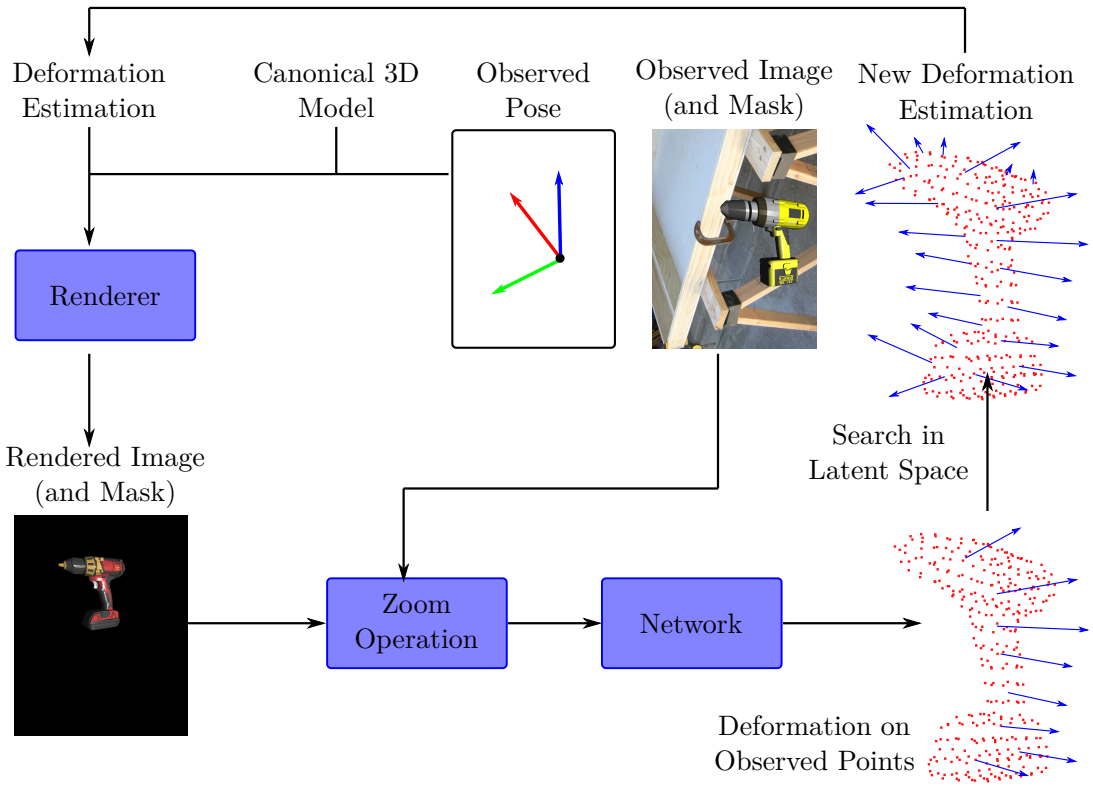

Figure 4.6: A deformation estimation (zeros in the first step) is used to render the canonical model in the observed position. The result gets zoomed in, together with the observed image and the regarding masks. The network processes the images to a deformation on the observed parts and the latent space is used to determine a new deformation estimation.

To build the dataset we collect textured 3D data with different objects from the same object category. First, we choose a canonical model heuristically and separate the objects into the canonical model and k testing models. The collection of 3D data can be the same as the one that is used to solve the problem of occluded points and build a latent space for the object category (Section 4.4). The main idea is to calculate ground truth deformations from the canonical model towards all the other models. Then, we change the viewpoint on each model according to a tessellated sphere and render an image. We then use the method described in Section 4.3. to generate a ground truth target pictures for the network to learn.
To calculate all the deformations we rely on CPD and underlying point clouds generated by ray-casting. Since good quality 3D textured models are sparse, we interpolate models that are in between the canonical and the other observed models. To achieve this, $\mathcal{T}(\mathbf{C}, \mathbf{W}_i)$ is calculated for all the testing models, as described in Equation (4.3). By using Equation (4.1), a deformation from the canonical model towards the observed model is found. Note that we can not use this deformation for interpolating between both instances directly. Doing this means, that all drills seen on the training images would share the texture of the canonical model. This is problematic since the network would would not be able to generalize to unseen models. To overcome this problem, we define an inverse deformation based on the underlying point cloud of the observed training instance $\mathbf{T}_i$ and the previously calculated matrix $\mathbf{W}_i$:

$$\mathcal{T}^{-1}(\mathbf{T}_i, \mathbf{W}_i) = \mathbf{T}_i + \mathbf{G}(\mathbf{T}_i, \mathbf{C}) * (-\mathbf{W}_i). \tag{4.13}$$

