FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Combining Multiple Depth Cameras for Reconstruction

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Computergraphik und Digitale Bildverarbeitung

eingereicht von

Katharina-Anna Wendelin
Matrikelnummer 0425160

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Ass. Mag.rer.nat. Dr.techn. Hannes Kaufmann

Wien, 12.11.2012

(Unterschrift Verfasserin)

(Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Combining Multiple Depth Cameras for Reconstruction

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Visual Computing

by

Katharina-Anna Wendelin
Registration Number 0425160

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Ass. Mag.rer.nat. Dr.techn. Hannes Kaufmann

Vienna, 12.11.2012

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasserin)

Danksagung

Zunächst möchte ich mich bei meinem Betreuer Hannes Kaufmann bedanken, der sehr viel Geduld mit mir hatte und mir immer mit Rat und Tag zur Seite gestanden ist. Auch Christian Schönauer, Annette Mossel und Ingo Schiller möchte ich für ihre Hilfe bei der Kalibrierung der Kameras meinen Dank aussprechen.

Meinen Freunden, meinen Studienkollegen und meiner Familie möchte ich dafür danken, dass sie immer an mich geglaubt haben und mich in schwierigen Zeiten motiviert haben.

Der größte Dank geht an meinen Verlobten, der mich während der gesamten Studienzeit unterstützt hat, meine Launen ertragen musste und mich immer motiviert hat, weiter zu machen. Ich danke dir Andi, ohne dich wäre das alles nicht möglich gewesen.

Acknowledgements

First of all I want to thank my advisor Hannes Kaufmann who was very patient and helped me with words and deeds. Also I want to thank Christian Schönauer, Annette Mossel and Ingo Schiller who helped me during the calibration process.

Further more I also want to thank my friends, my colleagues and my family for always believing in me.

Special thanks go to my fiancé who supported me throughout the whole time at university, had to bear my mood and always motivated me. I thank you Andi, without you this all would not have happened.

Abstract

In the past few years depth cameras and their applications have gained more and more attention. Especially the publication of the Microsoft Kinect which is a cheap alternative to the expensive industrial cameras has initiated the research and development in this area. Adding depth information to images makes it possible to develop a lot of different application areas. Gesture and motion recognition, 3D Reconstruction of people and objects and the analysis of work movements are just a few possibities for using the depth camera technology. This work treats the combination of depth cameras in order to support future work like 3D reconstruction. The configuration of the cameras, external factors and camera calibration have to be considered. Different techniques for 3D image generation are described and analized and similar publications with various approaches are explained and evaluated. Subsequently the practical part of this work is explained covering the combination of two different depth cameras and the challenges which arrise out of it.

Kurzfassung

In den letzten Jahren haben Tiefenbildkameras und ihre Anwendungsgebiete immer mehr an Bedeutung gewonnen. Vor allem durch die Veröffentlichung der Microsoft Kinect, die eine kostengünstige Alternative zu den teuren industriellen Kameras darstellt, hat die Forschung mit Tiefenbildkameras angeregt. Die Anreicherung von Bildern mit Tiefeninformation öffnet die Tür für eine große Anzahl von Anwendungsgebieten. Erkennung von Gesten und Bewegungen, 3D Rekonstruktion von Menschen und Objekten und Analyse von Bewegungsabläufen sind nur einige der vielen Möglichkeiten, die sich durch diese Technologie ergeben. Diese Arbeit beschäftigt sich mit der Kombination von Tiefenbildkameras, um zukünftige Arbeiten wie zum Beispiel die 3D Rekonstruktion von Menschen und Objekten zu ermöglichen. Dabei sind einige Faktoren wie die Beschaffenheit der Kameras, die Einwirkung von äußerlichen Faktoren und die Kalibrierung der Kameras aufeinander zu beachten. Es werden die unterschiedlichen Verfahren, 3D Bilder zu erzeugen, beschrieben und analysiert und ähnliche Arbeiten mit verschiedenen Herangehensweisen zu diesem Thema geschildert und evaluiert. Anschließend wird der praktische Teil dieser Arbeit beschrieben, der sich mit der Zusammenführung zweier unterschiedlicher Tiefenbildkameras und die Herausforderungen, die sich dadurch ergeben, beschäftigt.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Object Capturing with Depth cameras

