Department of Mathematics and
Computer Science
Freie Universität Berlin

Acquisition of 3D-Head-Models using SLR-Cameras and RGBZ-Sensors

A thesis submitted to receive the degree of

Master of Science

April 29, 2013

Wolfgang Paier
Mat.Nr: 4400789

1st examiner	Prof. Dr. Raúl Rojas
2nd examiner	Prof. Dr.-Ing. Peter Eisert
Supervisor	David C. Blumenthal-Barby

Abstract

For automatic authentication of subjects at access controls or security gates it can be usefull to have a three-dimensional model of a subject's head, for example to improve the performance of face-recognition algorithms by adjusting lighting or pose. This thesis describes a system for the automatic acquisition of a well textured face model of a subject walking through a security gate. Two Kinects and an SLR-camera are used to gather depth-data and to take a high quality picture of the subject's head. The collected depth-data is merged to obtain an improved head model. Since the subject is moving, all depth-scans must be aligned before further processing can performed. For the alignment process it's important to consider that head and torso are not tied rigidly together, so only the head alone can be used to perform the alignment.

By employing the ICP-algorithm and warp-optimization, the collected depth-scans are aligned to each other and to the SLR-image. Then, the depth-scans can be merged to obtain an improved facial geometry and the SLR-image can be used to replace the low definition Kinect-image. The main contribution of this work is the registration of consecutive depth scans from both Kinects and the alignement of the resulting model to the SLR-image.

Contents

List of abbreviations

avg.	Average
DoF	Degree of freedom
ICP	Iterative Closest Point
SLR	Single-Lens Reflex
SVD	Singular value decomposition
TSDF	Truncated signed distance function
I	Identity matrix

1 Introduction

With the emergence of consumer 3D-sensors in the last years (e.g. Microsoft Kinect, ASUS Xtion) new possibilities for the reconstruction of 3D-models arose. These devices are inexpensive, small and typically provide more than just raw depth-data (e.g. colour images, body motion tracking, depth-segmentation). Also they can be used out of the box to generate point-clouds or coloured triangle-meshes (with a little more effort) of scanned objects.

However, compared to expensive laser scanners or sophisticated multi-camera setups, they deliver only low quality depth-data since their main purpose is to serve as an input device for natural user interaction applications. But it has been shown that it is possible to increase the quality of the obtained 3D-model by merging many depth-scans from different viewpoints [22]. The main challenge for such methods is the correct alignment of depth scans before they can be merged. The resulting 3D-model is smoother, contains more details and is more complete since it contains depth-information from different points of view. However, if a nicely textured 3D-model is required the images captured by the integrated colour-camera are not sufficient, since they are noisy and blurry. An alternative is to use an additional camera that captures a high quality colour image which is then used to texture the 3D-model. Depending on the type of colour-camera a more or less complex method must be employed to align the high quality colour-image with the obtained 3D-model.

A well known method for general image alignment problems is warp optimization. A warp describes an arbitrary parametric image deformation that can be used to remove optical distortions introduced by a camera or viewing perspective, to register an image with a reference grid such as a map, or to align two or more images [18]. However, if depth information for an image is available, a warp can also be used to re-render the image of an object from a different viewpoint. And inversely, if an image of the moved object is available, the new object-position can be inferred by finding optimal parameters of a warping function that re-produces the image of the moved object.

In this thesis, this technique of warp-optimization is used to obtain a well textured 3D model of a person's head. Depth-maps from two Kinects are pre-aligned using the ICP (Iterative Closest Point) algorithm. Then, warp-optimization is employed to register the depth-maps with the SLR-image. Finally, the aligned face scans are averaged to obtain an improved geometry and the SLR-image can be used to replace the low quality texture.

Head models have a wide range of applications: they can be used to establish virtual eye contact in the context of video conferencing, they are necessary for many image-based-rendering tasks or they can be used in the domain of security and surveillance to improve the quality of automatic as well as manual authentication.

This thesis is structured as follows: in chapter two, existing methods used in the context

of 3D reconstruction, especially the reconstruction of facial geometry, are summarized. Chapter three will cover necessary fundamentals of computer vision which are assumed to be known by the reader. The actual implementation is explained in chapter four and five, where the calibration of the camera setup, the scanning procedure and the actual reconstruction is described. In chapter six presents the results of the developed method and chapter seven draws a conclusion.

2 Related Work

Reconstruction of head and facial geometry or 3D-geometry in general is a well studied problem in the field of computer vision. Therefore a large variety of possible solutions exist, especially image based methods have a large share which is subsumed under the common term of Shape-From-X. Probably the most popular way to extract depth information from images is stereo vision. For example, Beeler et al. describe in [7] a stereo-vision system to capture highly detailed facial geometry. They use a NCC winner-take-all block-matching strategy. Then a refinement-algorithm is applied to enhance the initially coarse geometry. Finally a mesoscopic-augmentation-step adds fine features to the model that are basically too small to be reconstructed by the stereo-algorithm.

Schneider et al. [40] propose a global optimization based solution to calculate head-and-shoulder portraits from calibrated stereo images, but in contrast to [7], the reconstruction problem was formulated for a mesh-based inference of depth. They described the task as parametric warp estimation problem which is then solved in a robust Gauss-Newton framework.

Another solution for the reconstruction of 3D-geometry is to use an active light source that projects a known pattern onto the scene. Based on the deformation of that pattern in captured images, depth values can be inferred as well. Such approaches are presented in [32, 39], where structured light is used for head motion tracking and surface reconstruction.

In contrast to the previously mentioned methods that use image correspondences to infer depth values, Blanz et al. [10] propose to derive a morph-able model from a set of 3D face models by transforming the shape and texture of training samples into a vector space representation. New faces or facial expressions can then be modeled by forming linear combinations of the prototypes. In [25] Liao et al. use morph-able models but they are combined with a shape from shading approach to develop an algorithm for rapid 3D face reconstruction.

With the emergence of low cost consumer 3D-sensors in the last years (e.g. Kinect, Xtion) new possibilities for the reconstruction of 3D-models arose. RGBZ-sensors are used to scan an object from different points of view and an improved model is obtained by integrating many 3D-scans. The basic idea of such approaches is to improve the quality by averaging aligned depth-scans. Recent works on pose estimation show good results when working with geometry alone [15] or coupled with colour information [11]. For example in [22] Idazi et al. present an ICP based solution for camera pose tracking and depth-map fusion. They estimate the camera-pose in real time by aligning the current scan to a reference-model using a GPU implementation of the ICP-algorithm. Knowing the correct camera-pose, the current depth-scan is added to a volumetric model of the scene that integrates all captured depth-maps. Hernandez et al. describe in [28] the

application of such low-cost depth sensors to capture facial geometry based on temporal integration and spatial smoothing of depth scans. When only alignment of point clouds is needed, the ICP algorithm or one of its variants is a near choice [38, 41, 12]. However if the registration of 3D data to an image is desired, other algorithms must be applied. For example Paterson et al. [33] describe an image-based approach to estimate a head-pose using morph-able head-models. They use a non-linear optimization technique to find a reasonable head-pose as well as parameters for the head-model.

3 Theoretical background

This chapter will cover basics of image processing and computer vision which are referred to in the following chapters. Firstly, the model of a pinhole camera and its mathematical formulation is explained. The second part will give a short explanation of Euler angles, since they are used for the parametrization of the warp-optimization. The next section explains the mathematical model behind the Kinect's depth-sensor. The last two parts of this chapter will introduce the ICP algorithm for point-cloud registration and Least-Squares Problems.

3.1 Camera model and image formation process

Section 3 is mainly adapted from *Multiple View Geometry in Computer Vision* [19]. For a more detailed overview, interested readers may refer to this book.

A *camera model* is a mapping between the 3D coordinates and a 2D image coordinates. In computer vision applications, typically the pinhole camera model is used, see figure 3.1. The pinhole camera model represents a central projection of points in space onto the image plane. The projection of a point $\mathbf{x}$ given in camera coordinates is done by intersecting the image plane ($\mathbf{z} = f$) with a ray that joins $\mathbf{x}$ with the coordinate origin (figure 3.1).

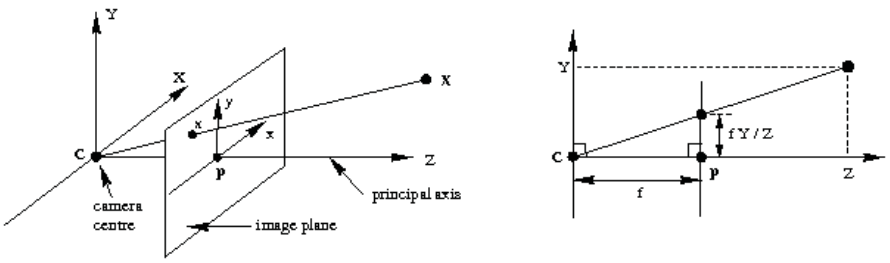


Figure 3.1: Model of a pinhole camera [19]

f	focal length
p	principal point

By similar triangles, it can be seen that the 3D-point $\begin{bmatrix} x & y & z \end{bmatrix}^T$ is mapped to the pixel at image coordinates $\begin{bmatrix} f\frac{x}{z} & f\frac{y}{z} \end{bmatrix}^T$.

$$\begin{bmatrix} x & y & z \end{bmatrix}^T \rightarrow \begin{bmatrix} f\frac{x}{z} & f\frac{y}{z} \end{bmatrix}^T \tag{3.1}$$

However, equation 3.1 assumes that the origin of image coordinates is identical to the principal point, while in practice this is not true (figure 3.2).

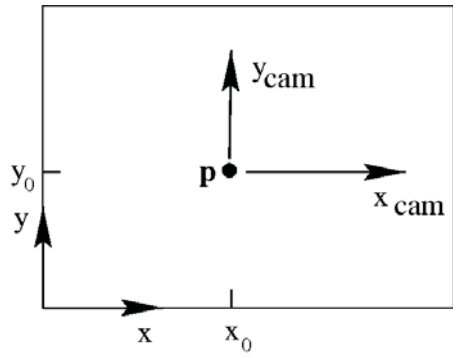


Figure 3.2: Image (x,y) and camera(x_{cam}, y_{cam}) coordinate system [19]

Therefore the mapping (equation 3.1) must be changed to consider the principle point offset $\begin{bmatrix} x_0 & y_0 \end{bmatrix}^T$.

$$\begin{bmatrix} x & y & z \end{bmatrix}^T \rightarrow \begin{bmatrix} f\frac{x}{z} + x_0 & f\frac{y}{z} + y_0 \end{bmatrix}^T \tag{3.2}$$

Using homogenous coordinates, equation 3.2 can be expressed by a single matrix multiplication, see equation 3.4). A point is converted to homogenous coordinates by adding a trailing one. All homogenous points $\begin{bmatrix} \mathbf{p}\omega & \omega \end{bmatrix}^T$ refer to the same point $\mathbf{p}$ in euclidean space. Geometrically spoken, the point $\begin{bmatrix} \mathbf{p}\omega & \omega \end{bmatrix}^T$ given in homogenous coordinates represents a ray in euclidean space that starts at the origin 0^T and passes $\mathbf{p}$. It can be converted back to euclidean coordinates by dividing it by ω, which is called de-homogenization (see equation 3.3):

$$\begin{bmatrix} x\omega \\ y\omega \\ z\omega \\ \omega \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.3}$$

Equation 3.2 can now be expressed by one matrix multiplication.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fx + zx_0 \\ fy + zy_0 \\ z \end{bmatrix} = \begin{bmatrix} \mathbf{K} & 0^T \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.4}$$

The matrix $\mathbf{K}$ is called the camera calibration matrix and it contains the focal length f

and the principal point $\begin{bmatrix} x_0 & y_0 \end{bmatrix}^T$.

$$\mathbf{K} = \begin{bmatrix} f & & x_0 \\ & f & y_0 \\ & & 1 \end{bmatrix} \tag{3.5}$$

For the previous equations it is assumed that the 3D point is given in camera coordinates. However, in general points will be expressed in terms of the world coordinate frame. Both coordinate frames are related via a rotation and a translation (figure 3.3).

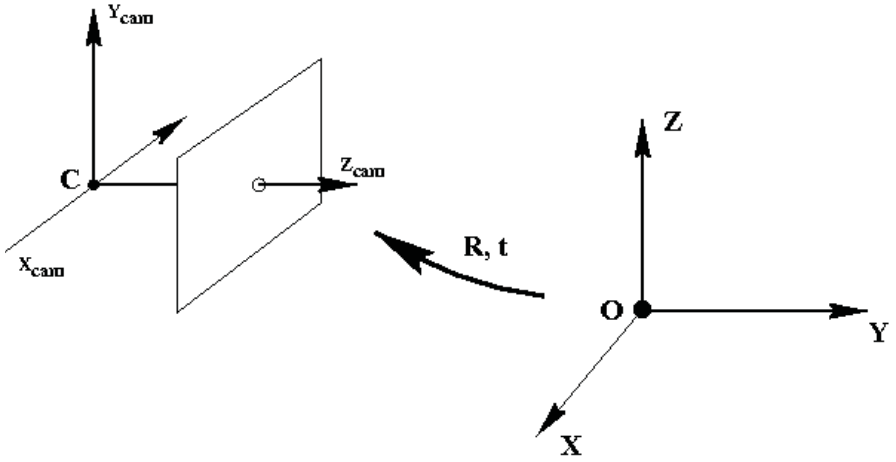


Figure 3.3: The euclidean transformation between the world and camera coordinate frames [19]

If $\mathbf{x} \in \mathbb{R}^4$ is a point in the world coordinate-frame given in homogenous coordinates and $\mathbf{x}_{cam}$ represents the same point in the camera coordinate frame, then $\mathbf{x}_{cam} = \mathbf{R}(\mathbf{X} - \mathbf{c})$ where $\mathbf{c}$ represents the coordinates of the camera centre in the world coordinate frame. $\mathbf{R}$ is a 3×3 rotation matrix that holds the orientation of the camera coordinate frame.

$$\mathbf{x_{cam}} = \begin{bmatrix} \mathbf{R} & -\mathbf{Rc} \\ 0 & 1 \end{bmatrix} \mathbf{x} \tag{3.6}$$

Putting all together leads to

$$\mathbf{p} = \mathbf{KR}\,[\mathbf{I} \mid -\mathbf{c}]\,\mathbf{x} \tag{3.7}$$

$$\mathbf{p} = \mathbf{K}[\ \mathbf{R} \quad | \quad \mathbf{t}\ ]\mathbf{x} \tag{3.8}$$

With $\mathbf{t} = -\mathbf{Rc}$. All parameters in $\mathbf{K}$ are called intrinsic parameters while $\mathbf{R}$ and $\mathbf{t}$ are referred to as extrinsic parameters. Both $\mathbf{K}$ and $[\ \mathbf{R} \quad | \quad \mathbf{t}\ ]$ can be calculated from a set of images with a known 3D geometry (e.g. chessboard pattern with known side length).

3.1.1 Image Distortion

For real (i.e. non pinhole) lenses the assumption of a linear projection model does not hold. The most important deviation is the radial distortion, see figure 3.4.

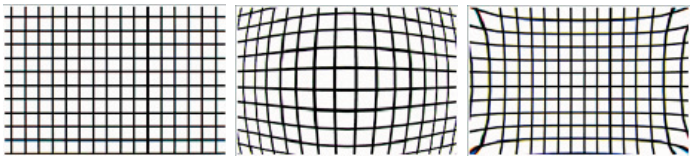


Figure 3.4: no distortion, barrel distortion and pincushion distortion [24]

The distortion pushes or pulls pixel away from or to the distortion center. This effect depends on the pixel's distance from the distortion center, i.e. the farther away the stronger the distortion is. The cure is to correct the image measurements to those that would have been obtained under a perfect linear camera action (figure 3.5) [19].

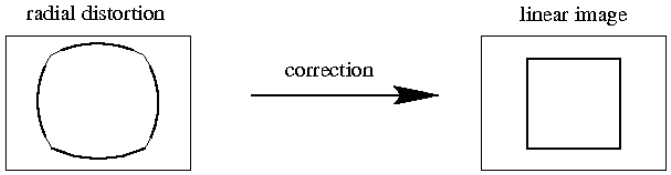


Figure 3.5: The image of a square with strong radial distortion is corrected to one that would have been obtained under a perfect linear lens [19]

For the correction, an undistortion-factor $\mathcal{L}(r)$ can be approximated by a Taylor expansion $\mathcal{L}(r) = 1 + \kappa_1 r + \kappa_2 r^2 + ... + \kappa_n r^n$. With $[\ x_c \quad y_c\ ]^T$ as the distortion center and r as the pixel's distance from the distortion center, the correction can be calculated by:

$$\begin{bmatrix} x_{corr} \\ y_{corr} \end{bmatrix} = \begin{bmatrix} x_c + \mathcal{L}(r)\,(x - x_c) \\ y_c + \mathcal{L}(r)\,(y - y_c) \end{bmatrix} \tag{3.9}$$

The corrected point $[\ x_{corr} \quad y_{corr}\ ]^T$ is now related to the coordinates of the 3D world point by a linear projective camera. The undistortion coefficients $\kappa_1, \kappa_2, \kappa_3, ...$ are counted to the intrinsic parameters of a camera. In practice it is sufficient to use a reduced set of undistortion parameters (see equation 3.10), as well as to use the principal point as distortion centre.

$$\mathcal{L}(r) = 1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6 \tag{3.10}$$

The undistortion parameters can be estimated from the same set of images that was used to calculate $\mathbf{K}$ and $[\ \mathbf{R} \quad | \quad \mathbf{t}\ ]$. For wide angle lenses the intrinsic parameters can be extended further to cover the tangential distortion too [42]. The whole procedure of finding $\mathbf{K}$, $[\ \mathbf{R} \quad | \quad \mathbf{t}\ ]$ and $\kappa_1, \kappa_2, \kappa_3$ is known as camera calibration. There exist already many tools/frameworks (e.g. *OpenCV*) which can be used to calibrate a certain camera.