Equation (4.13) is capable of deforming the observed training point cloud $\mathbf{T}_i$ towards the canonical point cloud $\mathbf{C}$. To deform the vertices $\mathbf{T}_{m,i}$ of the observed training model into $\mathbf{T}'_{m,i}$ we do the following:

$$\mathbf{T}'_{m,i} = \mathbf{T}_{m,i} + \mathbf{G}(\mathbf{T}_{m,i}, \mathbf{C}) * (-\mathbf{W}_i). \tag{4.14}$$

Finally, by adding an interpolation factor ρ we obtain:

$$\mathbf{T}'_{m,i}(\rho) = \mathbf{T}_{m,i} + \mathbf{G}(\mathbf{T}_{m,i}, \mathbf{C}) * (-\rho * \mathbf{W}_i). \tag{4.15}$$

Equation (4.15) enables us to interpolate between every observed testing model and the canonical model, to generate images of the interpolated model with the texture from the observed testing models. We therefore encourage that the network is able to learn a lot of different shapes and textures. An example for the interpolation process is visualized in Figure 4.7. To generate a training dataset we produce all four images needed as network input directly. For a fixed observed testing

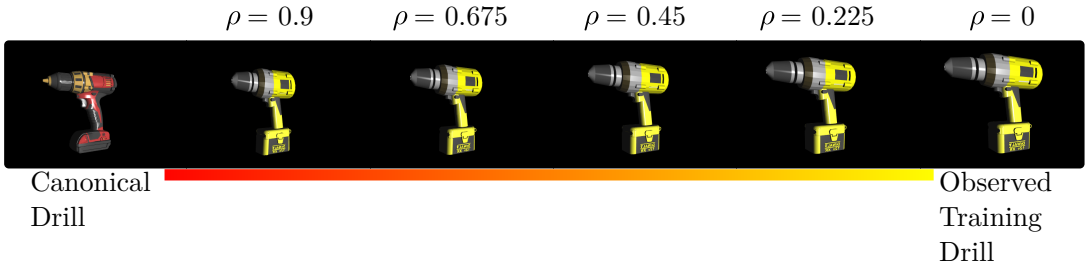

Figure 4.7: Interpolation process from the observed instance on the right towards the canonical instance on the left.

model i and a fixed interpolation constant ρ, we morph the observed testing three dimensional data according to Equation (4.15). We now select a point of view and render an image of this deformed observed testing model. At the same time, we render an image of the canonical model from the same point of view. This step can be done in real time but pre-rendering saves time for the training process. We generate a bounding box for the image of the observed model and a mask for the rendered canonical image. This four images are now cropped to the same size, such that the bigger one of both models will just fit in the cropped image. All four image are then resized to the targeted input size of 256 times 192 pixel by using a bilinear upsampling algorithm. Together with the canonical image we obtain a three channel position image from the renderer. This image holds in each pixel the information about the x-, y-, and z-coordinate of the part of the model, that is observed in the according pixel of the regular rendered image. The position image is cropped and upsampled in the same way as the other four images and it is the basis for calculating the closest point image that is needed in Section 4.4.
The ground truth target data for the network is generated by using the procedure presented in Section 4.4. However, since we interpolated between models we need to modify Equation 4.2:

$$\boldsymbol{\delta}(\mathbf{C}, \mathbf{T}_i) = \mathbf{G}(\mathbf{C}, \mathbf{C}) * (1 - \rho) * \mathbf{W}_i \tag{4.16}$$

We executed the described steps for a collection of three dimensional models of drills. Iterating through 16 training instances, 5 interpolation constants (as seen in Figure 4.7) and 74 points of view we obtain a dataset with 5920 entries.

5 Results

5.1 Experimental Setup

We conducted experiments on a synthetic dataset consisting of rendered images from drills build as described in Section 4.7. The complete collection of 3D mesh models is shown in Figure 5.2. We choose the drill in the top left corner of Figure 5.2 as the canonical drill, as it showed good results, when being deformed to match the shape of the other drills. Drill 04 and drill 12 are chosen as testing instances. The two drills represent two different difficulties for non-rigid registration: a case where the registration needs to focus on small and precise alignments (04) and a case, where the offsets that need to be applied to the canonical model are big (12). Both models are not used during training to be able to check the performance of the network on novel objects. Additionally we decide to use only the interpolated versions of drill 03, that are received from interpolating with $\rho = 0.9$, $\rho = 0.675$ and $\rho = 0.45$ for training. This means that the original shape of drill 03 is unknown to the network, but it had the chance to learn the texture of the drill. The remaining entries of the dataset are split up randomly. 90% are used for actual training and 10% are used to evaluate and monitor the error between desired ground truth and actual network output. We used the standard L_2 loss for evaluating the state of the network. The drills seen in Figure 5.2 are also used to build the latent space as described in Section 4.5. We performed this operation without drill 03, 04 and 12, since this objects should be used for testing. To build our latent space, we choose parameter l from Equation (3.8) from experience to $l = 5$.
To obtain ground truth deformations we used CPD and chose the parameters λ from Equation (3.4) and β from Equation (3.3) to be 2.0. Since we have input data without noise, the parameter ω from Equation (3.5) is chosen to be zero. We trained on three graphic cards with a batch size of 64 each, so in total we have an effective batch size of 192. The learning rate is decreased stepwise, starting from $5.e^{-5}$ it is divided by two every 660 epochs, until we reach a final learning rate of $1.e^{-6}$. A visualization of the learning rate together with the L_2 loss between ground truth and network output are shown in Figure 5.1. The training is done for approximately 8200 epochs, since the network stopped learning, i.e., the loss did not decrease further. Initial tests on training the network showed us, that it