When using multiple depth cameras the same challenges rise as using conventional two dimensional cameras. Every camera is constructed in a different way and in order to be able to capture data the different systems have to be aligned to each other. In order to achieve that every camera has to be calibrated in respect to all other cameras used in the system. Calibration is a comprehensive task that includes mathematical practices and image processing techniques. An accurate calibration however is not solely a guarantor for good image quality. External factors like illumination or the setting of the area the capturing takes place at can make the captured images useless.
When using depth cameras the same aspects as mentioned before have to be considered. Additionally every source of interference regarding the process of capturing three dimensional data has to be taken into account. In this case the construction of the depth camera determines the possible difficulties. When using different types of depth cameras the different sources of errors have to be considered as well as the different ways of capturing images.
This work explains the steps that have to be made in order to be able to capture three dimensional data with multiple depth cameras. The chapter „Related Work“ explains the technological background and takes into account the process of image capturing and the transformation into three dimensional space. After that different methods for retrieving three dimensional data are presented. At the end of the chapter „Related Work“, different types of object capturing technologies are presented. The next chapter explains the core of the work treating the setup of a rig consisting of two different types of depth cameras in order to capture a point cloud. In this chapter the hardware and software setup are explained as well as the calibration of the intrinsic and extrinsic camera parameters. Also methods like background segmentation and noise reduction are explained in order to increase image quality. The chapter „Implementation“ explains the structure of the practical part of this work. It covers the implementation of the data acquisition from the different cameras, the handling of the calibrated data, the enhancement of the retrieved data and the output of the merged views. Subsequently the results of the before mentioned

steps are presented and evaluated in chapter „Results“. The last chapter „Conclusion and Future Work“ evaluates possible enhancement of the data concerning real-time applications and treats the possible application fields of the resulting point cloud. The chapter „Appendix“ includes the explanation of mathematical operations used during this work.

1.2 Application Fields using Multiple Depth Cameras

The use of depth cameras is wide spread and reaches from application fields like object reconstruction for computer graphics, 3D interaction, visualization for medical purposes to preservation of cultural heritage. In all these application fields the reconstruction of the real world is the principal task. With the use of reconstructed three dimensional data it is possible for the user to receive new possibilities of interaction. Using only a single depth camera has the disadvantage of only delivering a single view at one time. So when trying to reconstruct the real world either the camera has to be moved or the object has to be moved. This limits the field of application to the reconstruction of only static scenes because movement that happens beyond the field of view can not be captured. In the following possible application fields are described with attention turned on the possible benefit of using multiple depth cameras.

3D interaction

3D interaction has become a very important application field in the past few years. Especially in the gaming industry where the Microsoft Kinect has revolutionized the interaction between the user and the game by making the traditional controller obsolete thus enhancing the gaming experience. The use of a single camera limits the user to a specific field of view. Gestures that are performed beyond that view can not be recognized. As well as the field of view, occlusions impair the result of the captured person. The use of multiple depth cameras would make it possible to resolve this limitations. The person captured could use the whole room to perform the gestures without having to care about the position of the camera or the field of view and would not have to worry about occlusions.

Applications in Medicine

The use of 3D data in medicine makes it possible to observe visual information in an intuitive way. The use of multiple depth cameras for motion analysis increases the process of diagnosis and therapy. With the use of depth cameras it is possible to observe a patient in an non-invasive way which is a benefit for the observed person.

Reconstruction of the Environment

Reconstruction of the environment has many application fields in daily life. The estimation of distances and the identification of objects are the main tasks that can be fulfilled when using depth cameras. These tasks are very important for application fields like robotics or the car industry. The possibility to reconstruct the environment in order to overcome an obstacle or to prevent accidents are important fields of research.

3D reconstruction

In order to capture not only a static scene but a course of motion the combination of multiple cameras is needed. The use of depth cameras makes it possible to identify the foreground objects faster than using the conventional 2D CCD (charged coupled device) cameras. The depth information can be used to make a prior selection of the area of interest and so drop non relevant information in an early step of the capturing process. This work should serve as preparatory work for 3D reconstruction facing the challenges when combining multiple depth cameras.

CHAPTER 2

Related Work

2.1 Technological Background

Overview

The following section covers information about prerequisites and different fields of applications of depth cameras. First of all the different kind of projections are discussed leading to active and passive 3D vision methods as a field of application. As this work covers the active methods the emphasis lies on different technologies in that area and the mathematical background. Last different approaches in view independent motion capturing systems will be discussed covering 2D, 3D and hybrid methods using either one or multiple cameras.

The Computer Vision Pipeline

This section will give a short overview about different projection methods and the application field explaining mathematical backgrounds and therefor the prerequisites for image capturing. The geometric relationships as well as optical principles are described being a precondition for 3D Vision in general. The underlying theory for projections in general is called the pinhole model [FLP01]. It describes the relationship between the captured scene in 3D and the corresponding mapping in 2D. All captured object points can be mapped onto the image plane by rays that pass the center of projection. The pinhole model assumes that there is always only one ray for each object point that leads to the corresponding projection. So a pinhole camera consists of an optical center, the image plane and the focal length which is the distance between the optical center and the image plane [FLP01]. The projection which is described by the pinhole camera model is called perspective projection or central projection and is described in the following.