3.1.2 Mapping between cameras

In a fully calibrated setup consisting of more than one camera, one can calculate the mapping of a pixel between two cameras if its depth is known. Given $\mathbf{p}_1$ as a pixel in camera $\mathbf{K}_1$ with known depth z_1, then corresponding pixel $\mathbf{p}_2$ in camera $\mathbf{K}_2$ can be calculated as follows. First the 3D point needs to be reconstructed from its 2D position and depth.

$$\mathbf{p}_1 = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\mathbf{x} = [\ \mathbf{R}_1^T \quad | \quad -\mathbf{R}_1\mathbf{t}_1\ ] \begin{bmatrix} z_1(x - x_0)/f_x \\ z_1(y - y_0)/f_y \\ z_1 \end{bmatrix} \tag{3.11}$$

The vector $\mathbf{x}$ represents $\mathbf{p}_1$ and $\mathbf{p}_2$ in world coordinates. Using equation 3.8 $\mathbf{x}$ can now be re-projected onto the image plane of camera $\mathbf{K}_2$.

$$\mathbf{p}_2 \quad = \quad \mathbf{K}_2[\ \mathbf{R}_2 \quad | \quad \mathbf{t}_2\ ]\mathbf{x} \tag{3.12}$$

3.2 Euler angles

Euler angles are a minimal and human readable form of defining a rotational motion. A point in 3D-space is rotated by applying three consecutive rotations around one of the three orthogonal axes, see figure 3.6.

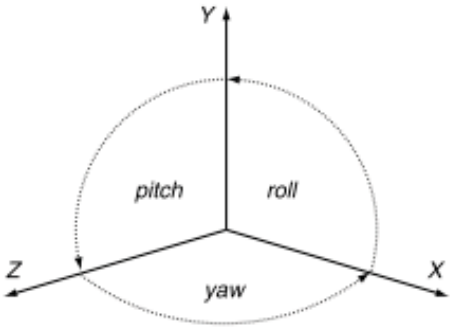


Figure 3.6: The convention for roll, pitch and yaw angles [46]

Rotate roll about the z-axis:

$$\begin{bmatrix} cos(\varphi_3) & -sin(\varphi_3) & 0 & 0 \\ sin(\varphi_3) & cos(\varphi_3) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate pitch about the x-axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\varphi_1) & -sin(\varphi_1) & 0 \\ 0 & sin(\varphi_1) & cos(\varphi_1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate yaw about the y-axis:

$$\begin{bmatrix} cos(\varphi_2) & 0 & sin(\varphi_2) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\varphi_2) & 0 & cos(\varphi_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

When these transforms are applied, the vertex is first rotated about the z-axis (roll), followed by a rotation about the x-axis (pitch), followed by a rotation about the y-axis (yaw). For performance reasons a pre-calculated rotation matrix can be used that performs all rotations in correct order at once instead of performing three separate rotations.

$T_{11} = cos(\varphi_3)cos(\varphi_2) - sin(\varphi_3)sin(\varphi_1)sin(\varphi_2)$

$$T_{13} = cos(\varphi_3)sin(\varphi_2) + sin(\varphi_3)sin(\varphi_1)cos(\varphi_2)$$

$$T_{21} = sin(\varphi_3)cos(\varphi_2) + cos(\varphi_3)sin(\varphi_1)sin(\varphi_2)$$

$$T_{23} = sin(\varphi_3)sin(\varphi_2) - cos(\varphi_3)sin(\varphi_1)cos(\varphi_2)$$

$$\mathbf{R} = \begin{bmatrix} T_{11} & -sin(\varphi_3)cos(\varphi_1) & T_{13} \\ T_{21} & cos(\varphi_3)cos(\varphi_1) & T_{23} \\ -cos(\varphi_1)sin(\varphi_2) & sin(\varphi_1) & cos(\varphi_1)cos(\varphi_2) \end{bmatrix} \tag{3.13}$$

Gimbal lock A major weakness of Euler angles is that under certain conditions one degree of freedom is lost. Then it is only possible to rotate a point around two axis. An example from [46] illustrates such a situation quite well:

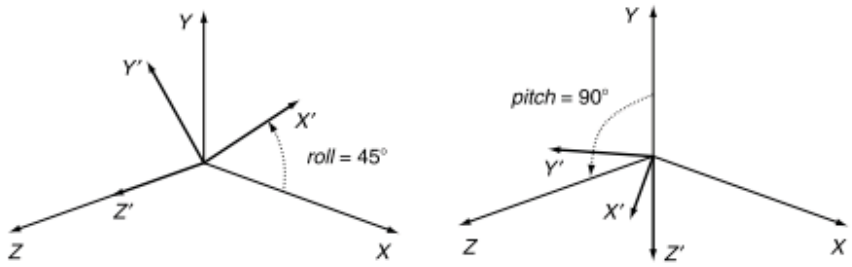


Figure 3.7: Roll of 45° (left) followed by a pitch of 90° (right) [46]

Figure 3.7 shows a roll of 45° followed by a pitch of 90°. If a yaw of 45° is performed now it would rotate the x'-axis towards the x-axis counteracting the effect of the original roll. Yaw has become a negative roll, caused by the 90° pitch. That means, basically one degree of freedom is lost, since it is only possible to rotate around two axis.

3.3 Acquisition of 3D Data

Microsoft's Kinect [16] is an USB based image, depth and audio sensor. It has an RGB camera, a 3D sensor and an array of four microphones to enhance voice recognition, see figure 3.8. The depth sensor consists of an IR laser emitter and an IR camera. The Kinect uses a structured light approach to estimate depth values instead measuring the run time of laser pulses. In the following, a short explanation of the used depth measurement process provided. For a detailed description see [1, 16].

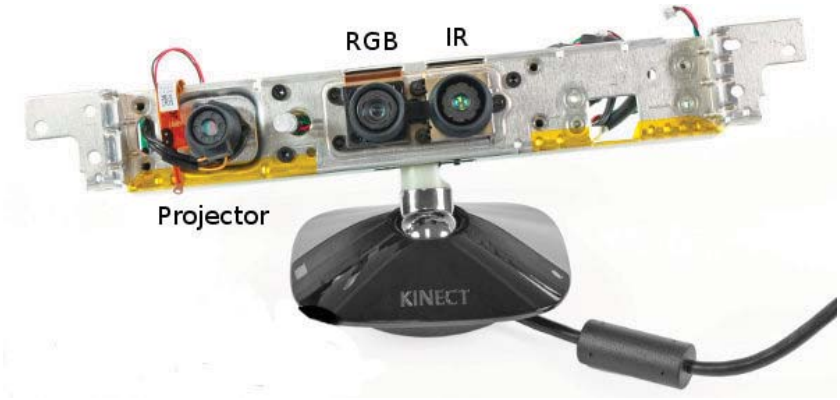


Figure 3.8: Exposed IR laser emitter, IR camera and RGB camera of a dismantled Kinect [36]

The laser source emits a single beam which is spit into multiple beams by a diffraction grating to create a constant pattern of speckles projected onto the scene, see figure 3.9.

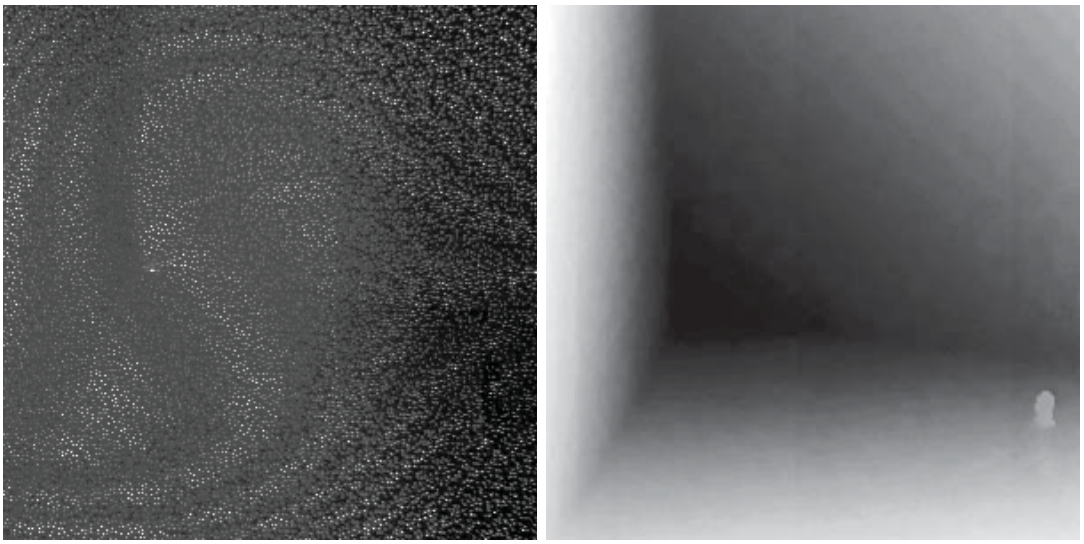


Figure 3.9: Left: cropped IR speckle pattern. Right: corresponding part of the depth image

Then, the IR camera captures the speckle pattern and it will be correlated against a reference pattern. This pattern was projected onto a plane at a known distance from the sensor and is stored in the sensor's memory. To estimate the depth at a single pixel of the depth image, the sensor analyzes the offset between the captured position and the position of the speckle in the reference pattern. When the distance is smaller or larger then that of the reference plane the position of the speckle in the infrared image is shifted in the direction of the baseline between the laser projector and the perspective centre

of the infrared camera. These shifts are measured using an image correlation procedure. Finally the depth can then be calculated from the measured disparity using triangulation. The following mathematical model is taken from [1] and explains the calculation of the object depth from the measured disparity and the stored reference pattern.

$$\frac{D}{b} = \frac{Z_0 - Z_k}{Z_0} \tag{3.14}$$

$$\frac{d}{f} = \frac{D}{Z_k} \tag{3.15}$$

Equation 3.14 and 3.15 can be derived from the similarity of triangles. Z_0 denotes the distance to the reference plane and Z_k the distance(depth) of the point k in object space, b is the base length, f is the focal length of the infrared camera, D is the displacement of the point k in object space and d is the observed disparity in image space. Substituting D from 3.15 into 3.14 and expressing Z_k in terms of the other variables yields:

$$Z_k = \frac{Z_0}{1 + \frac{Z_0}{fb}d} \tag{3.16}$$

Equation 3.16 is the basic mathematical model for the derivation of depth from the observed disparity, assuming constant parameters Z_0, f and b which can be determined by calibration. Besides depth, the Kinect delivers also colour images, a 3D skeleton of up to two persons and a simple segmentation of the depth-map. For resolutions up to 640x480 it runs at approximately 30 Hz. The colour camera can even deliver images at a resolution of 1280x720 but only at a reduced frame rate of 15Hz. Compared to other sensors it is inexpensive and easy to use. Basically it can be used out of the box with no further calibration, but I decided to calibrate it to achieve more accurate results.

3.4 Registration of point clouds

Often an object of interest needs to be reconstructed from multiple 3D scans, because the scanner provides a too small field of view or the object has a complex geometry which can not be captured from a single point of view. To achieve this, the scanner and/or the object is moved during the scanning procedure. The movement can be modeled as rigid body motion $\mathbf{T}$ which consists of a rotational and translational component.

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tag{3.17}$$

It does not matter if only the scanner, the object or both are moved during the scan. It always results in a single transformation T which represents the motion. So it can be assumed that scanner is always fixed and only the objects moves. To combine all scans into a single more complete model a compensation for this motion is needed. A well known approach to estimate the rigid motion between two point clouds is the ICP(iterative closest point) algorithm. The ICP was introduced by Chen and Medoni in 1991 [12]. It takes a source point cloud P, a target point cloud Q and estimates a geometric transformation that minimizes the distance between both point clouds. However, the point clouds have to be roughly aligned before the ICP can be applied. Since many improvements and optimizations have been proposed I stick to the basic algorithm because, it is simple and gives a good understanding. Two problems need to be solved to align the point clouds. Firstly, corresponding points from both point clouds must be identified and secondly, a rigid motion needs to be calculated that minimizes the distance between corresponding points. The advantage of the ICP is that point correspondences do not need to be known in advance, because the algorithm solves both problems (correspondence- and transformation-estimation). Since a single point does not contain enough information to discriminate it from others, spatial closeness is used to establish preliminary correspondences. That means, a point $\mathbf{p}$ from point cloud P corresponds to the closest point $\mathbf{q}$ from point cloud Q. With estimated correspondences a rigid transformation $\mathbf{T}'$ can be estimated. Applying the transformation $\mathbf{T}'$ on P yields a new point cloud P_{new} that is closer to Q. To further improve the alignment, P_{new} is used as source point cloud to estimate a transformation that improves the alignment between P_{new} and Q. It has been shown that the algorithm converges to the correct solution by iteratively refining the alignment between both point clouds [8].

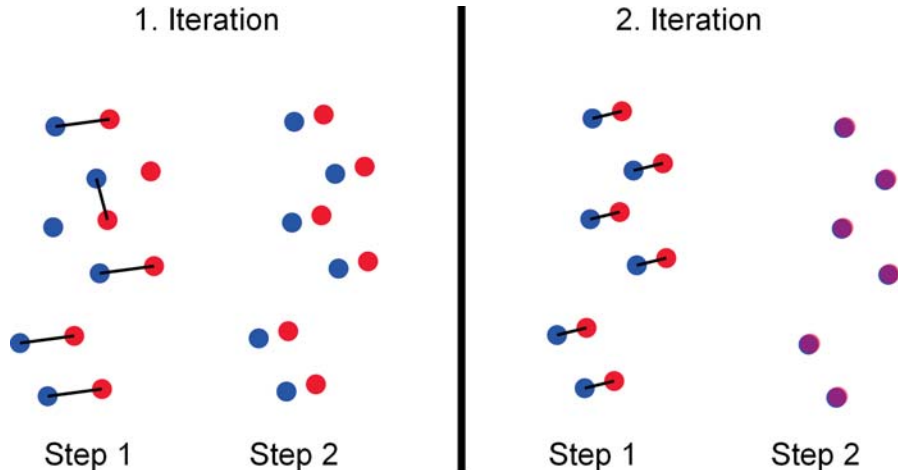


Figure 3.10: Two iterations showing functionality of ICP

Figure 3.10 shows the basic steps of the ICP-algorithm, that are repeated until convergence (e.g. distance between both point clouds falls under a certain threshold):

- Step 1: Estimate correspondences between points of P and Q

- Step 2: Estimate $\mathbf{T}'$ that minimizes the distance between corresponding points
- Step 3: Applying $\mathbf{T}'$ on P yields P_{new} that is used as source point cloud during the next iteration

The final $\mathbf{T}$ is obtained by concatenating all $\mathbf{T}'$. In the following a closed form solution for the transformation estimation problem is presented [6]. The objective is to minimize the distance the sum of the squared distances between corresponding point pairs (equation 3.18), with $\mathbf{R}$ being an rotation matrix and $\mathbf{t}$ a translation vector.

$$\mathcal{E} = \sum_{i=1}^{n} \|\mathbf{p}_i - (\mathbf{R}\mathbf{q}_i + \mathbf{t})\| \tag{3.18}$$

First calculate the centroid c of each point cloud.

$$\mathbf{c}_p = \frac{1}{n}\sum_{i=1}^{n} \mathbf{p}_i \ , \ \mathbf{c}_q = \frac{1}{n}\sum_{i=1}^{n} \mathbf{q}_i \tag{3.19}$$

Then calculate matrix $\mathbf{H}$ from the centred point clouds.

$$\mathbf{H} = \sum_{i=1}^{n} (\mathbf{p}_i - \mathbf{c}_p)(\mathbf{q}_i - \mathbf{c}_q)^T \tag{3.20}$$

Perform a singular value decomposition on $\mathbf{H}$ to calculate $\mathbf{R}$ and $\mathbf{t}$.

$$\mathbf{H} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T \tag{3.21}$$

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T \ , \ \mathbf{t} = \mathbf{q} - \mathbf{R}\mathbf{p} \tag{3.22}$$

This is only one possible solution to estimate the transformation between two point clouds. Other proposed methods use quaternions [21], additional information like colours [23] or minimize a point to plane distance [12]. Minimizing the point to plane distance is slower than the closed form solution for the point to point distance but it has a significantly better convergence rate [38].

Further improvements: The most time consuming part is typically the search for correspondences. Point clouds can easily hold several hundred thousand points. An easy approach to increase the speed is to work only on a subset of both point clouds. Different

sampling strategies like uniform [45]/random sampling [27] or normal space sampling (ensures that samples have normals distributed as uniformly as possible) have been proposed to aid speed or convergence. Gelfang et al. [17] propose a geometrically stable sampling strategy for the point-to-plane error metric. Based on the Eigenvalues of the covariance matrix it is possible to determine which parts of the transformation are constrained. They calculate for each point which DoF (degrees of freedom) are constrained by this point. Then for the transformation estimation, only a subset is used, which constrains all DoFs of the transformation equal. Nevertheless, a naive approach to find the correspondences has a complexity of $O(n^2)$. This is not feasible. Therefore many implementations use a kd-Tree to speed up the correspondence search. A different approach to establish correspondences is projecting a 3D point onto the image plane of the other range camera [9, 30]. This strategy produces a slightly worse alignment per iteration but is two orders of magnitude faster than other matching strategies. A further improvement is to reject outliers before estimating the optimal transformation. A simple method is to reject correspondences between points p and q if $\|p - q\| > \delta$. Chetverikov et al. [13] propose to sort correspondences by their point to point distance and use only a certain percentage of the correspondences with the lowest distance (e.g. if the overlap of two point clouds is 70% than use only the closest 70% of all correspondences). Another possibility is to use a Random Sampling Consensus method to calculate the optimal motion parameter. Turk et al. [45] propose to reject all correspondences containing points on boundaries.