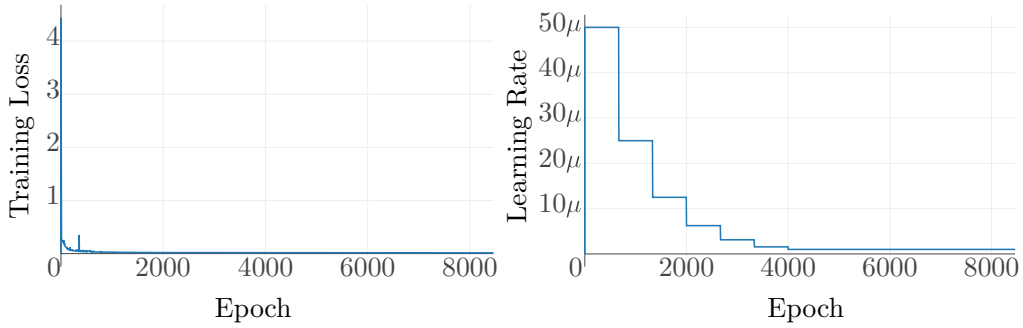

Figure 5.1: Monitoring of loss and learning rate during training.

was necessary to scale the ground truth deformation representation by the factor 100, to have a more clear boundary between foreground and background.

5.2 Experimental Results

To have a measure how good an observed shape is fitted by the deformation of the canonical model, we define an error function on two point clouds $\mathbf{C} \in \mathbb{R}^{n\times 3}$ and $\mathbf{T} \in \mathbb{R}^{m\times 3}$:

$$E(\mathbf{T}, \mathbf{C}) = \frac{1}{m} * \sum_{i=0}^{m-1} \min_{j} \|\mathbf{T}(i,.) - \mathbf{C}(j,.)\|^2 \tag{5.1}$$

The error can be described as the mean distance between the points of the observed model to the respective closest point of the deformed canonical point cloud. So the error will increase for every part of the observed model, that is not matched properly.
We compare our results to the latent space approach from [4], CPD [32] and the canonical model when no deformation is applied. For the approach from [4], CPD is parameterized in the same manner as for computing our ground truth deformations. For the latent space, we also choose the dimensionality to be five. The CPD to compare our results is computed with the same parameters, as well. To generate testing images, we rendered the testing drills from 74 points of view on a tessellated sphere and produced a point cloud to represent the observed parts of the drills. We produced point clouds comparable to what a RGB-D sensor would observe from the same point of view, our RGB image is taken from.
To evaluate how good the deformation representation is working and how reliable we are able to solve the problem of occluded parts, we calculate the deformation

Figure 5.2: Two rendered images from a side and a top view, followed by a visualization of the underlying point cloud for all meshes used to produce the synthetic dataset.

that results from the ground truth deformation estimation that represents the desired output of the network. This enables us to investigate a upper bound for the results of our approach and to evaluate how good our network is performing, since we have a side by side comparison between the results achieved with our network and results that are theoretically possible. We generated boxplots to compare the results for all 74 points of view, which are shown in Figure 5.3, but we had to remove one point of view for visualization purposes, since the error from the result of CPD was too high. An example for a testing image taken from this point of view is number 5 from Figure 5.4. It is the top down view on a drill, that has to few information for CPD to calculate a good deformation, since CPD does not have any knowledge of the object category. For drills whose texture is known (example 1 from Figure 5.4) and the drill that needs small but precise adjustments (example 2-4 from Figure 5.4) our approach is in general more precise than CPD and is not as susceptible against missing information, as the mean and the variance are lower for our approach than for CPD in this cases. For the bigger novel drill (example 5-6 from Figure 5.4) we see that our network had difficulties to estimate the deformation that should be applied to the top part of the drill. This has an impact on the results seen in Figure 5.4, as the mean and the variance is higher for our approach in this case. We also see for all three graphs from Figure 5.3, that the latent space approach from [4] performs best overall from the three approaches. This is the expected behavior, since the approach from [4] is proven to outperform CPD on partial observations and had in comparison to our approach access to three dimensional data.
The experiments showed our approach to be slower than the comparable approaches with a time consumption of circa 20 seconds per iteration. This comes due to the size of the network such as the rendering and zooming process that needs time. The time per iteration can be optimized in the future, for example with reducing the size of the network and using a faster renderer.
Figure 5.5 shows a comparison between the desired ground truth network output and the actual network output for three randomly chosen examples. The network estimation is not good enough for some instances, mostly when the offsets that should be estimated are rather big. This implies, that the network is the current bottleneck of our approach. The second boxplot of all three graphs in Figure 5.3, shows what is the upper bound for our approach, when actual network output and ground truth targets are the same. To compensate for errors in the estimation from the network, we used the iterative matching approach, which is visualized in Figure 5.6. The iterative approach improved the registration, especially for high ground truth offsets.