Perspective Projection

The perspective projection or central projection is a projection of the 3D world on a 2D surface generated by rays that go through a common point called the center of projection [Sab08]. The mapping from 3D to 2D is defined by the equation for the central projection [Sch05]:

$$\frac{x_w}{x} = \frac{y_w}{y} = \frac{z}{f} \tag{2.1}$$

Where the point (x_w, y_w, z_w) represents the world coordinates, the point (x, y, z) represents the image coordinates and f is the focal length. The index w denotes that the coordinates are represented in homogeneous coordinates (see 6.3). With this relationship the mapping done in the central projection can be written like the following equation:

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w \end{bmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

The relationship between world coordinates and image coordinates is explained in figure 2.1. The points $P1 = (x_{1w}, y_{1w}, z_{1w})$ and $P2 = (x_{2w}, y_{2w}, z_{2w})$ that lie on the captured object are mapped onto the image plane and result in the points $p1 = (x_1, y_1, z_1)$ and $p2 = (x_2, y_2, z_2)$. The ray captured by the camera goes through the center of the camera $O = (i, j, k)$. The intersection point between the image plane and the optical axis is called principal point c. The focal length (the distance between the image plane and O) is denoted as f in the figure.

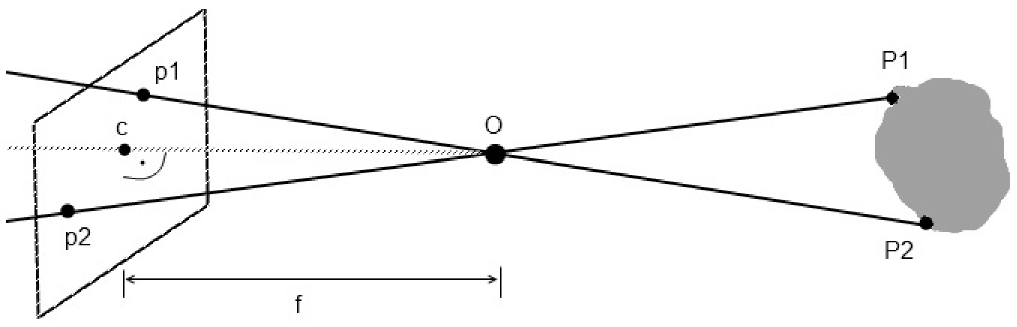


Figure 2.1: Perspective Projection

Orthographic Projection

As a special case of the perspective projection, the orthographic projection can be approximated by assuming that the focal length equals infinity. This is the case when objects that are far away from the optical center are captured. This projection is called orthographic or parallel projection because only the x and y values of a captured object are taken into account. So the mapping can be formulated as followed [Sch05]:

$$x = x_w \quad \text{and} \quad y = y_w \tag{2.2}$$

Orthographic projections can be used for aerial photographs or cartography. Figure 2.2 shows a parallel projection where every optical ray is perpendicular to the image plane.

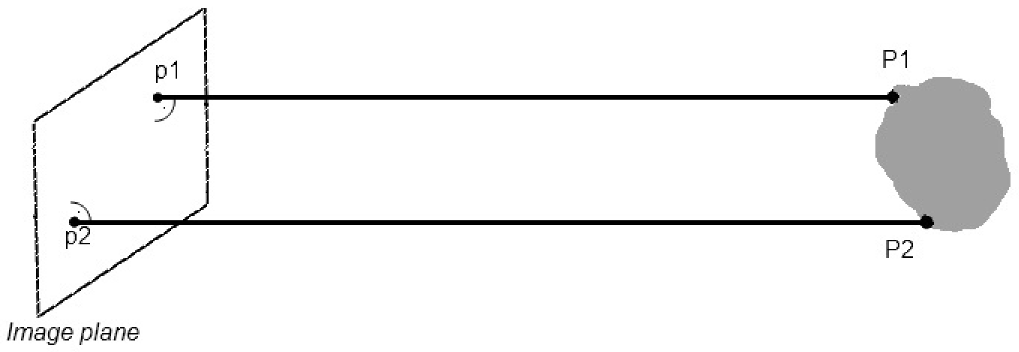


Figure 2.2: Orthographic Projection

In order to get the pixel coordinates of a point in the world reference frame, several transformation steps have to be performed [Sch05]. Figure 2.3 shows the steps of the viewing pipeline. In the following every single step is described briefly.

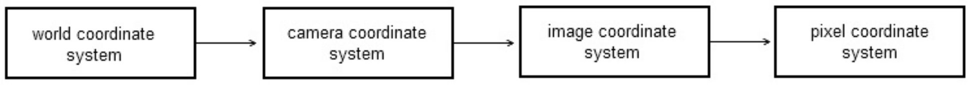


Figure 2.3: The computer vision pipeline

First of all the so called external transformation must be computed. In this transformation the world coordinate system is transfered into the camera coordinate system centered at the point P_c by performing a rotation and a translation:

$$P_{cH} = \begin{pmatrix} r_{11} & r_{21} & r_{31} & t_x \\ r_{12} & r_{22} & r_{32} & t_y \\ r_{13} & r_{23} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = BP_{wH} \tag{2.3}$$

Where P_{cH} is the transformed point with respect to the camera coordinate system, B is the external transformation matrix that contains a rotation and a translation part and P_{wH} is the original point with respect to the world reference frame. In literature the camera coordinate system is also denoted as the eye coordinate system [Li01]. The subscript H indicates that the point is in homogeneous coordinates. A short explanation to homogeneous coordinates is given in chapter 6.3.