3.5 Least squares problems

The largest source of unconstrained optimization problems are least squares problems[31], therefore this section will focus on this type of optimization problems. Solving a problem in the least squares sense minimizes the sum of squared errors. Figure 3.11, shows an example of a least squares problem. Data (blue) sampled from an unknown function $g(t)$ is used to train a model (red) that explains the observed values. This is done by minimizing the difference between measured and predicted values.

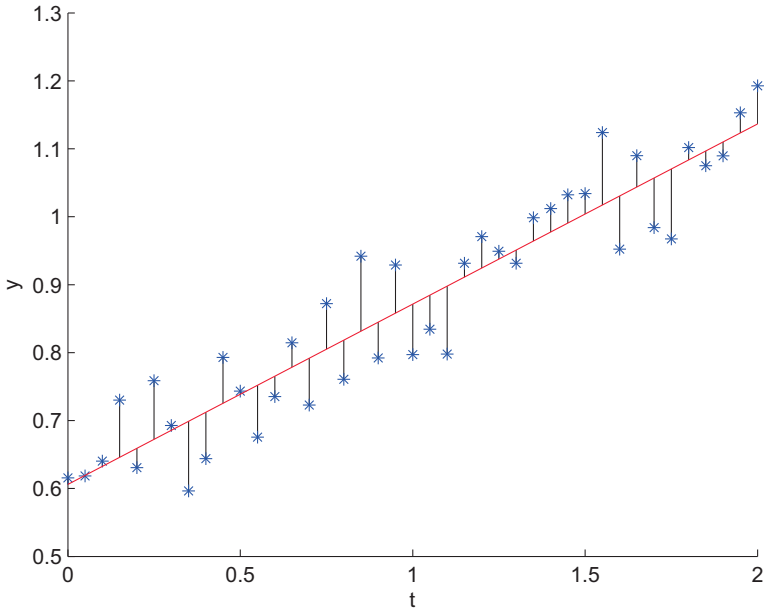


Figure 3.11: Least squares fit (red) of noisy data y (blue) sampled from a function $g(t)$. The black lines symbolize the errors between the fitted function and the sampled data.

The objective function of least-squares problems has a special form as shown in equation 3.23 with $\mathbf{x}$ as model parameter and $r_j(\mathbf{x})$ as residuals that represent differences between the observed and predicted values.

$$f(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(\mathbf{x}) \tag{3.23}$$

The sampled data from figure 3.11 seems to originate from a linear function plus some measurement errors. A probable model that describes the observed data could look like equation 3.24, with t being an input value and $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$ being the parameters of the linear model.

$$g(t_j) = \mathbf{x}_1 t_j + \mathbf{x}_2 \tag{3.24}$$

For this model a residual looks like equation 3.25.

$$r_j(\mathbf{x}) = g(t_j) - y_j \tag{3.25}$$

Equation 3.26 shows the objective function in matrix notation that must be minimized to solve the fitting problem with $\mathbf{A} \in \mathbb{R}^{m \times 2}$ holding the input values and $\mathbf{y} \in \mathbb{R}^m$ containing the measured data.

$$\mathbf{A} = \begin{bmatrix} t_1 & 1 \\ \vdots & \vdots \\ t_m & 1 \end{bmatrix}$$

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2 \tag{3.26}$$

It can be seen that this objective function is convex and that any solution $\mathbf{x}^*$ with $\nabla f(\mathbf{x}^*) = 0$ is a global minimizer of f [31]. The gradient of $f(\mathbf{x})$ can be expressed as follows (equation 3.27) with $J(\mathbf{x})$ being the Jacobian and $r(\mathbf{x})$ as the residual vector.

$$r(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{y}$$

$$\nabla f(\mathbf{x}) = \sum_{j=1}^{m} r_j(\mathbf{x}) \nabla r_j(\mathbf{x}) = J(\mathbf{x})^T r(\mathbf{x}) \tag{3.27}$$

The Jacobian $J(\mathbf{x})$ is the $m \times n$ matrix of first partial derivatives of the residuals [31]:

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots & \frac{\partial r_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial x_1} & \frac{\partial r_m}{\partial x_2} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix} \tag{3.28}$$

Equation 3.29 shows the gradient of the objective function (equation 3.26).

$$\nabla f(\mathbf{x}) = \mathbf{A}^T (\mathbf{A}\mathbf{x} - \mathbf{y}) \tag{3.29}$$

Therefore the solution $\mathbf{x}^*$ must satisfy the following equation that can be obtained by rearranging equation 3.29:

$$\mathbf{A}^T \mathbf{A}\mathbf{x}^* = \mathbf{A}^T \mathbf{y} \tag{3.30}$$

Equation 3.30 contains the *normal equations* of the optimization problem (3.26). The solution can be found by solving equation 3.30 for $\mathbf{x}^*$ (e.g. by employing the Moore-

Penrose pseudoinverse of $\mathbf{A}$)

$$\mathbf{x}^* = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{y} \tag{3.31}$$

Nonlinear least-squares problems

However, more complex tasks can not be described well by a linear model. Therefore a nonlinear objective function must be used. Such objective functions can not be minimized directly, so iterative methods must be employed. Iterative methods are for example Gradient Descent, Newton's method or the Gauss-Newton algorithm.

They need an initial estimate for the parameter vector $\mathbf{x}$ where the optimization is started. The objective function is then approximated around the current working point to find an update direction that is used to improve the parameter vector $\mathbf{x}$. The resulting parameter vector is then used to start a new iteration as long as the value of the objective function decreases. Other stopping criterions could be a maximum number of iterations or a minimal change $\epsilon > 0$ of the objective function $f(\mathbf{x})$. It is crucial that the initial parameter vector $\mathbf{x}_0$ is already close to a good solution otherwise the algorithm might get stuck in a local minimum which leads to a bad overall solution.

A simple method for nonlinear optimization is called Gradient Descent. It approximates the objective by a linear function and makes a step towards the negative gradient of $f(\mathbf{x})$.

Algorithm 3.1 Pseudo code of gradient descent

$\mathbf{x}_0 = \mathbf{x}_{init}$
$e_0 = f(\mathbf{x}_0)$
do
$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha\nabla f(\mathbf{x}_i)$
$e_{i+1} = f(\mathbf{x}_{i+1})$
while$(e_i - e_{i+1} \geq \delta)$

If α is too small convergence can be slow, especially on plateaus where the gradient is small. If α is too big, the algorithm fails to decrease the cost function. A common way to improve the algorithm is to use an adaptive step size determined by some kind of line search [44]. In that case an inexact line search finds a step size which reduces the function value (an exact line search would try to find a local minimum). The step size is increased (e.g. $\alpha = \alpha * 1.1$) as long as the error improves, otherwise try to make smaller steps (e.g. $\alpha = \alpha * 0.5$). This helps to improve the convergence speed at plateaus and allows to reduce the error monotonically.

Second order methods

Second order methods also use the second order derivative to perform a parameter update. One well known algorithm for second order optimization is Newton's method. It approximates $f(\mathbf{x})$ by a quadratic function $g(\mathbf{x})$ at the current working point, instead of a linear function as before. Then, a step towards the minimum of $g(\mathbf{x})$ is made which can

be calculated analytically. The algorithm is similar to *Gradient Descent*, but it differs in the parameter update:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma \mathbf{H}^{-1} \nabla f(\mathbf{x}) \tag{3.32}$$

Where $\mathbf{H}$ is the *Hessian* of $f(\mathbf{x})$ that contains the second order derivatives.

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{3.33}$$

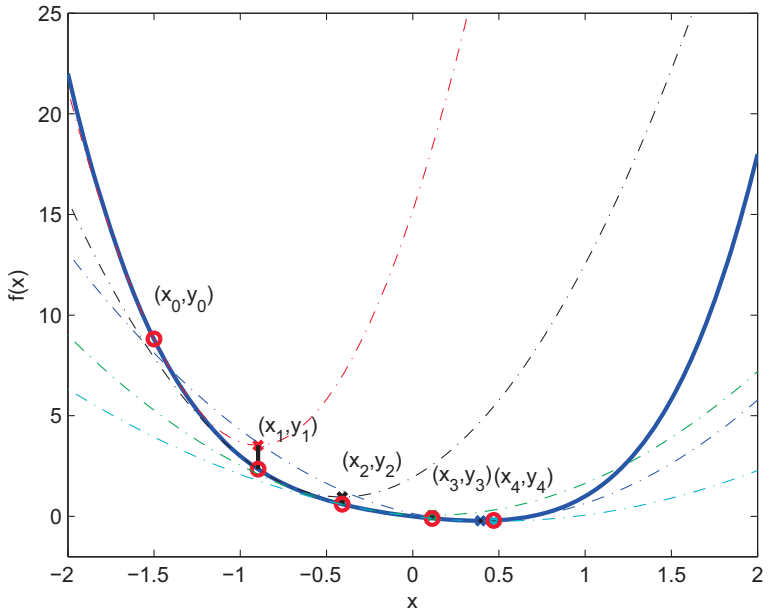


Figure 3.12: Illustration of the Newton's method

Figure 3.12 shows an illustration of Newton's method. A first local quadratic approximation at the initial point $x_0 = -1.5$ is formed (red). The corresponding minimizer is the new iterate, x_1. A new quadratic approximation of the function is formed at that point (black), leading to the second iterate, x_2. The process is repeated until a minimum in x_4 is reached. One drawback of this method is the need to calculate the inverse Hessian matrix, which can be time consuming (especially when $\mathbf{x}$ is a high dimensional vector).

The *Gauss-Newton* algorithm is an alternative to Newton's method, since it does not need *H(f)* to perform a parameter update. It provides better convergence properties like Newton's method without the need to explicitly calculate the Hessian. This advantage comes with a loss of generality, because it can only be applied to cost functions that are a sum of squared function values (e.g. $E(x) = \sum_{i=0}^{n} f(x_i)^2$). The *Hessian* of a least squares objective function can be calculated as follows:

$$\nabla^2 f(\mathbf{x}) = \sum_{j=1}^{m} \nabla r_j(\mathbf{x}) \nabla r_j(\mathbf{x})^T + \sum_{j=1}^{m} r_j(\mathbf{x}) \nabla^2 r_j(\mathbf{x}) \tag{3.34}$$

$$= J(\mathbf{x})^T J(\mathbf{x}) + \sum_{j=1}^{m} r_j(\mathbf{x}) \nabla^2 r_j(\mathbf{x}) \tag{3.35}$$

In many applications the first term in equation 3.34 is dominating, because the residuals are close to affine near the solution (i.e. $\nabla^2 r_j(\mathbf{x})$ are relatively small) or the residuals ($r_j(\mathbf{x})$) itself are small [31]. The Gauss-Newton algorithm exploits this fact and approximates the Hessian by the following expression:

$$\nabla^2 f(\mathbf{x}) \approx J(\mathbf{x})^T J(\mathbf{x}) \tag{3.36}$$

Now the search direction $\mathbf{p}$ can be obtained by solving the following equation:

$$J(\mathbf{x})^T J(\mathbf{x}) \mathbf{p} = -J(\mathbf{x})^T r(\mathbf{x}) \tag{3.37}$$

The parameter update changes to:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma \left(J(\mathbf{x}_i)^T J(\mathbf{x}_i) \right)^{-1} J(\mathbf{x}_i)^T r(\mathbf{x}_i) \tag{3.38}$$

One can clearly see, that the update corresponds to the Linear-Least-Squares solution of (3.39) with $r(\mathbf{x})$ containing the residuals and γ as step size.

$$J(\mathbf{x}) \mathbf{p} = -r(\mathbf{x}) \tag{3.39}$$

4 Calibration and Scanning

In this chapter the camera setup and some preliminary work is presented that is necessary to actually collect test data. That includes a recording software and a calibration process for the camera setup.

4.1 Camera Setup

The setting for my experiments is a security gate where subjects need to be authenticated before they are allowed to move on. Subjects passing the gate are recorded by two Kinects and an SLR-(Canon EOS 550D) camera to create a 3D representation of the head. The Kinects are mounted on the left and on the right side of the gate at about 1,7m. The SLR-camera is placed farther away having a frontal view on the face.

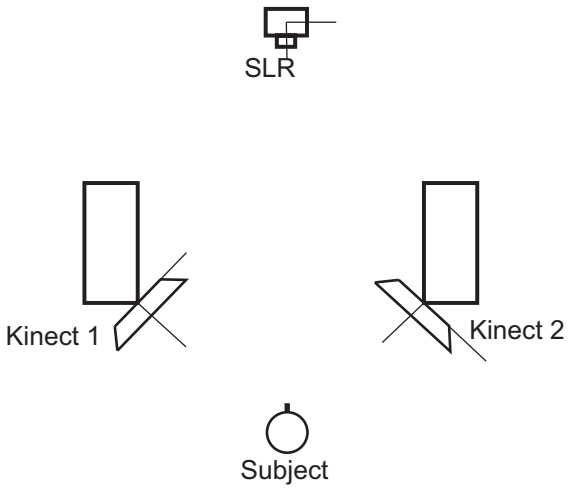


Figure 4.1: Topview of the camera setup

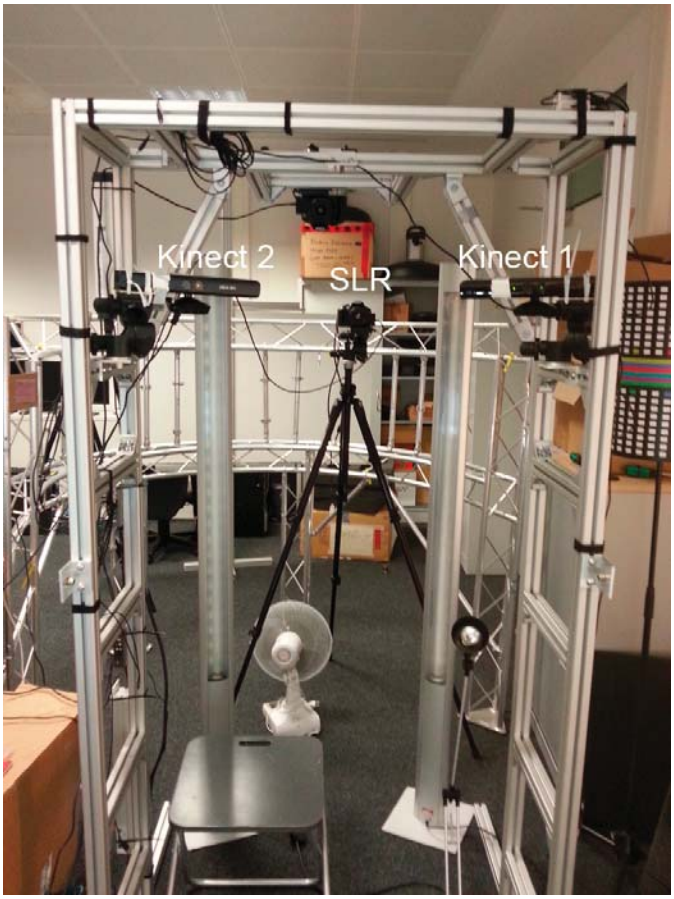


Figure 4.2: Photo of the used setup to acquire test data for the reconstruction

The mock security gate is an aluminum frame made from item® profiles. The Kinects are attached to the frame using an angle iron, a mount plate and a tripod head. The tripod head can be directly attached to the gate. To fix the Kinects rigidly, a small solution had to be developed (figure 4.3), since the Kinect has no original mounting point. This allows for a good view on the person's face for each Kinect without losing the benefit of a rigid camera setup that helps to reduce calibration errors caused by small changes of the camera's pose. It also helps to keep the calibration effort down to a minimum, since a single calibration of the whole setup is sufficient. Both Kinects are aligned inwards to have optimal view on the persons face.

Since a single Kinect occupies already a major part of the USB bandwidth, every Kinect must be connected to a separate USB-controller. The SLR camera is also controlled via USB using a library provided by the manufacturer. However a drawback is that the trigger time can not be determined accurately since there exists a comparatively high delay between the trigger command and its execution. Another issue is that the transfer of

Figure 4.3: Self made mount point

the SLR-image leads to several missing depth- and colour frames, which is problematic, since there is only a short period of time where useful depth-image can be captured. Therefore the SLR-camera was configured to store the image on the SD-card and the photo is added to the record afterwards by hand.

4.2 Third party libraries

Some third party libraries are used to reduce the coding effort for components which are not in the focus of this work. That includes OpenNI 1.5.4/Nite 1.5.2, OpenCV and the Point Cloud Library. OpenNI is an open source SDK to develop 3D sensing applications [3]. It consists of libraries for 3D sensing, voice command recognition, hand gestures and body motion tracking. In contrast to other SDKs it is not restricted to a special sensor (e.g. Microsoft Kinect, ASUS Xtion), or platform (e.g. Windows, Linux, OS X) which makes it very attractive to a wide range of developers.