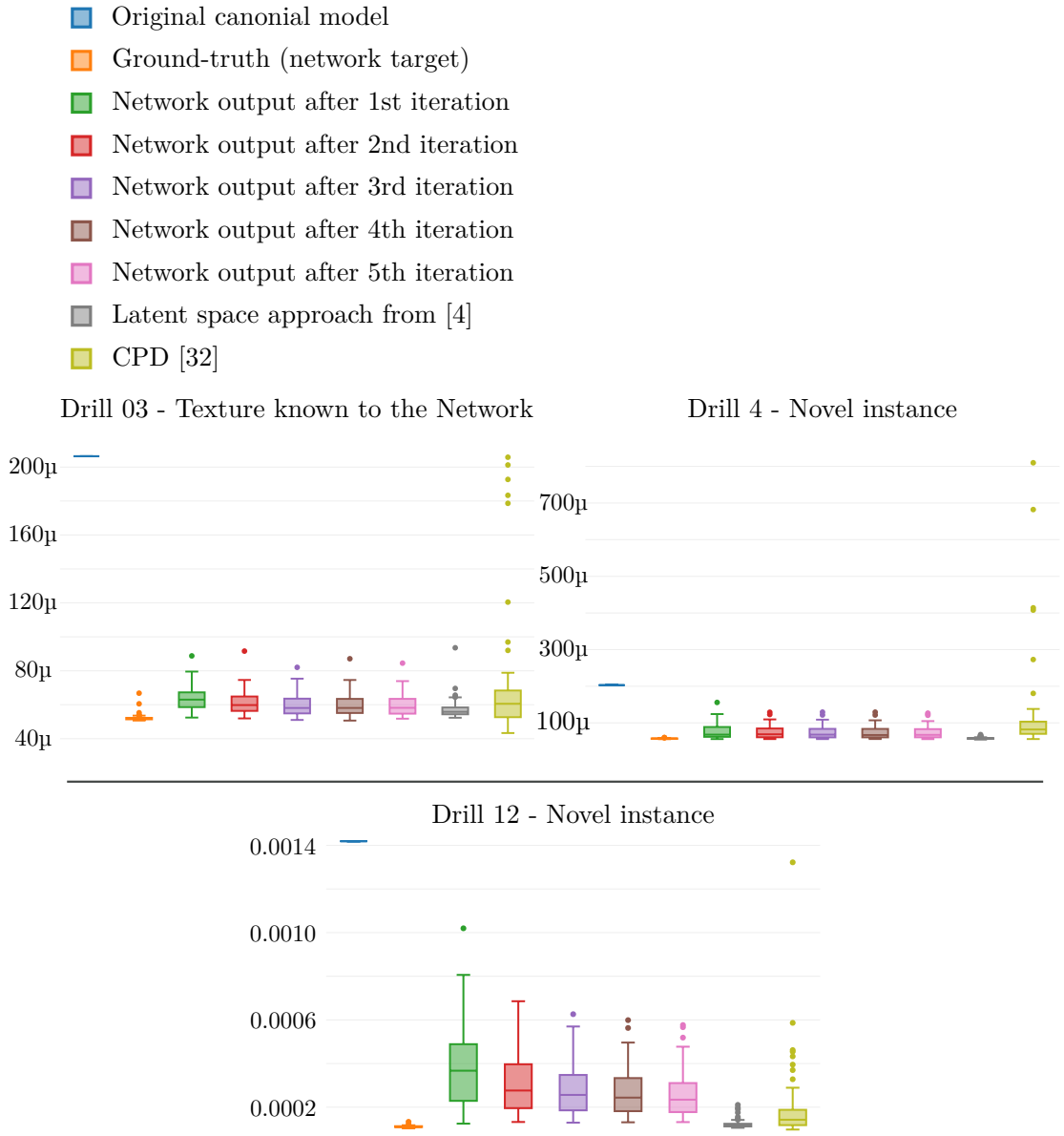

Figure 5.3: Boxplots showing the error described in Equation (5.1) between the point cloud of the observed drill and the canonical point cloud after it was deformed according to different approaches. For each boxplot the same 73 views on the observed object where used.