After the external transformation the so called perspective transformation has to be made. This transformation transforms the point P_{cH} into the sensor coordinate system by using the perspective projection matrix P. The result is the point P_{iH}:

$$\mathbf{P_{iH}} = \begin{pmatrix} fx & 0 & 0 & 0 \\ 0 & fy & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = PP_{cH}$$

The last step is the internal transformation where the camera coordinate system point is transformed into discrete image coordinates. For this transformation we need the internal matrix A resulting in the point P_pH:

$$\mathbf{P_{pH}} = \begin{bmatrix} s_u & 0 & u_0 \\ 0 & s_v & u_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = AP_{iH}$$

So finally the resulting transformation from world coordinates into pixel coordinates can be written as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{pmatrix} s_u & 0 & u_0 \\ 0 & s_v & u_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} fx & 0 & 0 & 0 \\ 0 & fy & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} r_{11} & r_{21} & r_{31} & t_x \\ r_{12} & r_{22} & r_{32} & t_y \\ r_{13} & r_{23} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$s = (s_u, s_v)$ is the scale factor and describes the ratio of pixel spacing in x- and y-direction [Hor00]. In the following the intrinsic and extrinsic part of the computer vision pipeline are described in more detail.

Intrinsic and Extrinsic Parameters

In the computer vision pipeline the external and internal transformations are used for transforming the world coordinates into pixel coordinates. Those parameters can be subdivided into so called intrinsic and extrinsic parameters. The intrinsic parameters define optical characteristics as well as internal geometry of the camera [HS97]. The intrinsic parameters for a camera are:

- Principal point $C = (C_x, C_y)$: The principal point is the intersection point between the image plane and the optical axis
- Focal length $f = (f_x, f_y)$: The focal length is the distance between the image plane and the center of projection.
- Scaling factor $s = (s_u, s_v)$: The scaling factor describes the horizontal and vertical scaling.

With these parameters the so called intrinsic matrix can be formed as follows:

$$\mathbf{A} = \begin{bmatrix} fs_u & 0 & u_0 & 0 \\ 0 & fs_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Where u_0 and v_0 describe the transformation towards the principle point.

The extrinsic parameters describe the orientation and location of the camera reference frame with respect to the world frame [Sab08]. They are described in the following:

- Rotation parameters yaw θ, pitch ϕ and roll ψ. The rotation parameters describe the rotation of the coordinate systems
- Translation $t = (t_x, t_y, t_z)$: t describes the translation of the object coordinate system towards the center of projection.

The euler angles yaw, pitch and roll can be combined into one rotation matrix R:

$$\mathbf{R} = \begin{pmatrix} \cos\Theta\cos\Psi & -\cos\Theta\sin\Psi & \sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi + \cos\Phi\sin\Psi & -\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & -\sin\Phi\cos\Theta \\ -\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi + \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{pmatrix}$$

A short description of the euler angles can be found in chapter 6.3.

The extrinsic parameters (rotation and translation) can now be combined into one external transformation matrix [Sch05] :

$$\mathbf{B} = \begin{pmatrix} r_{11} & r_{21} & r_{31} & t_x \\ r_{12} & r_{22} & r_{32} & t_y \\ r_{13} & r_{23} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{bmatrix} R & t \\ 0_3^T & 1 \end{bmatrix}$$

The extrinsic and intrinsic parameters can now be used in the computer vision pipeline.

Lens Distortion

In an ideal pinhole camera model we can expect that the central projection is conducted without any distortion. When taking images with a camera that uses a lens, one has to cope with two types of distortions: radial and tangential distortion which both have to be corrected [MSB99]. Radial distortion signifies a non-linear mapping on the image plane when increasing the distance radially from the principle point [Sch05]. Tangential distortion describes the decentering of the principal point away from the optical axis [MSB99]. Lens correction can be performed by adding the error compensation factors δu and δv to the uncorrected image coordinates $\tilde{u}$ and $\tilde{v}$:

$$u = \tilde{u} + \delta u \tag{2.4}$$

$$v = \tilde{v} + \delta v \tag{2.5}$$

δu and δv contain the error correction for radial and tangential distortion and can be described by using the following equations [Sch05]:

$$\delta u = \tilde{u}(\kappa_1 r_d^2 + \kappa_2 r_d^4 + ...) + [\eta_1(r_d^2 + 2\tilde{u}^2) + 2\eta_2\tilde{u}\tilde{v}](1 + \eta_3 r_d^2 + ...) \tag{2.6}$$

$$\delta v = \tilde{v}(\kappa_1 r_d^2 + \kappa_2 r_d^4 + ...) + [2\eta_1\tilde{u}\tilde{v} + \eta_2(r_d^2 + 2\tilde{u}^2)](1 + \eta_3 r_d^2 + ...) \tag{2.7}$$

$$\text{with} \quad r_d = \sqrt{\tilde{u}^2 + \tilde{v}^2} \tag{2.8}$$

The parameters κ_i describe the radial distortion, the parameters η_i describe the tangential distortion [Sch05] and r_d describes the distance between the measured pixel and the principal point $p = (u, v)$ [MSB99]. The distortion correction parameters can be combined with the matrix of the intrinsic parameters resulting in a so called calibration matrix K_k that has to be constructed for every camera that is used during image acquisition [SBK08].