OpenCV is a well known and widely used library for computer vision and image processing [2]. It provides data structures to hold images or matrices, basic image processing functions (e.g. filtering, conversion, ...) as well as advanced algorithms for feature detection, 3D reconstruction or machine learning. OpenCV is used in this project mainly for calibration, which will become apparent in the next section.

The Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing [4]. As the name already indicates, the target scope of PCL are algorithms for point cloud processing. That covers for example filters, feature detection, registration or segmentation. PCL's ICP implementation is used for the registration of point clouds.

For general image processing purposes a department internal library called VMLib is employed. It contains basic data structures and algorithms for filtering, resizing etc.. Other department interal libraries are: CVCGWarpLib for the Gauss-Newton based refinement of head poses, CanonCaptureLib to trigger the SLR-camera via USB and the CVCGKinectLib to collect depth and color image from the Kinects.

4.3 Calibration

In contrast to other reconstruction methods (e.g. [5]) the camera setup needs to be calibrated in advance. That involvs intrinsic and extrinsic calibration of all cameras. As already explained in Chapter 3.1, the intrinsic calibration consists of the camera matrix and undistortion coefficients,

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

$$[\kappa_1, \kappa_2, \kappa_3, ...]^T$$

while the extrinsic calibration [$\mathbf{R}$ | $\mathbf{t}$] holds the pose of a camera w.r.t. a certain reference coordinate system. The intrinsic parameters are necessary to convert a depth-map into a point cloud or to project a 3D point (in camera coordinates) onto a camera (e.g. when determining colour values for a point cloud). The extrinsic calibration for example allows to switch from camera coordinates into certain reference coordinate system (e.g. world coordinates or just another camera coordinate system).

Although OpenNI provides API functions for pointcloud generation from depth maps and alignment of depth and colour image, a calibration of both Kinects yields higher accuracy for further processing steps. Also, the extrinsic and intrinsic parameters are necessary for the warp-based refinement of head poses. The calibration is done once when all cameras are adjusted and the their poses are fixed. For the calibration a 66cm×48cm chessboard pattern is used. Every Kinect has an IR- and colour-camera to be calibrated. For the calibration of the IR camera the IR projector must be covered otherwise the speckle-pattern confuses the chessboard detector. It is also important to have a light source that emits enough infrared light (e.g. sun or a lamp with no energy saving light bulb) otherwise the chessboard appears too dark. For the intrinsic calibration of the IR- and colour-camera several images of the chessboard-pattern are captured at different poses in front of the cameras. The chessboard corners are then extracted from these images by *cv::findChessboardCorners(...)* and *cv::cornerSubPix(...)*. Based on the extracted chessboard corners and the known geometry of the chessboard *cv::calibrateCamera(...)* calculates the intrinsic parameters of both cameras. The Kinect-internal extrinsics (i.e. pose of the colour camera w.r.t. the depth camera) are then calculated seperately by *cv::stereoCalibrate(...)* as proposed in the OpenCV's manual. Interestingly, the IR-camera's intrinsics can not be used directly to align the colour- and depth-images, because then there is an obvious offset between the depth- and the aligned colour image. It appears to be a consequence of the correlation window size used for the depth estimation [36]. A raw depth-image (figure 4.4) contains a null band on the right side. A reasonable explanation is that the Kinect starts to send the depth image at the first pixel with a calculated depth value. Since a correlation window is used the first pixel in

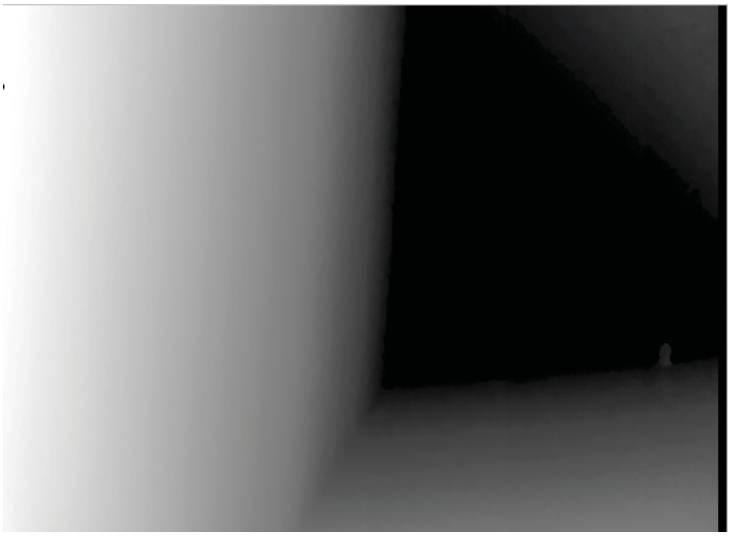


Figure 4.4: Raw depth image with null band on the right side

the depth-image does not correspond to the first pixel in the IR-image. The result is a constant principle-point offset on the order of -4.8 x -3.9 pixel between the calibration of the IR camera and the calibration which is used for depth-map processing. The extrinsic calibration of both Kinects with respect to the SLR-camera is precarious, since there are only a few poses where the chessboard pattern is completely visible by all cameras. A tool is used that automatically captures images from both Kinects and the SLR-camera when the pattern is detected. Since the Kinect as well as the SLR-camera does not support any kind of synchronization the chessboard pattern is placed on a tripod to ensure that small motions (e.g. when held by a human) do not degrade the calibration quality because the images are taken at different times. Finally, *cv::stereoCalibrate(...)* is used again to calculate the pose of both Kinects with respect to the SLR-camera.

4.4 Scanning

Two Kinect sensors are placed at the left and the right side of the gate to capture both sides of the subject's head. An additional SLR-camera is placed in front of the gate to take a high quality image, that can be used as texture. The scanning works as follows: The subject starts about three meters away in front of the gate. As the subject approaches the gate, the body tracking information can be used to decide which scans are stored for the reconstruction and which are discarded.

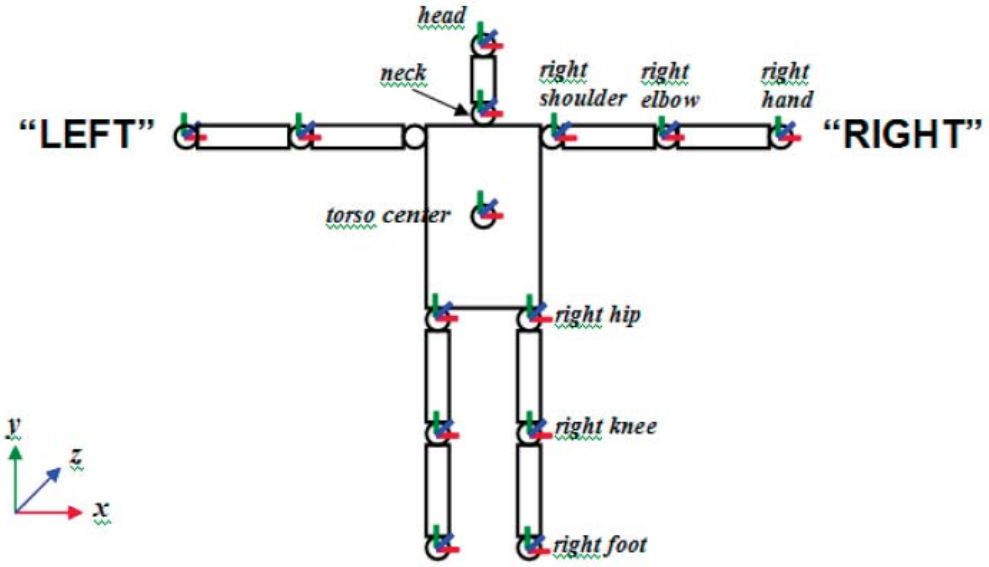


Figure 4.5: Joints of the skeleton tracking [34]

The body tracking information is represented as skeleton of about fifteen joints, as shown in figure 4.5. To decide if a 3D scan is stored or not, the head's distance to the Kinect sensor is observed. A 3D-scan is only used if the head is between 0.65m and 1.1m away. The lower threshold is chosen to ensure that the depth-maps are still valid and do not contain too many holes.

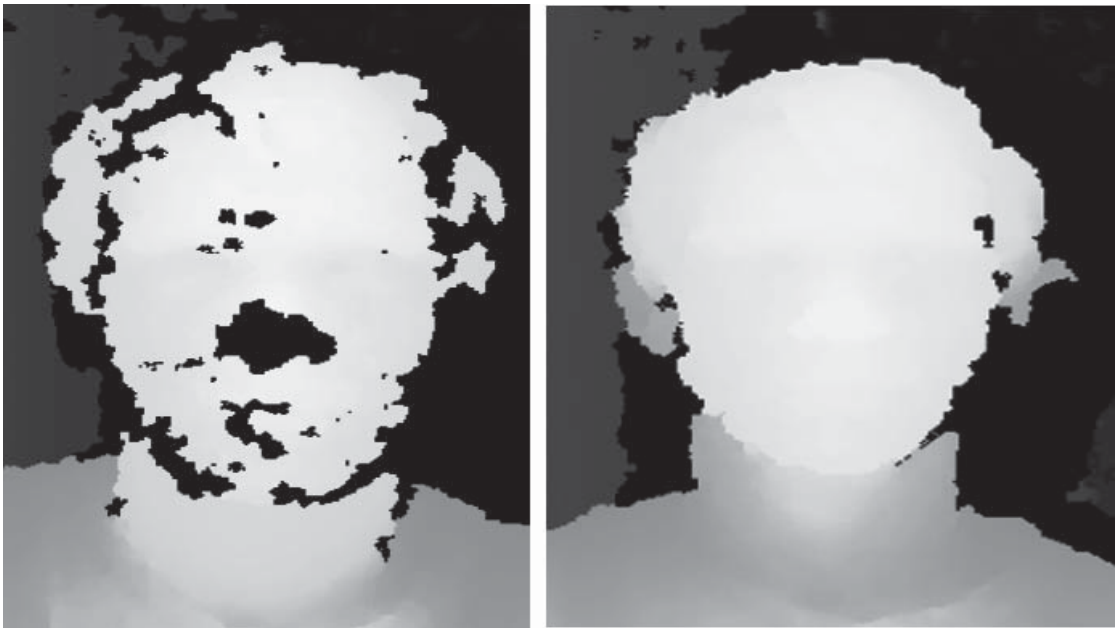


Figure 4.6: Raw depth-map at approx. 0.62m (left) and 1m (right)

One advantage of OpenNI over Microsoft's Kinect (for XBox) SDK is that it does not apply a pass through filter on the depth-map (i.e. it delivers also depth values smaller than 800mm and bigger than 4000mm). This can lead to incomplete depth-maps (see figure 4.6) when a subject gets closer, but it can be compensated over time and leads to a more detailed 3D model.

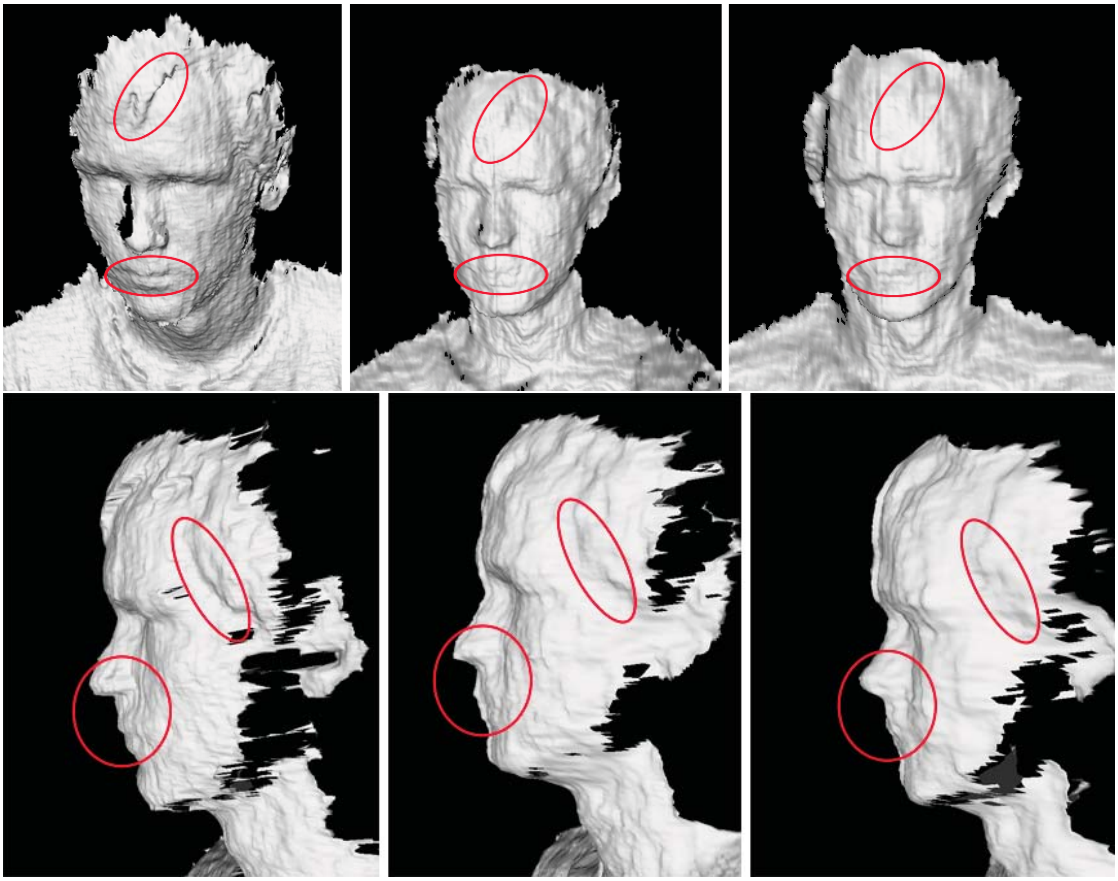


Figure 4.7: Facial geometry averaged from 50 depth-frames of a non moving subject at: 0.6m (left), 0.95m (middle) and 1.2m (right)

The upper threshold ensures that the depth-maps are still accurate enough to generate a detailed model of the subject's face. As demonstrated in figure 4.7 the distance of a scanned object plays a big role since the measurement error grows according to the square law [1]. Details of lips, hair and nose that are cleary visible at 0.6m disappear at a larger distance.

The SLR-camera is triggered automatically via the serial interface when the person is detected at a certain distance.

5 Reconstruction

Raw depth-data suffers from noise and discretization errors, see figure 5.1. To improve the quality of the final depth-map, consecutive depth scans are aligned and combined to a smoother and more detailed geometry. Also, a single Kinect can not cover the whole face of a subject. Therefore two depth sensors are used, one at the left and another one at the right side of the gate. Their scans are combined to obtain a complete model of the subject's face. Since the Kinect's RGB images are of inferior quality, an additional SLR-camera is used to acquire a highly detailed texture. By combining the data of all sensors, a better 3D model with a high quality texture can be created. The challenges which need to be dealt with are:

1. The synchronization of the left and the right depth-maps
2. The alignment of consecutive face scans
3. The alignment of the resulting 3D-model with the SLR-texture

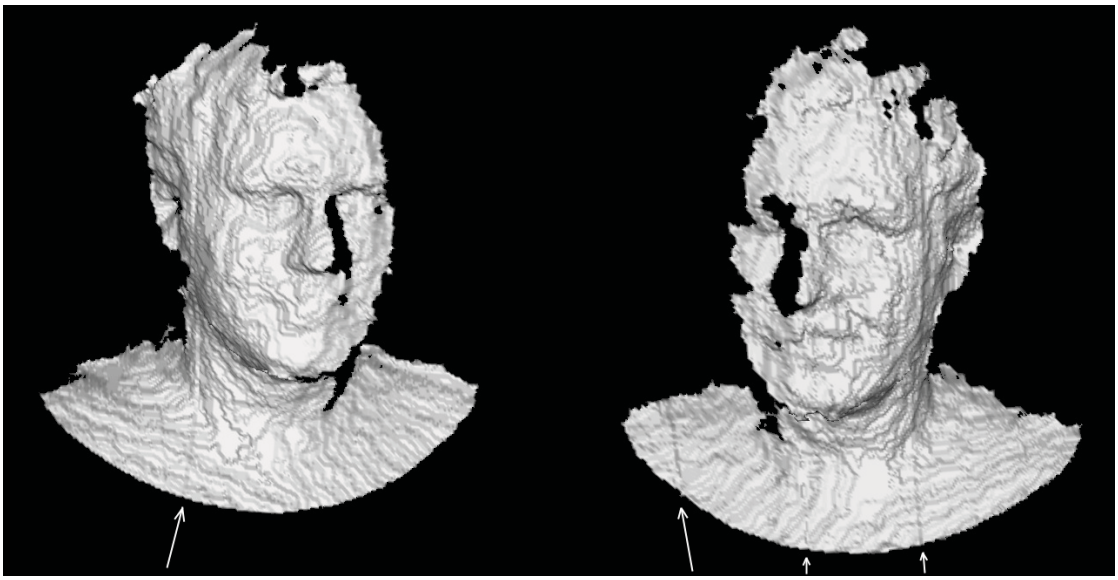


Figure 5.1: Noisy point cloud with discretization error and vertical bands (white arrows)

The following paragraph describes the reconstruction process (see figure 5.2), the used data and all assumptions which are made. For every subject that passes the gate, a Kinect sensor delivers approximately 20 depth-maps with corresponding colour images

at a usable definition. Every depth-colour image pair can be aligned using the calibration of the corresponding Kinect sensor (i.e. intrinsics/extrinsics of its IR and colour camera). An aligned pair of depth and colour image forms an RGBZ-image. The raw data is very noisy and suffers from discretization errors and other artifacts (e.g. vertical banding). To obtain an improved and well textured 3D model all RGBZ-images are aligned with the SLR-image. Then, the aligned data is averaged by projecting them onto the SLR-camera's image and calculating a robust average of all depth values.