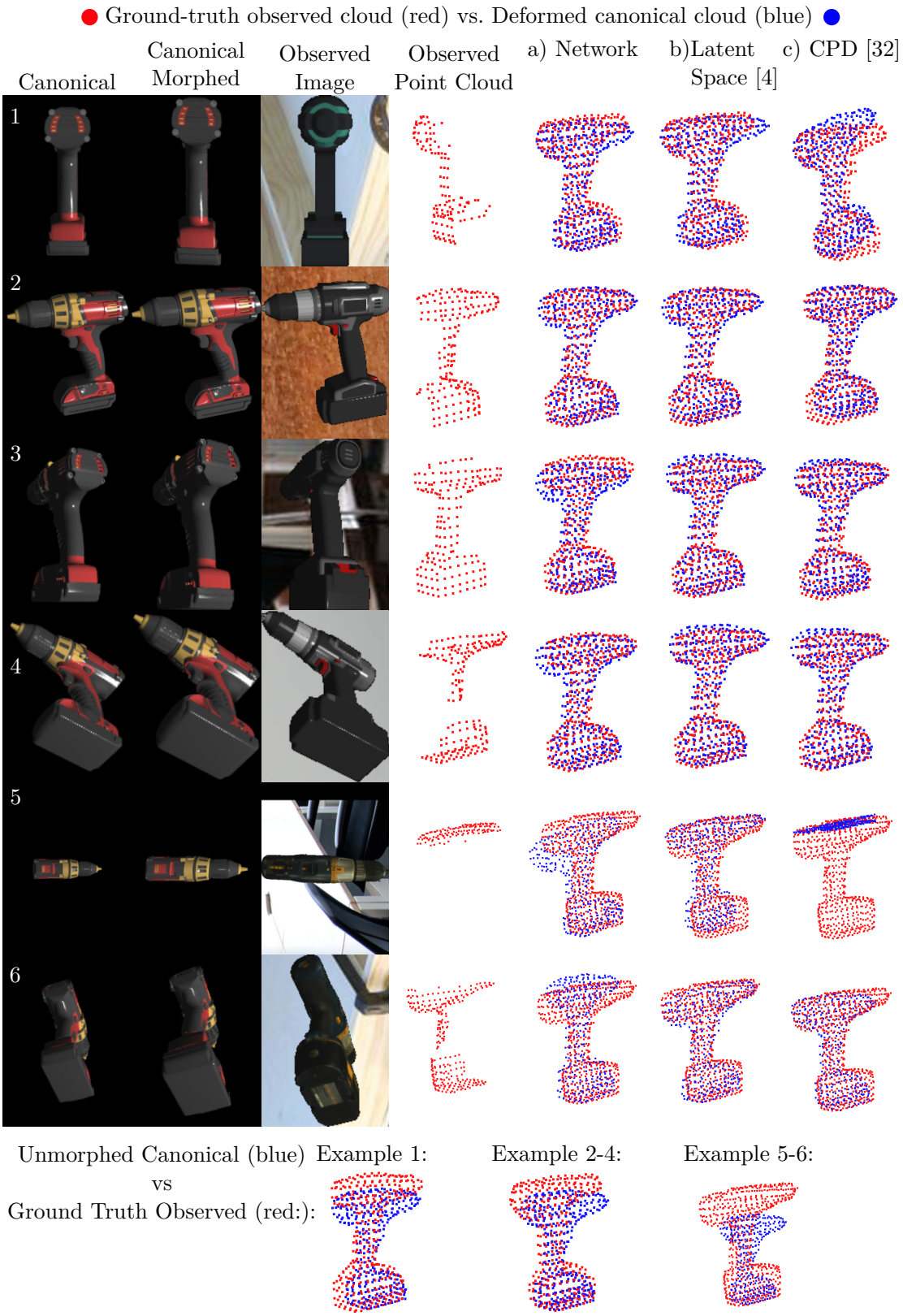

Figure 5.4: The rendered canonical image on the left, together with the observed RGB image are base for the rendered image of the deformed canonical model, placed second from the left, and for the deformation of the canonical point cloud placed fifth from the left. The observed point cloud seen in the image fourth from the left is the base for the deformations seen in the last two images (results from the latent space approach [4] and CPD [32])