$$\mathbf{K_k} = \begin{pmatrix} fs_u & 0 & u_0 + \delta u \\ 0 & fs_v & v_0 + \delta v \\ 0 & 0 & 1 \end{pmatrix}$$

3D Vision

There are many ways to retrieve 3D information. Considering only optical systems there are two umbrella terms describing the functionality of the respective systems namely active and passive systems. The choice of the system basically depends on the available equipment. These two categories will be described below.

Active Systems

Active Systems use controlled emission of light. In the following different active optical methods are explained in more detail [CBS00] [Gmb09].

- **Time of Flight:** Time-of-Flight cameras consist of an illumination source and a sensor. A modulated infrared light signal is emitted by the light source and reflected by the measured object. Depth information is computed either by the phase difference between the emitted and the received signal or by the time between emitting and receiving a light pulse. So Time-of-Flight technologies can be subdivided into two categories: Systems that use continuous wave modulation and systems that use pulsed wave modulation [Sab08]. Continuous wave modulation techniques use sinusoidal or more commonly square waves for wave modulation. The range can be estimated by using the following equation where c is a constant for the speed of light, φ_0 is the measured phase and N is the range estimation ambiguity [Gmb09]:

  $$R = \frac{c}{2f_{mod}}\left(\frac{\varphi_0}{360°} + N360°\right) \qquad \text{with } N = 0, 1, 2, 3... \tag{2.9}$$

  Choosing varying values for the modulation frequency f_{mod} the NAR (non-ambiguity range) can be estimated which describes the maximum distance that can be captured without capturing ambiguous range data caused by overlapping waves [Gmb09]:

  $$NAR = \frac{c}{2f_{mod}} \tag{2.10}$$

  Continuous wave modulation techniques have the advantage of not depending on a high power laser which allows these cameras to use a wide field of light sources. The disadvantage of these systems is the correlation between modulation frequency and the measuring range. Because of the ambiguity a high modulation frequency decreases the measurable distance. The pulsed wave modulation technique is also known as LIDAR (LIght Detecting And Ranging) and requires a high power laser because of the short time intervals between emission and detection. The use of a high power laser limits the application field because of the danger of injuring the eyes of captured people. Also just one range at a time can be acquired which extends the time of capturing a whole scene. However, the LIDAR technology has a range from 0.1 mm to several kilometers having an accuracy of about 0.01 mm which makes this technology very usable for many technological applications like industrial robotic or military use [MSB99].

- **Triangulation:** A range finder based on the triangulation technique consists of a camera and a projector. In principle the projector emits a light signal that is captured by the camera. With help of the triangulation principle the distance to the object can be estimated using the following equation [Sab08]:

$$Z = \frac{b \sin \alpha \sin \beta}{\sin (180 - \alpha - \beta)} \tag{2.11}$$

  b describes the distance between the camera and the projector, α describes the angle between b and the light ray and β describes the angle between b and the normal vector of the camera. Figure 2.4 shows the connection between object point, camera and projector. The angle between the baseline b and the light ray of the projector $proj$ is the angle α, the angle between the baseline b and the viewing ray of the camera cam is the angle β. To be able to use the triangulation principle α, β and b have to be known.