The alignment process is split into two parts. Firstly, all scans are aligned to one reference using the ICP algorithm. After that initial alignment it is already possible to produce an improved depth-map since all scans are aligned. However, at this point of the reconstruction the SLR-image does not fit well since it is not aligned with the averaged model. In the second part the alignment of all scans is refined using warp optimization. The advantage of this technique is that it is possible to align a textured 3D model with an image. The alignment is driven by minimizing the overall intensity errors between RGBZ- and SLR-images.

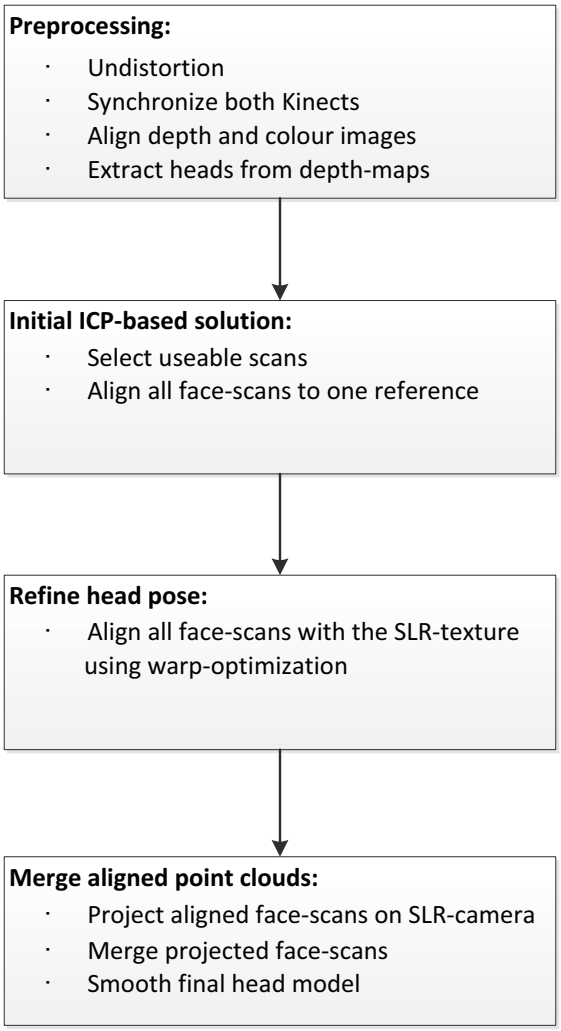


Figure 5.2: Diagram of the reconstruction process

It is assumed, that the left and right part of the subject's face perform almost the same rigid motion during the warp optimization (i.e. the left and right depth-map were captured at the same time). However, the Kinect does not provide any kind of synchronization with other Kinects. Therefore a synchronization stage is ran before starting the actual reconstruction which is presented in section 5.1.1. Another constraint is that corresponding pixel in the RGBZ- and SLR-image have similar intensity values. That is a problem, since both images origin from different types of image sensors. The solution for this problem is explained in subsection 5.2.4. Finally the aligned depth-maps are averaged to obtain an improved depth-map as explained above. Depending on the number of merged face-scans (i.e. the walking-speed of the person), the resulting model can still look a bit coarse. To cope with that a smoothing operator applied on the depth-map.

The following list explains which data is used during the reconstruction process:

1. Depth images of left and right Kinect are the basic data for the improved depth-map
2. Colour images of left and right Kinect are needed during the warp optimization
3. Body motion tracking information is used to extract only the person's head from point clouds
4. The SLR-image is used as texture for the final model
5. Intrinsic and extrinsic calibrations is needed to (un)project pixel and switch from camera to world coordinate system

5.1 Pre-processing

To determine which point clouds are actually used for the reconstruction, two thresholds δ_{min} and δ_{max} are used. A scan is only accepted if the subject's head is at least δ_{min} and at most δ_{max} away from the sensor. As explained in chapter 4,these thresholds are chosen to ensure that the point clouds do not contain too many holes (when the subject is too close) and are still detailed enough (because the subject is not too far away).

Though the Kinect has good lenses with low distortion coefficient, all images are undistorted. Also irrelevant data from the depth scans is removed using the body-tracking-information provided by the Kinect sensor. A bounding-cylinder is placed around the head-joint and all depth-pixel that correspond to 3D-points outside this cylinder are invalidated (i.e. set to zero). However, head and shoulder do not form a rigid object so a small cylinder is used that cuts off the shoulders too, otherwise the pose estimation would not work reliably since it assumes a non-morph-able object. Finally corresponding colour- and depth-images are aligned to form an RGBZ-image where corresponding depth- and colour-pixel have same image coordinates. Since intrinsic and extrinsic parameter of all Kinect cameras (IR and RGB) are known, the alignment can be accomplished by projecting a depth pixel onto the colour image (equation 3.11). The RGB-value can then be obtained by sampling (e.g. bilinear interpolation) from the colour image. The last part of the pre-processing stage is explained in the following subsection.

5.1.1 Synchronization

As USB device, the Kinect does not support any kind of synchronization with other devices. This is a problem, since 3D data from two Kinects should be merged without creating artifacts in the overlapping region. A simple idea would be to estimate a frame-offset based on the recording time, but that would ignore all kinds of timing issues between two not synchronized devices. Every frame is timestamped, but they origin from different clocks. Nevertheless this can be used to estimate the constant time offset between both clocks. Therefore a simple search strategy is used to estimate a time offset that minimizes the distance between corresponding point pairs from the left and right

point cloud, because temporal proximity results in small distances. The correspondences are established by reprojecting a depth pixel onto the other depth-image.

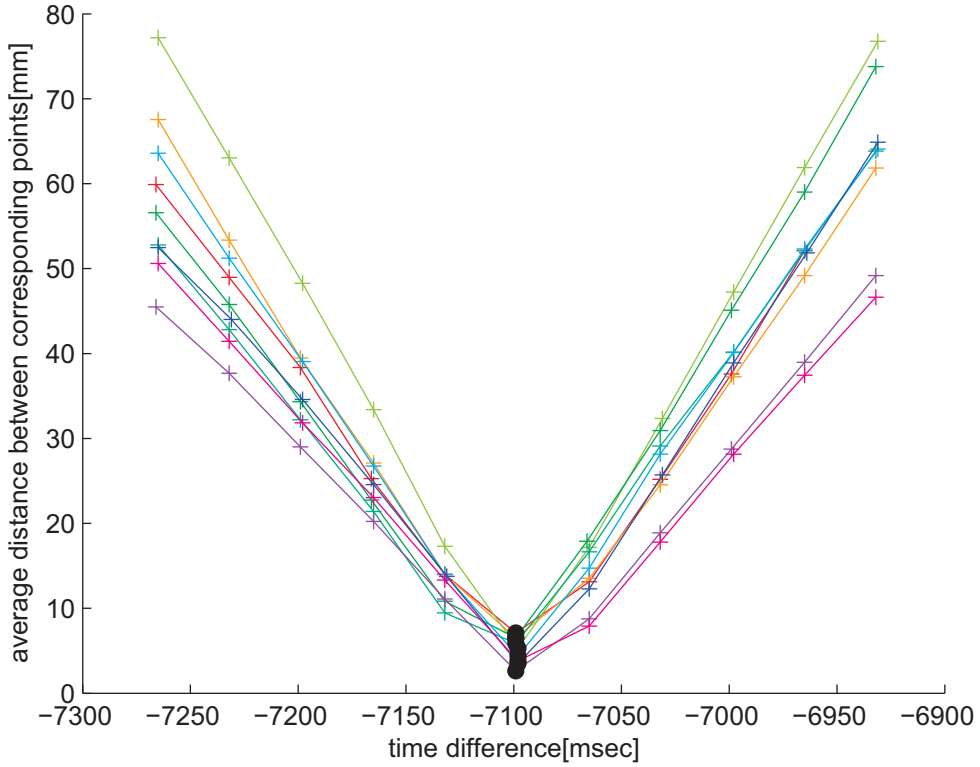


Figure 5.3: Average distance between left and right scan depending on the time offset. The best offset is makred by a black circle

The final estimate is an average time offset calculated from several results at different positions in the stream (e.g. the offset is calculated at every fifth frame). As shown in figure 5.3, the average distance decreases in a linear fashion until the optimal value for the time offset is reached and increases after that point again linearly, since a person walks approximatly at a constant speed.

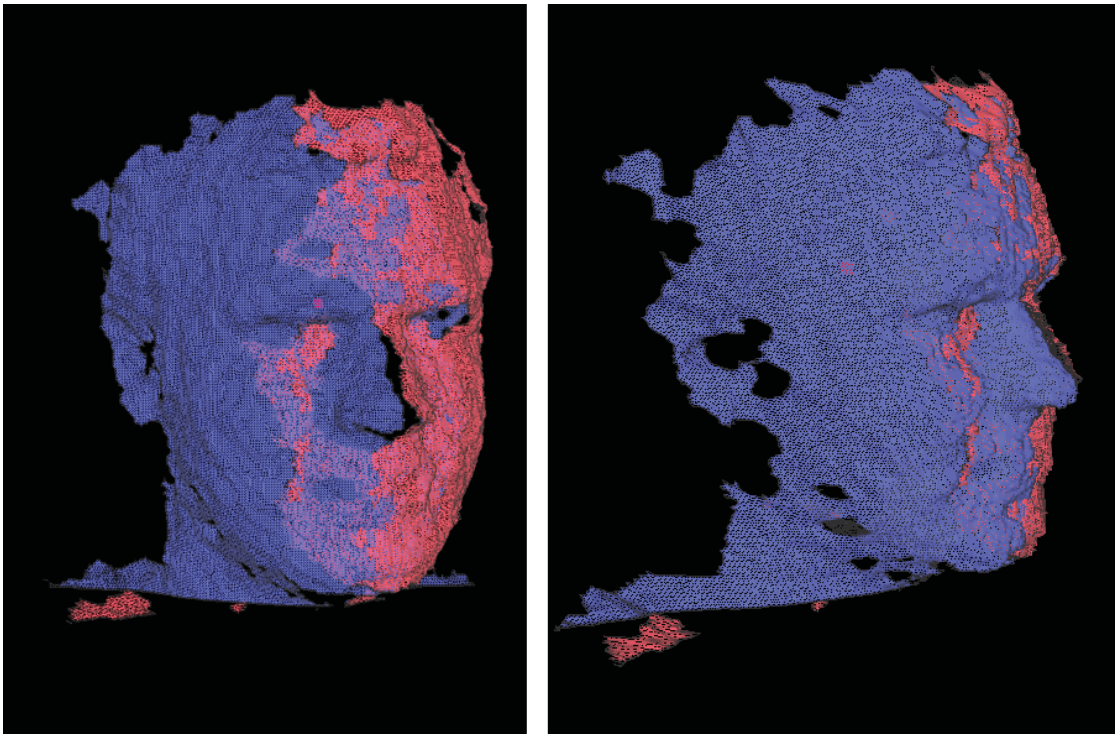


Figure 5.4: Synchronized raw data from both Kinects

Refining the synchronization

Now, left and right scans should be 16 milliseconds off at most, since the average gap between consecutive frames is 33 milliseconds. Though it seems neglect-able it still can cause a noticeable gap in the overlapping region (see figure 5.5). To cope with that problem, one point cloud is shifted forward or backward along an estimated motion-path until the previously mentioned distance is minimal. Possible rotational components are omitted, since it includes only motions on the order of some millimeters and this possible error can be corrected later during the warp-optimization-based refinement of the head pose. Also the overlapping region is too small to estimate rotations correctly (figure 5.4). To generate a motion-path for the synchronization refinement the body tracking information is used (i.e. the position of the head-joint at the previous, current and next time step). Figure 5.5 shows a reconstruction of the head with frame-exact and refined synchronization.

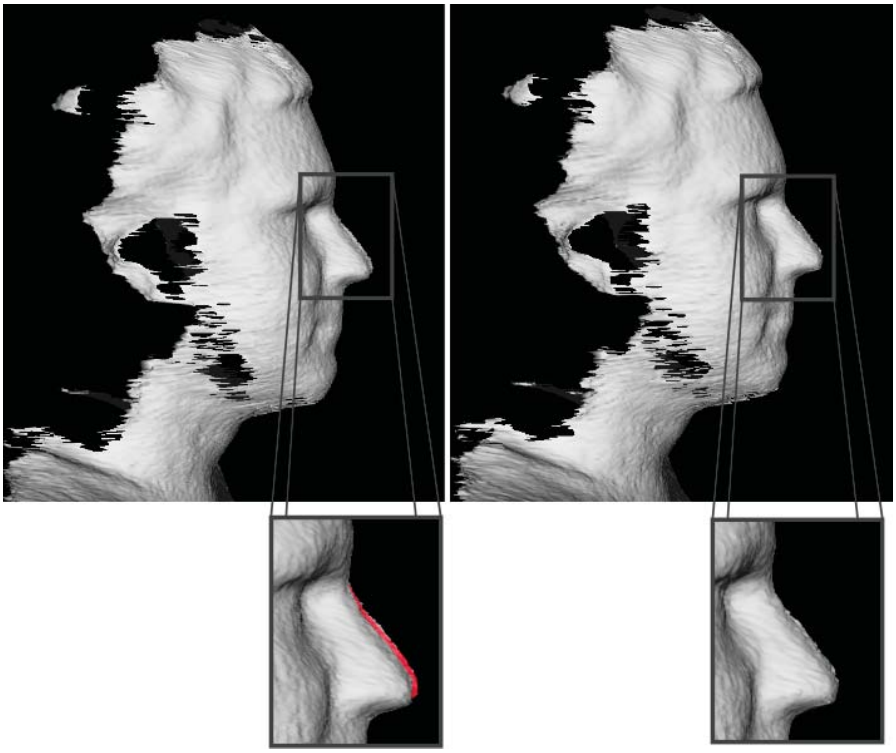


Figure 5.5: Left: reconstruction with frame exact synchronization. Right: reconstruction with refined synchronization.

5.2 Alignment of point clouds

This section describes how face-scans from different time steps are aligned, because that is necessary to average depth-images. A synchronized pair of a left and a right RGBZ-image form a face-scan F. All scans $F_{m:n}$ are aligned to a single reference scan F_{REF} using ICP. This approach is used to ensure that registration errors are not propagated (e.g. when performing only pairwise registration between consecutive scans F_i and F_{i+1}). The alignment process is started at F_{REF-1},F_{REF+1} and continues to F_m and F_n, respectively. This way, the alignment of every scan F_i can be initialized with the result of the previous one, since the motion between consecutive scans is small. Both parts of a face scan are aligned at once to ensure that they can not drift away from each other. Therefore the alignment has to be performed in world-coordinates, because a common rigid motion can not be optimized in different coordinate systems. The face-scan that fits the SLR-image best (i.e. it was recorded almost at the same time as the SLR-image) is selected as F_{REF}. As explained in section 4.1, it is not possible to determine exactly when the SLR-camera was triggered, since there is a comparatively high delay and the camera does not provide any usefull timing information. However, since all extrinsic and intrinsic camera

parameters are known, a simple intensity based approach can be used. By projecting every RGBZ-pixel (that belongs to the subject's head) onto the SLR-image an average intensity difference for every face-scan can be calculated, see figure 5.6. Since most alignment errors result from the fact that face-scans and SLR-image are captured at different times, it is likely that the face-scan with the lowest average intensity difference is also that one with the lowest temporal difference to the SLR-image. Brightness and contrast are adjusted before calculating the difference as explained in section 5.2.4 to achieve better results.

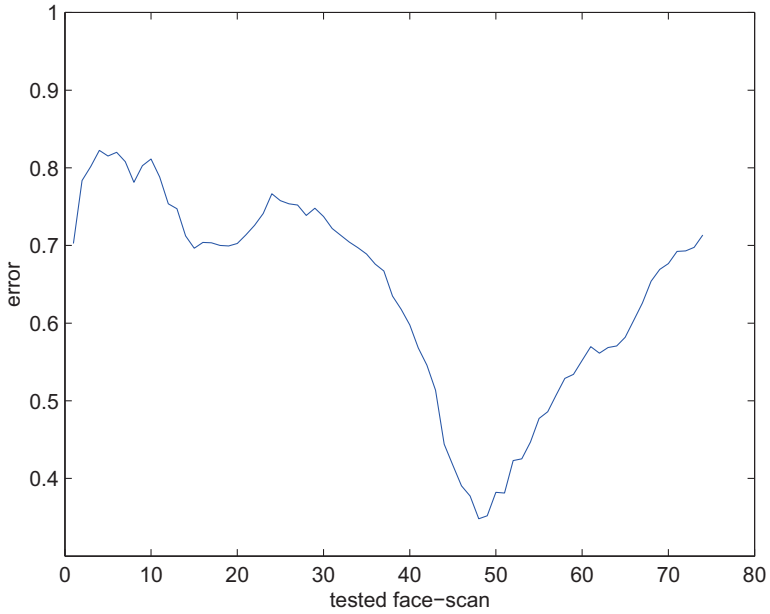


Figure 5.6: Avg. intensity difference between corresponding pixel in SLR-image and Kinect-image

After the ICP based registration, there might be still alignment errors between left and right part of the face due to errors in the extrinsic calibration, errors induced by synchronization refinement and the fact that F_{REF} might not be perfectly chosen.