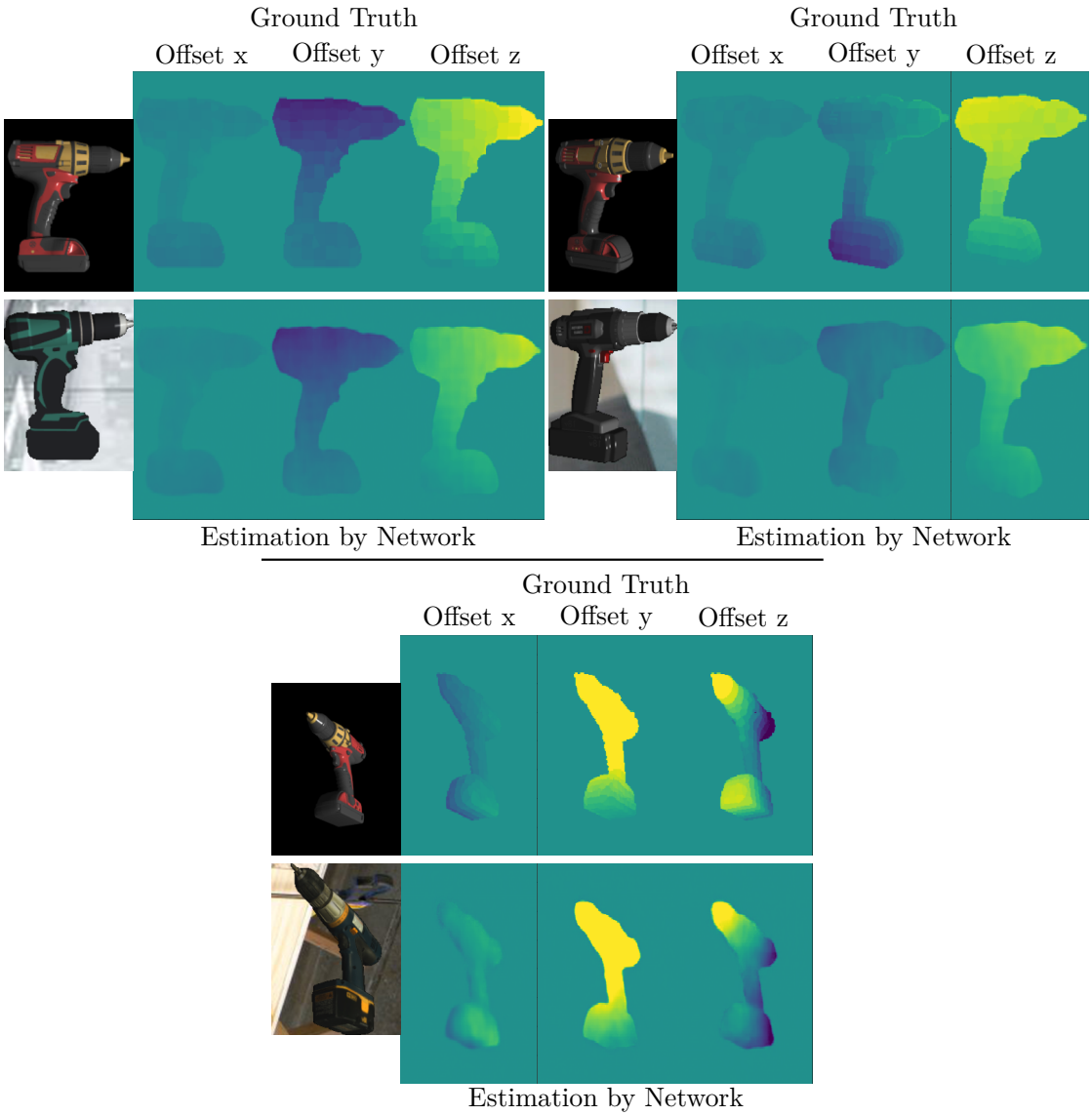

Figure 5.5: Three randomly chosen examples to compare the desired ground-truth to the actual network output on novel drills. The rendered canonical and observed input images are on the left, followed by heatmaps to represent the ground-truth at the top and the estimation from the network at the bottom.