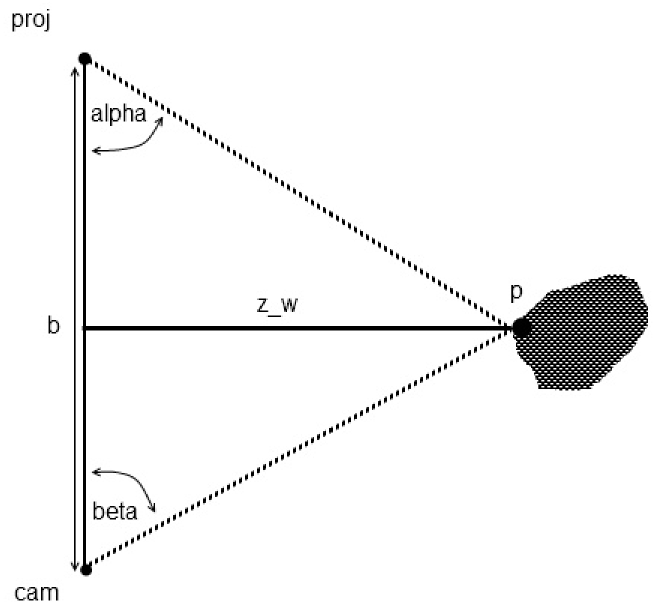


Figure 2.4: The triangulation principle

There are many possibilities of projecting light onto the object. The simplest method is to project one single point onto the object so no correspondence problems can occur. The disadvantage of this method is that the use of only one laser point can lead to occlusions more easily. Also the image retrieval process takes some time because the laser has to be steered all over the object to capture all points. Because of these limitations other light projection methods have been developed like sheets of light, pattern projection or coded light techniques. Using one sheet of light there are several possibilities to capture the object. For example the projector can be equipped with a deflection unit that deflects the light beam in different directions. A cheaper method is to move either the object or the projector to different locations. Like the single point projection, the single light projection method is slow because only one light sheet at a time is captured. That is why multiple coded light stripes or patterns are used more often. All of these methods have a similar procedure [Sab08]:

1. A laser, infrared or coded light pattern is projected into the scene.
2. The pattern that is reflected by the object surface is captured by a camera.
3. The observed structure is evaluated in every frame receiving depth information by using the triangulation principle.

An example for light projection techniques is the Moirè technique that can be categorized into Shadow Moirè or Projection Moirè [CBS00]. This method uses two patterns also called gratings - one grating is projected on the object the other grating is used to observe the reflected image. The emitter could be a projector using lines and the receiver could be a camera equipped with a line filter. There are multiple methods to increase the image acquisition speed and precision leading from binary coded light-cutting over phase shifting to color coding. These improvements have the advantages that they are easy to implement and since there is no deflection unit needed, they are more affordable than a regular Moirè range finder. Further information about different triangulation methods can be found in [CBS00].

Using triangulation one has to cope with various challenges. First of all depth information can only be retrieved if the laser reaches a point on the object surface that is captured by the camera. So in order to capture the whole object surface it has to be ensured that every object point is captured simultaneously by receiver and emitter at least once during image retrieval which can be very tricky especially if the object surface contains a lot of concave surfaces which produce occlusions. Another problem regarding the evaluation of the scanning process is varying results caused by different surface characteristics and surface edges.

Passive Methods

Basically passive systems use the information that can be found in one or several images. These methods are known as “Shape from X“ methods and acquire depth information using intensity information from the retrieved image. Not all passive methods are able to compute the exact depth value for an object point because the main idea of passive methods is to calculate depth information out of 2D images. In the following a few techniques are described in more detail. Further information can be found in [MSB99].

- **Shape from Shading:** This method makes it possible to gather information about lighting and depth conditions by just processing characteristics that can be found in a 2D image. The human brain serves as an example for this method because it is able to estimate depth by processing surface characteristics and lighting conditions. The prerequisites for Shape from Shading are:

  - surfaces that are invariant respecting rotations
  - surfaces that are illuminated by an illumination source that is far away and whose position is known
  - surfaces that have no cast shadows and do not contain any interreflections

  Also the image has to be an orthographic projection. In [ZTCS99] different Shape from Shading methods are compared:

  1. **Minimization approach:** This method was first implemented by Ikeuchi and Horn. It minimizes an energy function consisting of a brightness factor and other constraints like an integrability constraint. The result of the calculation with a given surface shape leads to the surface normal and surface gradients. Further information can be found in [Ike89]
  2. **Propagation approach:** The propagation method was also implemented by Horn who assumed that the depth and orientation of all points can be computed if the depth and orientation of the starting point of a line is known [ZTCS99]. To accomplish that, characteristic points in the image are chosen and evaluated. Characteristic points are e.g. points with maximum intensity. The shape information between characteristic points is then interpolated.

  Zhang worked out that these methods have problems with the uniqueness of the points when no additional information about the surface is known. Also both methods can only handle ideally opaque surfaces with ideal diffusion (so called Lambertain surfaces) [MSB99].

- **Shape from Texture:** Like Shape from Shading this method is also inspired by the processes that happen in the human brain because humans are able to determine depth out of texture information. Texture that is viewed from a specific angle in a 3D setting is distorted due to the perspective projection and with increasing distance the pattern is getting smaller. Using this properties it is possible to gain depth information provided that the texture is a regular pattern whose texel are all the same size. By estimating the vanishing point and vanishing line it is possible to calculate the orientation of the surface.

- **Stereopsis:** In contrast to the other passive methods that used the information of only one 2D image, stereo vision uses the information that can be gathered by observing two or more 2D images. In principle stereo vision simulates the human visual system having two eyes to observe a scene. By capturing images from two different views it is possible to retrieve depth information. The precondition of being able to get 3D information is the epipolar constraint which implies that every point that lies on the epipolar line of one image has to lie on the epipolar line of the other image too. Figure 2.5 shows the principle of epipolar geometry. The captured point $p = (x_w, y_w, z_w)$ is mapped onto Image 1 and Image 2 resulting in the points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$. The epipolar plane is generated by three points: the object point p, the center of projection of the left image c_1 and the center of projection of the right image c_2. The intersection lines between the images and the epipolar plane are called the epipolar lines. The baseline is the distance between the centers of projection.