5.2.1 Refining the head pose

After the initial ICP based registration all RGBZ-images should fit together but they do not fit the SLR-image well. However, it is not possible to use a geometry based alignedment since no depth information is available for the SLR-image. Therefore, the head pose is refined based on the texture match between RGBZ- and SLR-image. Assuming that corresponding pixel in $\mathcal{P}_{SLR}$ (SLR-image) and $\mathcal{P}_{Kinect}$ (Kinect-image) have an equal intensity an optimal alignment yields a zero valued difference image. This is an often

made assumption (e.g. when calculating optical flow) that is called brightness constancy constraint.

Based on that insight, it is a near choice is to formulate the task as optimization problem because the magnitude of intensity differences is related to the alignment quality. That means a good alignment yields low differences while a bad alignment causes high intensity differences between corresponding pixel. To calculate the image difference, every RGBZ-pixel from a face-scan is projected onto the SLR-image as explained in subsection 3.1.2 and the intensity difference is evaluated. Firstly, this calculation must be expressed as a parameterized function that can be processed by an optimization framework. In this case, the function is parametrized by a vector $\psi \in \mathbb{R}^6$ that defines a rigid motion $\mathcal{T}$ with $\mathbf{R}_\tau$ being a rotation matrix determined by three Euler angles $\varphi_1, \varphi_2, \varphi_3$ and $\mathbf{t}_\tau$ being a translation vector $\begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^T$:

$$\psi = [\varphi_1, \varphi_2, \varphi_3, t_x, t_y, t_z]^T$$

$$\mathcal{T}(\mathbf{p}, \psi) = \begin{bmatrix} \mathbf{R}_\tau & \mathbf{t}_\tau \end{bmatrix} \mathbf{p}$$

$\mathcal{T}$ represents a rigid motion that is used to align a face-scan with the SLR-image. Though Euler angles suffer from the possibility of a Gimbal lock, they offer an easy way of linearization which is an advantage when solving optimization problems. Also, all rotational motions are relatively small, so the probability of a Gimbal lock is very low.

The next part explains the reprojection of an RGBZ-pixel $\mathbf{q}_{RGBZ} = \begin{bmatrix} x & y \end{bmatrix}^T$ onto the SLR image to calculate the intensity difference between corresponding pixel. Firstly, the pixel $\mathbf{q}_{RGBZ}$ must be transformed from 2D image coordinates to 3D camera coordinates $\mathbf{p}_{RGBZ}$ using the measured depth value z and the intrinsic calibration $\mathbf{K}_{depth}$ of the depth camera.

$$\mathbf{K}_{depth} = \begin{bmatrix} f_x & & x_0 \\ & f_y & y_0 \\ & & 1 \end{bmatrix}$$

$$\mathbf{p}_{RGBZ} = \begin{bmatrix} z(x - x_{0,depth})/f_{x,depth} \\ z(y - y_{0,depth})/f_{y,depth} \\ z \\ 1 \end{bmatrix} \tag{5.1}$$

Based on the extrinsic calibration $\begin{bmatrix} \mathbf{R}_{SLR} & \mathbf{t}_{SLR} \end{bmatrix}$ of the Kinect w.r.t. the SLR-camera,

$\mathbf{p}_{RGBZ}$ can be transformed to the coordinate frame of the SLR-camera.

$$\mathbf{p}_{SLR} = \begin{bmatrix} \mathbf{R}_{SLR} & \mathbf{t}_{SLR} \\ 0^T & 1 \end{bmatrix} \mathbf{p}_{RGBZ}$$

At this point the optimized motion $\mathcal{T}$ is applied on $\mathbf{p}_{SLR}$.

$$\mathbf{p}_{aligned} = \begin{bmatrix} \mathbf{R}_\tau & \mathbf{t}_\tau \end{bmatrix} \mathbf{p}_{SLR}$$

The corresponding SLR-pixel $\mathbf{q}_{SLR}$ can now be found by projecting $\mathbf{p}_{aligned}$ onto the SLR-image using the intrinsic calibration $\mathbf{K}_{SLR}$.

$$\mathbf{q}_{SLR} = \mathbf{K}_{SLR}\Theta\left(\mathbf{K}_{SLR}\mathbf{p}_{aligned}\right)$$

The function $\Theta(\mathbf{p})$ performs a de-homogenization (i.e. divides a vector by its last component):

$$\Theta(\mathbf{p}) = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}, \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Now the intensity difference between a single RGBZ-pixel $\mathbf{q}_{RGBZ}$ and the corresponding SLR-pixel can be expressed as follows:

$$\mathcal{I} = \mathcal{P}_{SLR}\left(\mathbf{K}_{SLR}\Theta\left(\begin{bmatrix} \mathbf{R}_\tau & \mathbf{t}_\tau \end{bmatrix}\begin{bmatrix} \mathbf{R}_{SLR} & \mathbf{t}_{SLR} \\ 0^T & 1 \end{bmatrix}\mathbf{p}_{RGBZ}\right)\right) - \mathcal{P}_{Kinect}\left(\mathbf{q}_{RGBZ}\right) \quad (5.2)$$

Finally an objective function $\mathcal{E}(\psi)$ must be defined, that considers the overall intensity difference of a face-scan. Here, a least-squares formulation is used since their special structure makes them easier to be solved. The general structure of a least-squares problem looks like equation 5.3.

$$\mathcal{E}(\psi) = \frac{1}{2}\sum_{j=1}^{n} r_j^2(\psi) \quad (5.3)$$

Minimizing the objective $\mathcal{E}(\psi)$ yields an optimal parameter vector ψ^* (a rigid motion) that causes a low overall error (in this case low intensity differences). To formulate the objective for the alignment task the residual $r(\psi)$ needs to be replaced by equation 5.2. It is important to keep in mind that $\mathbf{p}_{j,RGBZ}$ is the corresponding 3D point (camera

coordinates) to the RGBZ-pixel $\mathbf{q}_{j,RGBZ}$, see equation 5.1.

$$r(\psi) = \mathcal{P}_{SLR}\left(\mathbf{K}_{SLR}\Theta\left(\mathcal{T}\left(\psi, \begin{bmatrix}\mathbf{R}_{SLR} & \mathbf{t}_{SLR}\\ 0^T & 1\end{bmatrix}\mathbf{p}_{j,RGBZ}\right)\right)\right) - \mathcal{P}_{Kinect}\left(\mathbf{q}_{j,RGBZ}\right)$$

$$\mathcal{E}(\psi) = \frac{1}{2}\sum_{j=1}^{n}\left(\mathcal{P}_{SLR}\left(\mathbf{K}_{SLR}\Theta\left(\mathcal{T}\left(\psi, \begin{bmatrix}\mathbf{R}_{SLR} & \mathbf{t}_{SLR}\\ 0^T & 1\end{bmatrix}\mathbf{p}_{j,RGBZ}\right)\right)\right) - \mathcal{P}_{Kinect}\left(\mathbf{q}_{j,RGBZ}\right)\right)^2 \tag{5.4}$$

Now $\mathcal{E}(\psi)$ represents the sum of squared intensity differences of all head pixel and minimizing equation 5.4 should yield a good alignment of the face-scan and the SLR-image.

A well known method to solve nonlinear least-squares problems is the Gauss-Newton algorithm. It uses the special structure of least-squares problems to provide better convergence properties of second order methods without the need to explicitly calculate the *Hessian*. Only the *Jacobian* of $r(\psi)$ and the value of $r(\psi)$ itself is needed to minimize $\mathcal{E}(\psi)$. In the following the calculation of the *Jacobian* is presented. The function $\mathcal{W}$ performs the reprojection of an RGBZ-pixel onto the SLR-image and is added only for notational reasons.

$$\mathcal{W}(\mathbf{p}, \psi) = \mathbf{K}_{SLR}\Theta\left(\mathcal{T}\left(\psi, \begin{bmatrix}\mathbf{R}_{SLR} & \mathbf{t}_{SLR}\\ 0^T & 1\end{bmatrix}\mathbf{p}\right)\right) \tag{5.5}$$

$$\frac{\partial r}{\partial \psi} = \frac{\partial}{\partial \psi}\mathcal{P}_{SLR}\left(\mathcal{W}(\mathbf{p}, \psi)\right) - \frac{\partial}{\partial \psi}\mathcal{P}_{Kinect}$$

Since $\mathcal{P}_{Kinect}\left(\mathbf{q}_i\right)$ does not depend on ψ its derivatives will always evaluate to zero and can therefore be omitted.

$$\frac{\partial}{\partial \psi}\mathcal{P}_{SLR}\left(\mathcal{W}(\mathbf{p}, \psi)\right) = \frac{\partial \mathcal{P}_{SLR}}{\partial \mathcal{W}}\frac{\partial \mathcal{W}}{\partial \psi}$$

The derivatives of $\mathcal{P}_{SLR}$ can be approximated by applying a Sobel filter on $\mathcal{P}_{SLR}$.

$$\frac{\partial \mathcal{P}_{SLR}}{\partial x} = \mathcal{P}_{SLR,dx} = \mathcal{P}_{SLR} * \mathcal{S}_x,\ \frac{\partial \mathcal{P}_{SLR}}{\partial y} = \mathcal{P}_{SLR,dy} = \mathcal{P}_{SLR} * \mathcal{S}_y$$

$$\mathcal{S}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \mathcal{S}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Here $\mathcal{P} * \mathcal{H}$ denotes a filter operation of image $\mathcal{P}$ with the kernel $\mathcal{H}$. The derivatives of $\mathcal{P}_{SLR}$ at pixel-coordinates $\mathbf{p}$ can be written as:

$$\frac{\partial}{\partial \mathcal{W}} \mathcal{P}_{SLR}\big|_{\mathbf{p}} = \begin{bmatrix} \mathcal{P}_{SLR,dx}(\mathbf{p}) \\ \mathcal{P}_{SLR,dy}(\mathbf{p}) \end{bmatrix}$$

Now Θ and $\mathcal{T}$ are the only factors that still depend on the parameter vector ψ. Applying the chain-rule on $\frac{\partial \mathcal{W}}{\partial \psi}$ yields:

$$\frac{\partial \mathcal{W}}{\partial \psi} = \frac{\partial \mathbf{K}_{SLR}}{\partial \Theta} \frac{\partial \Theta}{\partial \mathcal{T}} \frac{\partial \mathcal{T}}{\partial \psi}$$

For the implementation it is important to keep in mind that f_y has a negative sign because pixel coordinates increase actually from top to bottom.

$$\frac{\partial \mathbf{K}_{SLR}}{\partial \Theta} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & -f_y & c_y \end{bmatrix}$$

The derivative of the de-homogenization $\Theta(\mathbf{p})$ can be calculated as follows:

$$\frac{\partial \Theta}{\partial \mathcal{T}}\Big|_{\mathbf{p}} = \begin{bmatrix} \frac{1}{\mathbf{p_z}} & 0 & -\frac{\mathbf{p_x}}{\mathbf{p_z^2}} \\ 0 & \frac{1}{\mathbf{p_z}} & -\frac{\mathbf{p_y}}{\mathbf{p_z^2}} \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{p} \epsilon \mathbb{R}^3$$

The last row is zero, because $\Theta(\mathbf{p})$ returns always a vector with $\mathbf{p}_z = 1$.

Since $\mathcal{T}$ contains a rotation, it can not be linearized directly, but it is possible to find an approximate solution. As explained above the rotation is defined by three Euler angles. According to the law for small angles ($sin(\varphi) \approx \varphi$ and $cos(\varphi) \approx 1$ for $\varphi \approx 0$), equation 3.13 can be approximated as:

$$rot(\varphi) \approx \begin{bmatrix} 0 & -\varphi_3 & \varphi_2 \\ \varphi_3 & 0 & -\varphi_1 \\ -\varphi_2 & \varphi_1 & 0 \end{bmatrix}, \; with \; \varphi \approx 0^T$$

Following that, a small change in φ can be written as:

$$rot(\varphi + \triangle\varphi)x \approx rot(\varphi)x + \begin{bmatrix} 0 & -\triangle\varphi_3 & \triangle\varphi_2 \\ \triangle\varphi_3 & 0 & -\triangle\varphi_1 \\ -\triangle\varphi_2 & \triangle\varphi_1 & 0 \end{bmatrix} x$$

$$rot(\varphi + \triangle\varphi)x \approx rot(\varphi)x + \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \triangle\varphi \tag{5.6}$$

Using equation 5.6, $\frac{\partial \mathcal{T}}{\partial \psi}$ can be approximated by the following matrix:

$$\frac{\partial \mathcal{T}}{\partial \psi}\Big|_{\mathbf{p}} = \begin{bmatrix} 0 & -\mathbf{p_z} & \mathbf{p_y} & 1 & 0 & 0 \\ \mathbf{p_z} & 0 & -\mathbf{p_x} & 0 & 1 & 0 \\ -\mathbf{p_y} & \mathbf{p_x} & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now that all necessary derivatives are available, $\frac{\partial r}{\partial \psi}$ can be easily calculated:

$$\frac{\partial r}{\partial \psi}\Big|_{\mathbf{p}} = \frac{\partial \mathcal{P}_{SLR}}{\partial \mathbf{K}_{SLR}}\Big|_{\mathbf{p}_{warped}} \frac{\partial \mathbf{K}_{SLR}}{\partial \Theta} \frac{\partial \Theta}{\partial \mathcal{T}}\Big|_{\mathbf{p}_{aligned}} \frac{\partial \mathcal{T}}{\partial \psi}\Big|_{\mathbf{p}_{SLR}} \tag{5.7}$$

$$\mathbf{p}_{SLR} = \begin{bmatrix} \mathbf{R}_{SLR} & \mathbf{t}_{SLR} \\ 0^T & 1 \end{bmatrix} \mathbf{p}$$

$$\mathbf{p}_{aligned} = \mathcal{T}(\mathbf{p}_{SLR}, \psi)$$

$$\mathbf{p}_{warped} = \mathcal{W}(\mathbf{p}, \psi)$$

Finally, the *Jacobian* of $r(\psi)$ looks as follows:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial r}{\partial \varphi_1} & \frac{\partial r}{\partial \varphi_2} & \frac{\partial r}{\partial \varphi_3} & \frac{\partial r}{\partial t_x} & \frac{\partial r}{\partial t_y} & \frac{\partial r}{\partial t_z} \end{bmatrix} \tag{5.8}$$

$\mathbf{J}$ and $r(\psi)$ can now be used to create the normal equation (equation 3.37) of the current RGBZ-pixel $\mathbf{q}_{RGBZ}$, that is needed for the Gauss-Newton iteration.

5.2.2 Regularization

However, in some situations the optimization-process needs some guidance. Regularization is used when the optimization problem should not only consider some kind of error measure but also other factors. For example prior information is available and the final result should stay close to a given parameter configuration. Another typical situation is when smoothness constraints are added because similar objects should have similar parameter configurations. Then the objective function is extended with regularizing terms that are used to gain more control over the optimization process. Such an objective function could look like the following:

$$f(\mathbf{x}) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(\mathbf{x}) + \lambda\frac{1}{2}\sum_{i=1}^{n}\mathbf{x}_i^2$$

The second term of this objective function leads to a parameter vector $\mathbf{x}$ with values close to zero, because the squared norm of $\mathbf{x}$ is part of the optimization problem. Regularizing terms should be weighted by a scalar value λ that allows to determine their impact. That means a high value of λ causes the regularizer to be more important while a small λ gives more weight to the error measure. For the optimization of the head pose different regularizing terms are added. Firstly, the step-size of the parameter updates are constrained because the approximated *Jacobian* is only valid in a small neighborhood around the current pixel. Equation 5.9 restricts the step-size, with $\mathbf{p}^{GN}$ beeing the search direction of the Guass-Newton update and $\mathbf{I}$ beeing the identity matrix.

$$\mathbf{I}\mathbf{p}^{GN} = 0^T \tag{5.9}$$

Secondly, equation 5.10 constrains the final parameter vector to be close to the ICP solution. This is especially necessary because changes along the z-axis (depth) are less accurate than others, so that degree of freedom is more constrained than other parameters.

$$\mathbf{I}\mathbf{p}^{GN} = (\psi_{prior} - \psi) \tag{5.10}$$

The third regularizer (equation 5.11) is used to allow an independent optimization for the left and right part of the face. It was mentioned in subsection 5.2 that the left and right part of a face-scan are optimized at once to ensure that they can not drift away from each other. That is still true, but this constraint is relaxed in a way that allows slightly different motions for the left and right part of the face-scan. By doing so, a small misalignment between left and right part can be corrected. Therefore the parameter vector holds now two parameter sets.

$$\mathbf{Ip}^{GN} = \frac{1}{2}(\psi_{other} - \psi) \tag{5.11}$$

5.2.3 Implementation of the Gauss-Newton optimization

For the implementation of the head-pose refinement a well known approach called warping is used. An image warp establishes a geometrical relation between corresponding parts of images. In the simplest case, this could be a translation or a rotation but also more complex transforms like perspective projections or mesh based transformations are possible. In general, a warp (equation 5.12) is a parameterized function that defines an arbitrary mapping of one image onto another with $\mathbf{p}_{dst}$ being the coordinates of the destination image, $\mathbf{p}_{src}$ being the source coordinates and Ψ being the parameter vector that describes the transform.

$$\mathbf{p}_{dst} = \mathcal{W}(\mathbf{p}_{src}, \Psi) \tag{5.12}$$

Warp functions can be applied in forward or backward direction. Forward direction means that for a source pixel the destination coordinates are calculated and the source value has to be added to the destination image. Typically the destination coordinates are floating point values, this means the warped image can contain empty areas or aliasing effects if the insertion is not done carefully. If the inverse mapping is known it is much easier since the destination pixel has integral coordinates and the corresponding value in the source image can be simply obtained by sampling (e.g. bi linear, cubic). An image warp can not only be employed to perform a transformation. It can also be used as model that explains how a parameterized transformation affects an image. For example if depth values of an image are known a warp can be used to re-render the scene from a different viewpoint and inversely if an image from a different viewpoint is available the new camera pose can be inferred by estimating warp-parameter that re-produce the second image.