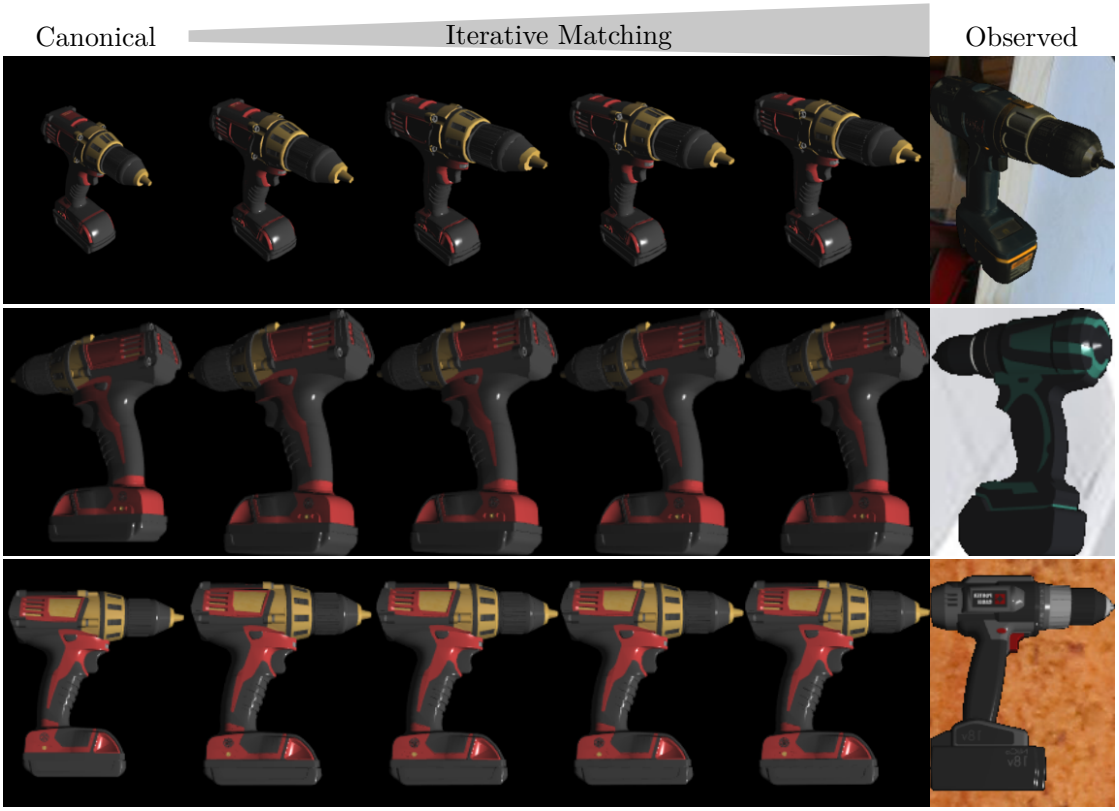

Figure 5.6: Stepwise evolution from the unmorphed canonical model on the left to match the observed model on the right. The examples are chosen randomly and five iterations where made.

6 Conclusion

We solved the problem of non-rigid registration from a single RGB image for the category of drills by finding an image based representation for deformation fields based on CPD. Building on this representation we proposed a network architecture that is inspired by optical flow estimation to compute a deformation field based on the observed image and a rendered frame of the canonical image in the same pose. We made a synthetic dataset for the category of drills to train and test our network and solved the problem of occluded parts by using PCA to generate a latent space of possible deformations within the category. We made experiments with the new dataset and calculated deformations in an iterative fashion, showing that our results are comparable to state of the art methods.
The approach excelled on the experiments with models where small and precise alignments had to be made and through the fact, that the variance of the results between different points of view was low. Our approach solved a much harder problem of non-rigid registration, than our competitors who had access to three dimensional data.
As a general insight from this thesis we noted that the bottleneck of our approach is the estimation done by the network and that future work should concentrate on minimizing this issue. The first idea is to adjust the training process to enable the network to generalize on the deformed canonical model and increase the performance of the network within the iterative registration process. Additionally we aim to simplify the network architecture to improve the estimation time and to decrease the consumption of memory. Finally, it is planned to apply our approach to different categories, to prove the approach to be working for other objects.

List of Figures

Bibliography

[1] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. "Flownet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2758–2766.

[2] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. "Flownet 2.0: Evolution of optical flow estimation with deep networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2462–2470.

[3] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. "DeepIm: Deep iterative matching for 6D pose estimation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 683–698.

[4] Diego Rodriguez and Sven Behnke. "Transferring category-based functional grasping skills by latent space non-rigid registration". In: *IEEE Robotics and Automation Letters (RA-L)* 3.3 (2018), pp. 2662–2669.

[5] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. "A naturalistic open source movie for optical flow evaluation". In: *European Conf. on Computer Vision (ECCV)*. Part IV, LNCS 7577. Springer-Verlag, 2012, pp. 611–625.

[6] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. "Vision meets robotics: the KITTI dataset". In: *The International Journal of Robotics Research (IJRR)* 32.11 (2013), pp. 1231–1237.

[7] Colin Rennie, Rahul Shome, Kostas E Bekris, and Alberto F De Souza. "A dataset for improved RGBD-based object detection and pose estimation for warehouse pick-and-place". In: *IEEE Robotics and Automation Letters (RA-L)* 1.2 (2016), pp. 1179–1185.

[8] Diego Rodriguez, Corbin Cogswell, Seongyong Koo, and Sven Behnke. "Transferring grasping skills to novel instances by latent space non-rigid registration". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 1–8.

[9] Anurag Ranjan and Michael J Black. "Optical flow estimation using a spatial pyramid network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4161–4170.

[10] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. "Deepflow: large displacement optical flow with deep matching". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013, pp. 1385–1392.

[11] Damien Teney and Martial Hebert. "Learning to extract motion from videos in convolutional neural networks". In: *Asian Conference on Computer Vision (ACCV)*. Springer. 2016, pp. 412–428.