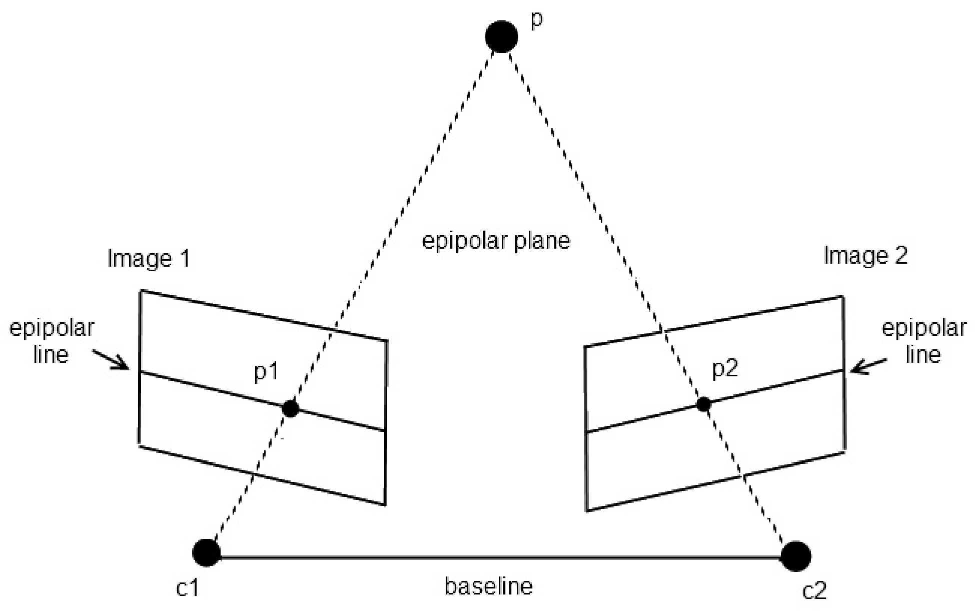


Figure 2.5: Epipolar geometry

Assuming that both cameras used are calibrated the so called epipolar equation can be described as followed:

$$\tilde{p_1}^T E \tilde{p_2} = 0 \tag{2.12}$$

$\tilde{p_1} = (x_1, y_1, f_1)$ is the corresponding point of $\tilde{p_2} = (x_2, y_2, f_2)$ where f_1 and f_2 are the focal lengths of both cameras. $E = [t]_x R$ is the essential matrix that describes the transformation between the two cameras by using a rotation matrix R and a translation vector $[t]_x$([Sch05], [MHS05]). The essential matrix can be estimated by using a set of corresponding points.

- **Shape from Motion:** As “Shape from Shading“ and “Shape from Texture“, “Shape from Motion“ originates from processes in the human brain. From the cradle the human being learns that objects that move faster during motion are nearer than objects that move slower (such as mountains). Shape from motion designates a method that uses the epipolar constraint for estimating the distances to the captured object. Instead of using two or multiple cameras, only one camera that is moved around the object, is used. The shift between a point in one image and the same point in the subsequent image is called disparity. To obtain the distance z_w of the point $p = (x_w, y_w, z_w)$ the following equation using the disparity can be formulated as: [Sab08]:

$$z_w = \frac{bf}{d} \tag{2.13}$$

The baseline which is the distance between the centers of projection, is multiplied with the focal length and is divided by the disparity $d = x_1 - x_2$. The formula can be established by the coherence between the world and the camera coordinate system:

$$\frac{x_w}{x_1} = \frac{z_w}{f} \tag{2.14}$$

Subtracting the distance between the centers of projection from the point x_w a second equation describing the coherence between the second image and the distance z_w can be established:

$$\frac{x_w - b}{x_2} = \frac{z_w}{f} \tag{2.15}$$

Now we can build the following equation and reduce it:

$$\frac{z_w x_1}{f} = \frac{b - z_w x_2}{f} \tag{2.16}$$

$$z_w = \frac{bf}{d} \quad \text{with} \quad d = x_1 - x_2 \tag{2.17}$$

2.2 Object Capturing

This chapter will give a short explanation about object/motion capturing in general containing the particular steps of the capturing pipeline focusing on the initialization and the different data capturing methods and comparing different acquisition techniques to each other.

An object or motion capturing system basically consists of four successive steps [MG01]:

- **Initialization:** This step deals with the setup of the system and contains the calibration of the cameras and the manipulation of the data such as segmentation.
- **Tracking:** The second step of the pipeline ensures that the captured object is recognized and its motion is traced in every frame. Manipulation of the data can be conducted in this step as well as it has been in the first pipeline step.