To refine the head-pose a function is defined that projects an RGBZ-image onto the SLR-image and calculates a pixel wise intensity difference over all head pixel, see equation 5.4. The part that reprojects an RGBZ-pixel can be expressed as image warp, see equation 5.5. That means, the refinement of the head-pose can be expressed as a warp-optimization problem which can be solved by a general warp-optimization-framework.

In this work, the CVCGWarpLib is used to run the warp-optimization. It provides interfaces to optimize an arbitrary warp-function and to add different regularizers to the optimization process. The class diagram in figure 5.7 displays only the relevant relations of the framework, since a detailed description would go beyond the scope of this work. The *WarpOptimizer* runs an optimization loop until a stopping criterion is met. Every warp- and regularizer-object contributes normal equations to a linear equation system that is solved by the *GaussNewtonStep* in order to perform the parameter update as explained in subsection 3.5.

The classes *WarpRGBZImage* and *StepRegularizerRGBZImage* are implementations of *AbstractWarp* and *AbstractRegularizer* respectively. *WarpRGBZImage* implements an image-warp that re-renders an object (the subject's head) from a different viewpoint. Therefore it implements two important methods (*compute(...)* and *normalEquations(...)*).

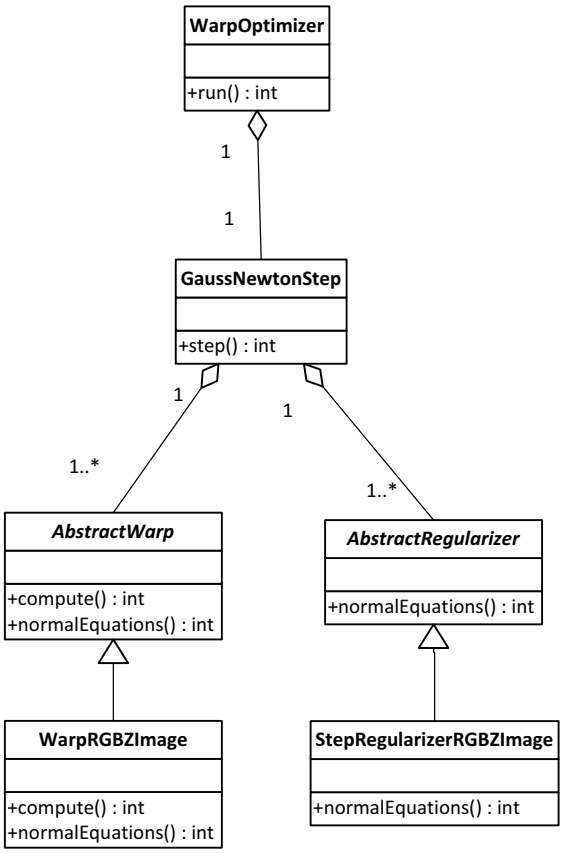


Figure 5.7: UML-Diagram of relevant classes

Compute(...) builds a look-up-table that contains floating point source coordinates for every pixel in the destination image. By performing a re-map operation the warp function can then be efficiently applied to images. The second important method, *normalEquations()* is used during the optimization to add a normal equation for every face pixel to a system of linear equations (equation 3.37) which is solved in order to calculate the parameter-update for the Gauss-Newton iteration. Based on equation 5.7 the *Jacobian* can be approximated at every RGBZ-pixel of a face-scan. Since the corresponding residuals are provided by the warping-framework, the normal equation for every RGBZ-pixel of the face-scan can be easily calculated by inserting into equation 3.37.

The left and right part of the face-scan are processed simultaneously with two separate *WarpRGBZImage*-objects but they share the same parameter vector ψ. The advantage

of this approach is that the relative pose of both face-scans can be improved while a regularizer (equation 5.11) is used to keep both parts close together.

Every regularizer contributes additional normal equations (equations 5.9, 5.10 and 5.11) but their impact can be controlled by a scalar weight. After all Warp and Regularizer objects have finshed, the update step can be calculated by equation 3.38 and the next iteration starts. An important fact that needs to be considered for the implementation is that the Kinect deliveres colour images at a resolution of 640×480 while the SLR-images have a resolution of several megapixel. That means, an SLR-image contains much more details (i.e. edges), it is much sharper and has different brightness distribution, which makes it difficult to run a pose estimation based on intensity differences. To cope with that, the SLR-images are smoothed and scaled before they are used for warp-optimization. The next section explains the used method to adjust the image brightness.

5.2.4 Brightness constancy constraint

One condition for the warp-optimization to work properly is that corresponding pixel in source and destination image have an equal or at least similar brightness. Correspondence means in this context, that the pixel depict the same part of the subject's face. This is especially important since the warp-optimization is performed between images from different cameras. The destination image is captured by the Kinect RGB-camera and the source image is taken by a Canon EOS 550D. Both images have different intensity distributions caused by a different aperture size, exposure time, point of view and of course image sensor. To make both images look more similar, a global and local brightness correction is performed. Figure 5.8 and 5.9 show the effect of the brightness correction.

However, the brightness correction ignores the fact that RGBZ-image and SLR-image are not perfectly aligned. This means that a correspondence between an RGBZ-pixel and a SLR-pixel is established simply by re-projection as described in section 3.1.2.

Firstly, two normal-distributions $\mathcal{N}(\mu_{Kinect}, \sigma^2_{Kinect})$ and $\mathcal{N}(\mu_{SLR}, \sigma^2_{SLR})$ that model the intensity distributions of all corresponding pixel in the SLR- and RGBZ-image are calculated. Then, the intensities of the adjusted image are corrected as follows:

$$I_{corr} = \frac{(I_{old} - \mu_{ref})\,\sigma_{ref}}{\sigma_{old}} + \mu_{ref}$$

Now, a local brightness correction is performed based on the average intensity difference between the neighbourhoods of corresponding pixel. To ensure that it works also near strong edges (e.g. hair, eye or mouth) a weighted mean is used. The weighting function is similar to the probability density of a normal distribution $\mathcal{N}(\mu, \sigma^2)$.

$$I_{corr} = I_{old} + (avg_{ref}(I_{old}) - avg_{old}(I_{old}))$$

$$avg(I_0) = \frac{\sum_i I_i \alpha(I_i, I_0)}{\sum_i \alpha(I_i, I_0)}$$

The function $avg(I_0)$ calculates the weighted mean intensity of a square image patch around a pixel of interest with intensity I_0. The weighting function $\alpha(I, I_0)$ reduces the impact of pixel that are very dissimilar to the adjusted pixel.

$$\alpha(I_i, I_0) = exp\left(-\frac{(I - I_0)^2}{\sigma^2}\right)$$

For the choice of the neighbourhood it is important to consider the expected alignment error to ensure that enough similar pixel are found in the reference image.

Figure 5.8: Source and destination image before intensity adjustment

Figure 5.9: Source and destination image after intensity adjustment

5.3 Fusion of face-scans

After the warp-optimization, all RGBZ-images are assumed to be aligned with the SLR-image and a new improved depth-map can be calculated. As explained in the beginning, the raw depth-maps suffer from noise, vertical banding and discretization errors. Also, a single depth-map captured by a Kinect does not depict the complete face and can contain holes when the person is very close. Noise, discretization errors and vertical banding can be reduced by averaging many scans. Vertical banding, however, does actually not vary over time, but the vertical lines vanish in the final model since all depth scans are taken from different points of view (w.r.t. the subject's head).

In order to merge the aligned face-scans they are projected onto a virtual camera using a triangle mesh that connects neighbouring pixel as explained in [35]. The virtual camera has the same pose as the SLR-camera but different intrinsics since the resulting depth-map has a lower resolution compared to the texture. The re-projected depth-maps are then averaged on a per pixel basis. Since two Kinects are operated simultaneously the projection of both IR-speckle patterns causes sometimes heavy outliers that lead to artifacts in the final model. To reduce the impact of such outliers depth-values that are more than a certain threshold δ_{depth} away from the median are ignored. To calculate a reasonable value for δ_{depth} the mad (median absolute deviation) is used since it is a robust and consistent estimator for the standard deviation σ of a normally distributed variable [37].

$$mad(X) = median\left\{|X_i - median(X)|\right\}$$

$$\hat{\sigma} \approx 1.4826 * mad$$

$$\delta_{depth} = 3\hat{\sigma}$$

Finally, the new depth-map is smoothed using the Guided Filter [20]. The Guided Filter performs an edge-aware image smoothing operation similar to the Bilateral Filter [43]. A big advantage of the Guided Filter is that it is computationally less expensive. To keep the flattening effect as low as possible only a small filter radius is used.

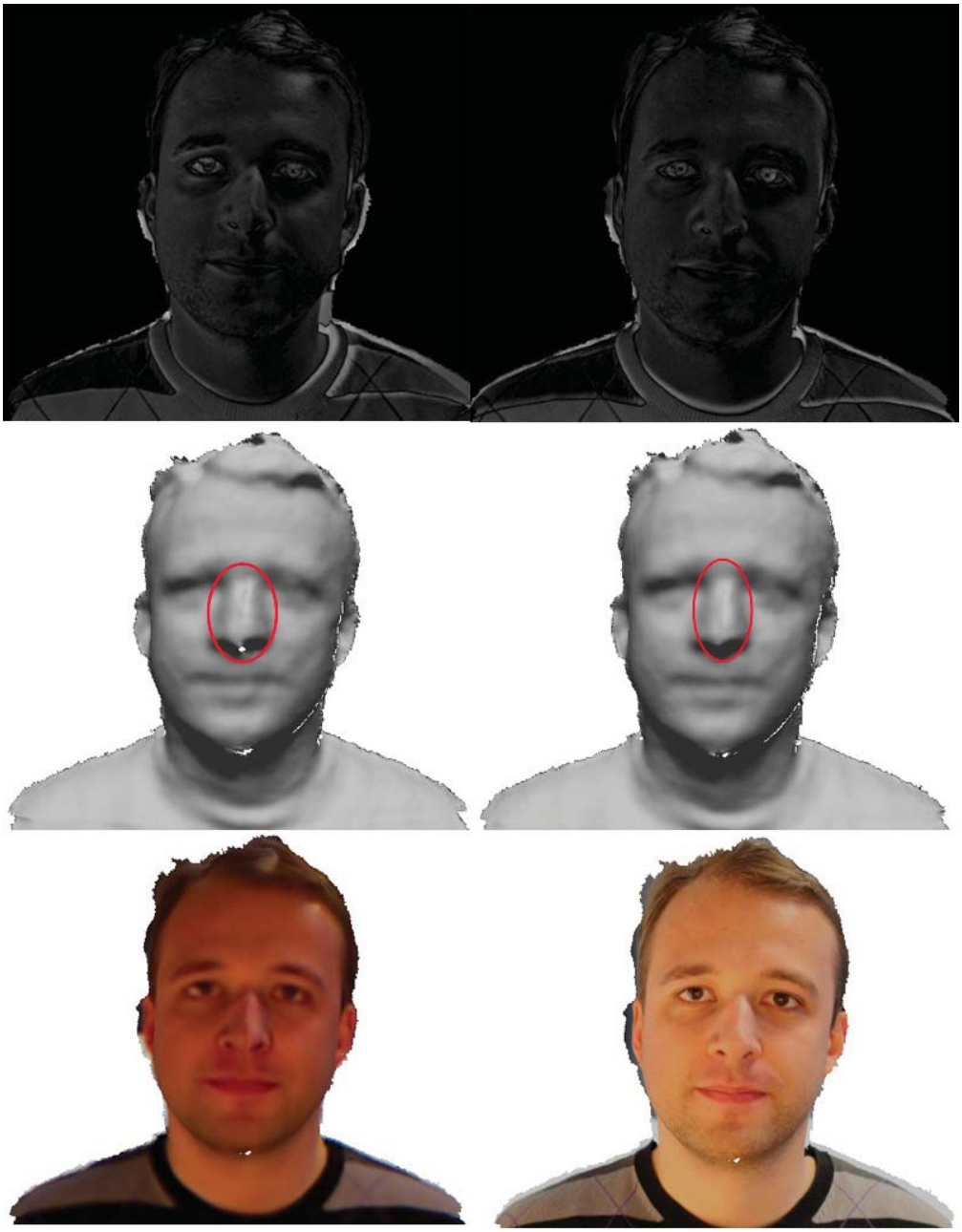
Since depth-values are only calculated for face-pixel that are visible in the SLR-image, it is sufficient to fuse all face-scans into one new depth-map, because only 3D-regions with colour-information are of interest. For more complex models or when a more complete model of the human head is needed other approaches would be more suitable. For example the usage of cylindrical depth-maps as described in [28]. As the name suggests 3D-points are not projected onto a planar depth image, but on a cylinder that is centred around the object of interest. This way also star-shaped-objects can be represented correctly.

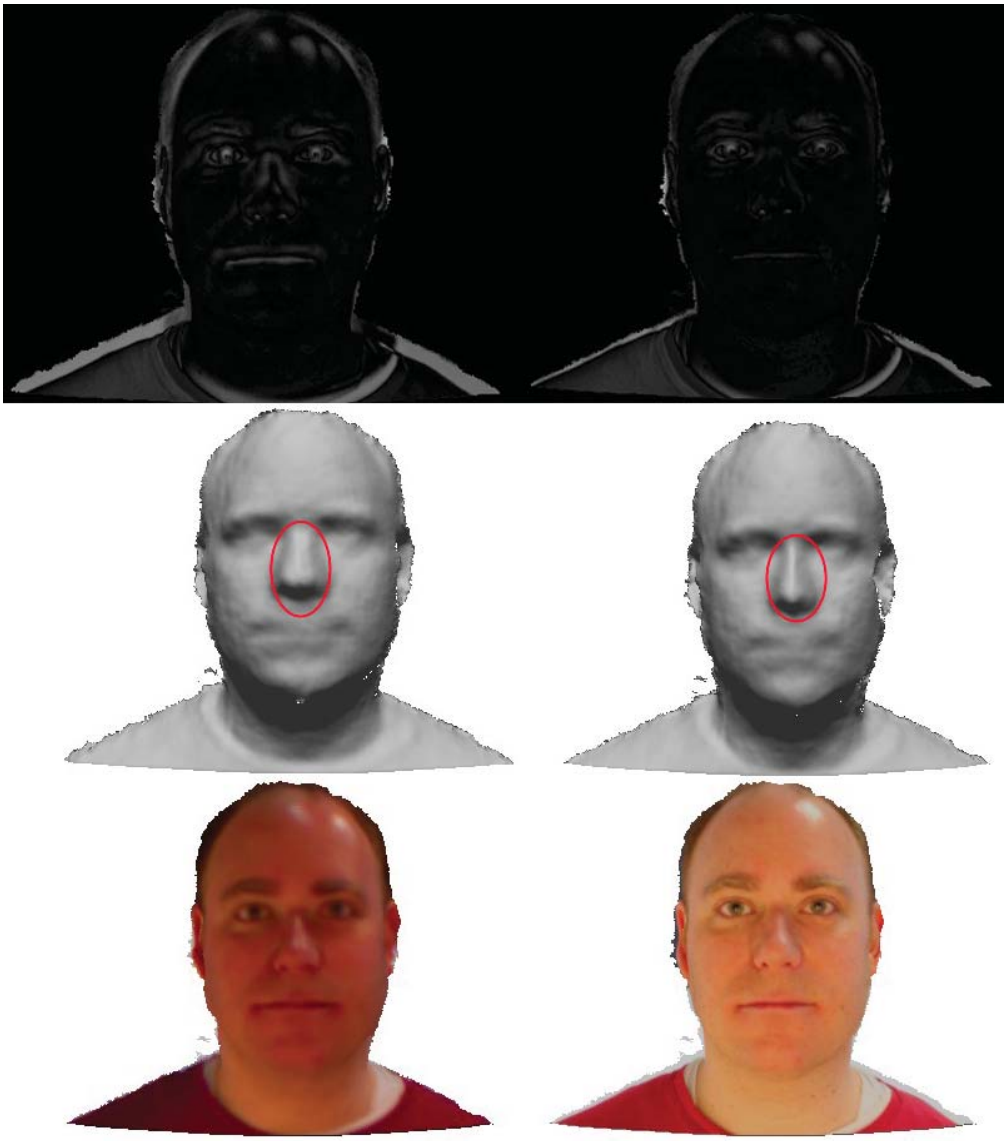
If depth scans of arbitrary objects or scenes need to be fused, a volumetric approach like [14] should be considered. Curless et al. use a TSDF (truncated signed distance function) to represent the surface of an object inside a scanned volume. The scanned volume can be for example represented by a 3D-grid. Every voxel of the grid holds the value of the TSDF at that position. A negative value corresponds to a voxel that is inside the object (or behind the surface) whereas a positive value denotes a voxel that is outside the object (or in front of the surface). The surface is located exactly at the zero-crossings of the TSDF. Before the reconstructed object can be manipulated or rendered as triangle mesh, all vertices need be to be extracted from the TSDF. To do so, Curless et al. for example employs a Marching Cube algorithm [26, 29] to extract the triangle-mesh.