[12] Christian Bailer, Kiran Varanasi, and Didier Stricker. "CNN-based patch matching for optical flow with thresholded hinge embedding loss". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3250–3259.

[13] David Gadot and Lior Wolf. "Patchbatch: A batch augmented loss for optical flow". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4236–4245.

[14] Aria Ahmadi and Ioannis Patras. "Unsupervised convolutional neural networks for motion estimation". In: *IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 1629–1633.

[15] J Yu Jason, Adam W Harley, and Konstantinos G Derpanis. "Back to basics: unsupervised learning of optical flow via brightness constancy and motion smoothness". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016, pp. 3–10.

[16] Yunjin Chen and Thomas Pock. "Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39.6 (2016), pp. 1256–1272.

[17] Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. "Reconstruction and representation of 3D objects with radial basis functions". In: *Proceedings of the 28th annual Conference on Computer Graphics and Interactive Techniques*. ACM. 2001, pp. 67–76.

[18] Michael Kazhdan and Hugues Hoppe. "Screened poisson surface reconstruction". In: *ACM transactions on graphics (ToG)* 32.3 (2013), p. 29.

[19] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. "3d-r2n2: a unified approach for single and multi-view 3D object reconstruction". In: *European Conference on Computer Vision (ECCV)*. Springer. 2016, pp. 628–644.

[20] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. "Shape completion using 3D-encoder-predictor CNNs and shape synthesis". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5868–5877.

[21] Jason Rock, Tanmay Gupta, Justin Thorsen, JunYoung Gwak, Daeyun Shin, and Derek Hoiem. "Completing 3D object shape from one depth image". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 2484–2493.

[22] Jiajun Wu, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T Freeman, and Joshua B Tenenbaum. "Learning shape priors for single-view 3D completion and reconstruction". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 646–662.

[23] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3D shapenets: A deep representation for volumetric shapes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1912–1920.

[24] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. "DeepSDF: Learning continuous signed distance functions for shape representation". In: *Arxiv preprint arxiv:1901.05103* (2019).

[25] Yun Zeng, Chaohui Wang, Yang Wang, Xianfeng Gu, Dimitris Samaras, and Nikos Paragios. "Dense non-rigid surface registration using high-order graph matching". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010, pp. 382–389.

[26] Vladimir G Kim, Yaron Lipman, and Thomas Funkhouser. "Blended intrinsic maps". In: *ACM transactions on graphics (TOG)*. Vol. 30. 4. 2011, p. 79.

[27] Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. "Efficient computation of isometry-invariant distances between surfaces". In: *SIAM journal on scientific computing* 28.5 (2006), pp. 1812–1836.

[28] Art Tevs, Martin Bokeloh, Michael Wand, Andreas Schilling, and Hans-Peter Seidel. "Isometric registration of ambiguous and partial data". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 1185–1192.

[29] Maks Ovsjanikov, Quentin Mérigot, Facundo Mémoli, and Leonidas Guibas. "One point isometric matching with the heat kernel". In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, pp. 1555–1564.

[30] Brett Allen, Brian Curless, and Zoran Popović. "The space of human body shapes: reconstruction and parameterization from range scans". In: *ACM transactions on graphics (TOG)*. Vol. 22. 3. 2003, pp. 587–594.

[31] Dirk Haehnel, Sebastian Thrun, and Wolfram Burgard. "An extension of the ICP algorithm for modeling nonrigid objects with mobile robots". In: *IJCAI*. Vol. 3. 2003, pp. 915–920.

[32] Andriy Myronenko and Xubo Song. "Point set registration: Coherent point drift". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 32.12 (2010), pp. 2262–2275.

[33] A Yuille. "The motion coherence theory". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEF. Computer Society Press. 1998, pp. 344–354.

[34] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society: series b (methodological)* 39.1 (1977), pp. 1–22.

[35] Sam T Roweis. "EM algorithms for PCA and SPCA". In: *Advances in neural information processing systems*. 1998, pp. 626–632.

[36] Michael E Tipping and Christopher M Bishop. "Mixtures of probabilistic principal component analysers". In: (1998).

[37] Michael E Tipping and Christopher M Bishop. "Probabilistic principal component analysis". In: *Journal of the royal statistical society: series b (statistical methodology)* 61.3 (1999), pp. 611–622.

[38] Andrew S Glassner. *Principles of digital image synthesis: Vol. 1*. Vol. 1. Elsevier, 1995.

[39] Shenchang Eric Chen and Lance Williams. "View interpolation for image synthesis". In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (CG)*. ACM. 1993, pp. 279–288.

[40] Arul Selvam Periyasamy, Max Schwarz, and Sven Behnke. "Refining 6D object pose predictions using abstract render-and-compare". In: *Arxiv:1910.03412* (2019).

[41] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 652–660.