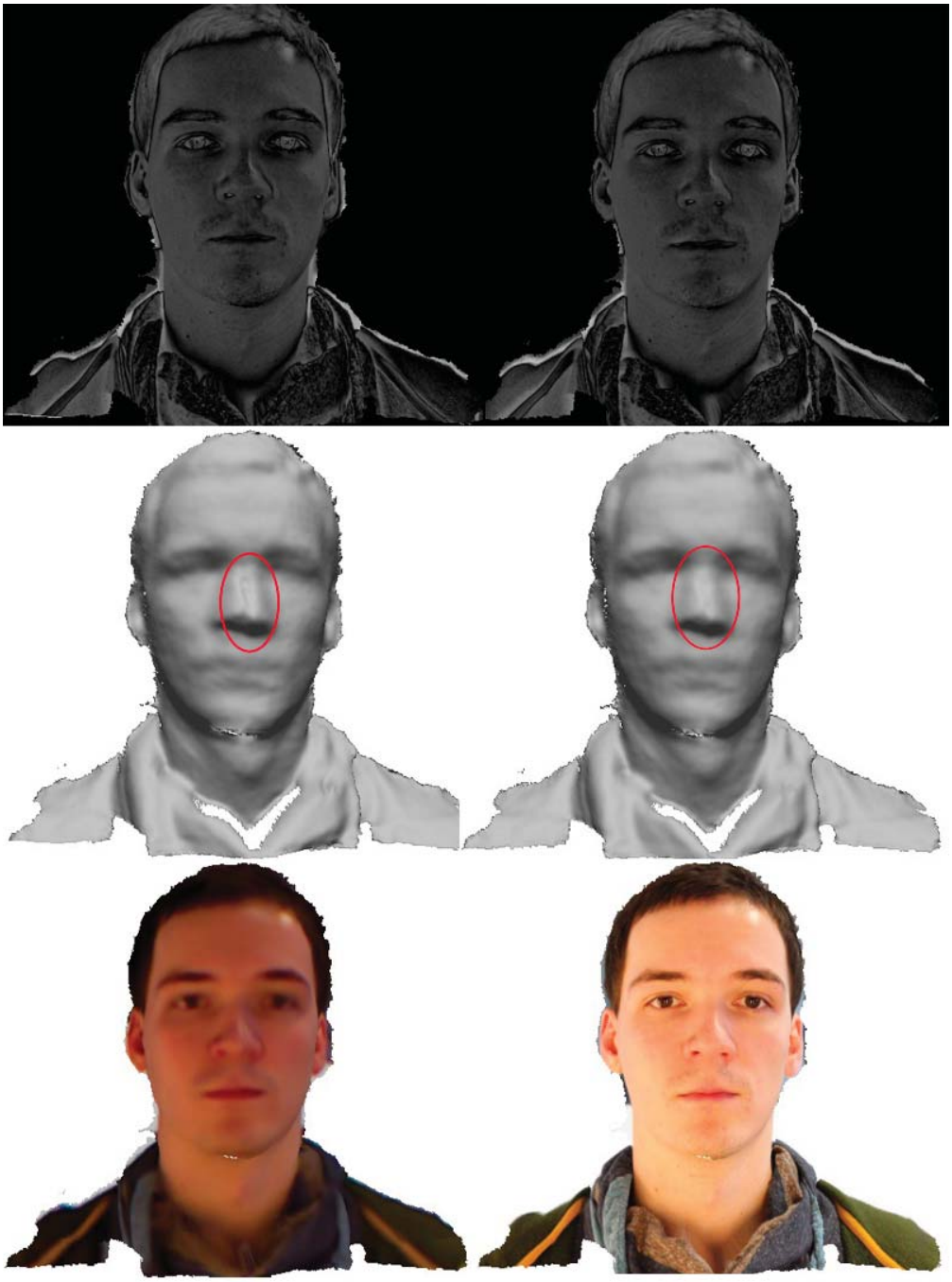
6 Results

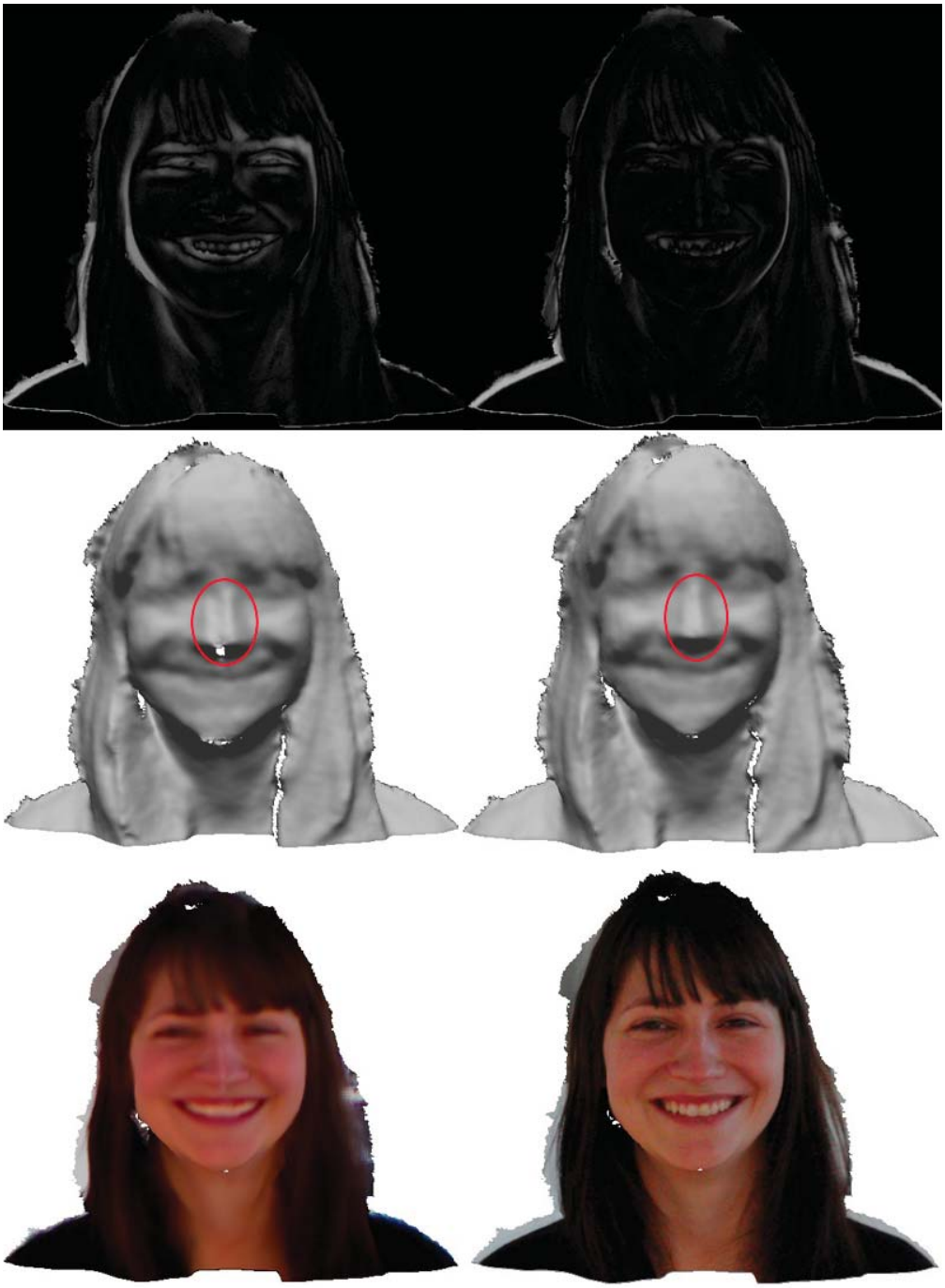
This section presents results of the reconstruction process. The left column shows to the reconstruction after the ICP-based alignment and the right column depicts the result after the warp-based optimization.

The first row shows the mean difference image between the SLR- and Kinect-texture. After the ICP-based alignment the model can still be about a half frame off which is clearly visible in the difference images. The second row depicts the rendered triangle-mesh after the ICP based registration and after the Warp-based optimization. In some cases a misalignment can be seen in the nose region which got finally corrected by the warp based alignment. However, there are also cases where the displacement is still visible. The last row shows the mesh textured with the Kinect-image on the left and with the SLR-image on the right side.









7 Conclusion and future works

This work shows how data from low-cost RGBZ-cameras (e.g. Kinect) can be combined with textures from high quality SLR-cameras to create a realistic looking 3D-model of a human head.

Therefore a system was developed that automatically selects suitable data from an RGBZ-stream and generates an improved model of the subject's face by temporal integration and spatial smoothing. Depth-scans from two RGBZ-sensors are fused to reduce negative effects like noise, discretization errors or other artifacts. A two-stage synchronization method was implemented to ensure that corresponding depth-scans from different sensors are almost correctly aligned before starting the actual alignment-optimization. This eases the final alignment since corresponding scans can be aligned at once which improves the alignment quality of the warp-based alignment since it constrains the optimization problem and leads to more reasonable results. However, the warp-based alignment solves a non-linear optimization problem and needs initially a good starting point to converge to a correct solution. Therefore, the ICP-algorithm is employed to pre-align all used face-scans to a dedicated reference scan. All face-scans are aligned to a single reference to ensure that small alignment errors are not propagated as it can happen with pairwise registration of consecutive face-scans. The final alignment is performed by a warp-optimization that aligns all RGBZ-images with the SLR-texture. Finally, all aligned face-scans are projected onto the image plane of the SLR-camera and averaged on a per-pixel basis. Depending on the walking speed of a subject, the number of useable face-scans can be low, so an edge-preserving smoothing is performed on the depth-map to obtain a smoother 3D-model.

Section 6 shows results of the reconstruction. The improved geometry is smoother and contains more details than the raw data. Additionally, the original texture was replaced by an SLR-image to create a more realistic impression.

However, one drawback of the current implementation is the consumed processing time. Currently the reconstruction of a face takes about two minutes. While the pre-processing and the ICP based registration takes about 20 seconds, the most time consuming part is the warp-based alignment. So this part could be optimized. Nevertheless, the combination of ICP and warp-optimization leads to a good alignment, but it is not perfect. The estimation of the head-pose along the z-axis is less accurate than other optimized parameters and the difference-images show that there are still small misalignments between the Kinect- and SLR-texture. This might be caused due to the fact that, the warp-optimization is performed on images that origin from different camera types. Another problem is that the Kinect does not support any kind of synchronization. That includes synchronization between different Kinects as well as the internal synchronization of the colour- and depth-camera. This can degrade the final pose-estimation as well since it

leads to small missalignments between a depth-map and its corresponding colour-image, but it can be avoided by using for example a different sensor.

Acknowledgements

Hereby I want to thank all people who supported me during my studies and the work on this thesis. Especially I want to thank my family who made my education possible and Diana for the proof reading and support. A big thank goes out to the image-processing team at Heinrich Hertz Institute for several useful hints, especially David for many advises and ideas. Finally, I want to thank Prof. Rojas and Prof. Eisert for supervising my thesis.

Declaration of academic honesty

Hereby I declare that I have produced this thesis entitled "*Acquisition of 3D-Head-Models using SLR-Cameras and RGBZ-Sensors*" without the prohibited assistance of third parties and without making use of aids other than those specified. Notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other German or foreign examination board. This thesis work was conducted from September 2012 to April 2013 under the supervision of David Blumenthal-Barby at the Fraunhofer Heinrich Hertz Institute, Berlin.

Wolfgang Paier

Berlin, 23 April 2013

List of Figures

Bibliography

[1] Accuracy analysis of kinect depth data. *ISPRS Workshop Laser Scanning*, 38(5/W12).

[2] Opencv. `http://opencv.willowgarage.com`.

[3] Openni. `http://www.openni.org/`.

[4] Pointcloud library. `http://pointclouds.org/`.

[5] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Richard Szeliski. Building rome in a day. *Commun. ACM*, 54(10):105–112, 2011.

[6] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, September 1987.

[7] Thabo Beeler, Bernd Bickel, Paul Beardsley, Bob Sumner, and Markus Gross. High-quality single-shot capture of facial geometry. *ACM Trans. on Graphics (Proc. SIGGRAPH)*, 29(3):40:1–40:9, 2010.

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

[9] Gerard Blais and Martin D. Levine. Registering multiview range data to create 3d computer objects. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 17:820–824, 1993.

[10] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[11] Amit Bleiweiss and Michael Werman. Robust head pose estimation by fusing time-of- flight depth and color, 2010.

[12] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. pages 2724–2729, 1991.

[13] D. Chetverikov, D. Svirko, D. Stepanov, and Pavel Krsek. The trimmed iterative closest point algorithm. In *In International Conference on Pattern Recognition*, pages 545–548, 2002.

[14] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.

[15] Gabriele Fanelli, Juergen Gall, and Luc J. Van Gool. Real time head pose estimation with random regression forests. In *CVPR*, pages 617–624. IEEE, 2011.

[16] Barak Freedman, Alexander Shpunt, Meir Machline, and Yoel Arieli. Depth mapping using projected patterns, 2009.

[17] Natasha Gelfand, Leslie Ikemoto, Szymon Rusinkiewicz, and Marc Levoy. Geometrically stable sampling for the ICP algorithm. In *Fourth International Conference on 3D Digital Imaging and Modeling (3DIM)*, October 2003.

[18] Chris A Glasbey and Kanti V Mardia. A review of image-warping methods. *Journal of applied statistics*, 25(2):155–171, 1998.

[19] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[20] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. In *Proceedings of the 11th European conference on Computer vision: Part I*, ECCV'10, pages 1–14, Berlin, Heidelberg, 2010. Springer-Verlag.

[21] Berthold K. P. Horn. Closed-Form Solution of Absolute Orientation Using Unit Quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, April 1987.

[22] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.

[23] Andrew Johnson and Sing Bing Kang. Registration and integration of textured 3-d data. In *IMAGE AND VISION COMPUTING*, pages 234–241, 1996.

[24] DxO Labs. Lens distortion. `http://www.dxo.com/en/photo/dxo_optics_pro/features/optics_geometry_corrections/distortion`.

[25] Hai-Bin Liao, Qing-Hu Chen, Qian-Jin Zhou, and Lin Guo. Rapid 3d face reconstruction by fusion of sfs and local morphable model. *J. Vis. Comun. Image Represent.*, 23(6):924–931, August 2012.

[26] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[27] Sakaue K. Masuda, T. and N. Yokoya. Registration and integration of multiple range images for 3-d model construction. In *CVPR*, 1996.

[28] Jongmoo Choi Matthias Hernandez and Gerard Medioni. Laser scan quality 3-d face modeling using a low-cost depth camera.

[29] Claudio Montani, Riccardo Scateni, and Roberto Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *The Visual Computer*, 10(6):353–355, 1994.

[30] P. J. Neugebauer. Geometrical cloning of 3d objects via simultaneous registration of multiple range images. In *Proceedings of the 1997 International Conference on Shape Modeling and Applications (SMA '97)*, SMA '97, pages 130–, Washington, DC, USA, 1997. IEEE Computer Society.

[31] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer series in operations research and financial engineering. Springer, New York, NY, 2. ed. edition, 2006.

[32] Oline Vinter Olesen, Rasmus R. Paulsen, Liselotte HÃžjgaard, Bjarne Roed, and Rasmus Larsen. Motion tracking for medical imaging: A nonvisible structured light tracking approach. *IEEE Trans. Med. Imaging*, 31(1):79–87, 2012.

[33] J. A. Paterson and A. W. Fitzgibbon. 3d head tracking using non-linear optimization. In *Proceedings of the British Machine Vision Conference 2003*, volume 2, pages 609–618, September 2003.

[34] PrimeSensor. Preime sensor nite 1.3 algorithms notes. `http://pr.cs.cornell.edu/humanactivities/data/NITE.pdf`.

[35] Christian Richardt, Carsten Stoll, Neil A. Dodgson, Hans-Peter Seidel, and Christian Theobalt. Coherent spatiotemporal filtering, upsampling and rendering of RGBZ videos. volume 31, May 2012.

[36] ROS.org. Technical descripton of kinect calibration. `http://www.ros.org/wiki/kinect_calibration/technical`.

[37] D. Ruppert. *Statistics and data analysis for financial engineering*. Springer texts in statistics. Springer New York, 2011.

[38] Szymon Rusinkiewicz and Marc Levoy. Efficient Variants of the ICP Algorithm. In *Proceedings of the Third Intl. Conf. on 3D Digital Imaging and Modeling*, pages 145–152, 2001.

[39] Joaquim Salvi, Sergio Fernandez, Tomislav Pribanic, and Xavier Llado. A state of the art in structured light patterns for surface profilometry. *Pattern Recogn.*, 43(8):2666–2680, August 2010.

[40] David C. Schneider, Markus Kettern, Anna Hilsmann, and Peter Eisert. A global optimization approach to high-detail reconstruction of the head. In *VMV*, pages 9–15, 2011.

[41] David A. Simon, Martial Hebert, and Takeo Kanade. Real-time 3-d pose estimation using a high-speed range sensor. In *in Proceedings of IEEE International Conference on Robotics and Automation (ICRA 94*, pages 2235–2241, 1994.

[42] C. C. Slama, C. Theurer, and S. W. Henriksen, editors. *Manual of Photogrammetry*. American Society of Photogrammetry, 1980.

[43] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–, Washington, DC, USA, 1998. IEEE Computer Society.

[44] Marc Toussaint. Lecture notes: Some notes on gradient descent. `http://userpage.fu-berlin.de/mtoussai/notes/gradientDescent.pdf`.

[45] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 311–318, New York, NY, USA, 1994. ACM.

[46] John Vince. *Introduction to the Mathematics for Computer Graphics*. Springer-Verlag, Berlin, Heidelberg, 3rd edition, 2010.