- **Pose Estimation:** The third step takes care of the identification of the orientation of the different parts of the captured body having the goal of estimating the total pose of the body in relation to the camera. To accomplish that there are different methods like model-free, indirect model use and direct model use. For further information about Pose Estimation see [MG01].
- **Recognition:** The last step of the pipeline tries to recognize the pose by using the information gained in step three. This can be done for example by using predefined templates of different poses that can be compared with the estimated pose.

In the following only steps one and two will be treated in more detail as a spadework for pose estimation and recognition in future work.

Initialization

As described in the pipeline, initialization deals with preparatory operations concerning the hardware setup and other modifications influencing the data acquisition. First of all an appropriate camera setup for tracking has to be chosen. In general it has to be decided whether one or multiple cameras are used in the system.

An important point for initialization is camera calibration which has been explained mathematically in chapter 2.1.

In the following related work is presented that deals with various techniques that can be applied to steps one and two in the tracking pipeline. Traditional approaches using 2D technologies, novel methods using depth cameras and hybrid methods are annotated and evaluated.

2D Recording

In [SC06] a framework for motion tracking using eight calibrated gray scale cameras is presented. Silhouettes and motion information are used to estimate the pose of the captured user. Parametric shape models are compared with the captured body that is an 3D estimation calculated from 2D motion captured by the cameras using pixel displacement. Another example of 2D motion capturing published by [LB08] uses a similar method to track a body. In this work a set of four high resolution industrial CCD cameras are used that are placed in a large room covered in blue fabric. For markerless motion tracking skinned mesh models are used which are created using a body scanner. In addition a skeleton described by a kinematic tree is used representing the main body parts of the mesh. The kinematic tree is used to hierarchically describe the setup of the skeleton containing a state representation for each bone. After scanning the body the movement of the body is captured by the four cameras producing images that contain motion cues. These motion cues are used to define an objective function. In order to be able to use this function, pixel correspondences between the model and the current image has to be found. To be able to get this information, optical flow and silhouette techniques are used. With this information the objective function is minimized using the Levenberg-Marquardt algorithm. The tracking algorithm provides satisfying results for normal movement but does not cope with very fast movements. Another work done by [CMC$^+$06] uses adapted fast simulated annealing in order to match an a priori model to the visual hull. The purpose of this work is to supply reliable information of the movement of a body in order to study musculoskeletal biomechanics. First of all the visual hull is constructed using an eight CCD camera setup with each camera having a resolution of 640 x 480 pixel. The a priori model is then created using a laser scan of a human being and segmenting it manually afterwards. In addition to capturing a body in a real environment the authors used a 3D model created in a virtual environment in order to evaluate the proposed method excluding errors arising from camera calibration or background subtraction. In summary the results obtained from the real environment are comparable to the virtual environment showing an effective tracking algorithm that does not require an accurate initialization of the model [CMC$^+$06]. Nevertheless due to the symmetry of the thigh and shank the method produces noisy results while tracking the rotation of these body parts.

3D Recording

In order to capture motion without using any markers Pekelny and Gotsman [PYGC08] use a single depth camera delivering a depth video sequence. The depth camera is used to capture a 3D point cloud that is transformed into a dataset containing all rigid bone transformations. The bones are found by using the ICP (iterative closest point) algorithm. Another work done by Swadzba et al. [SBSS08] uses a 3D camera mounted on a movable robot for capturing the scene. The goal of this work is to capture a static scene while moving around the robot that has to cope with moving objects and human beings. In order to capture a point cloud the robot moves around in the room and captures the data received by the 3D camera. After data acquisition the point cloud is edited using a median filter to eliminate noise generated by different surface reflections during acquisition. Swadzba et al. also used velocity information for each valid 3D point by using optical flow methods. This information is needed to identify moving people and objects. By removing the identified moving objects and people a 3D representation of a static scene is generated.

Combined Approaches

There is also the possibility of combining the 2D and 3D technologies in order to compensate the disadvantages of the 2D approach and to increase the results using additional 3D information. In [SBKK07] a camera setup consisting of a CCD camera and a TOF camera is used to estimate the pose of the camera. The scene is captured simultaneously by the CCD and the TOF camera with the CCD camera covering the whole field of view of the depth camera. The introduced method is a combination of SfM (Structure from Motion) and the time-of-flight method and is able to overcome the restriction of the traditional SfM method that needs a lateral movement to be able to get good initialization results. Supported by metric information gathered by the time-of-flight camera the pose estimation using a single CCD camera can be improved noticeably. To improve the process of segmentation Crabb et al. [CTPD08] also use a high resolution CCD camera with a TOF camera. The main part of the foreground segmentation is done by constructing a trimap of depths. Here depth values are either categorized as into the foreground, into the background or indeterminate based on a likelihood for each pixel [CTPD08]. The threshold for the foreground depth area of the pixels is assigned by the user.

Comparison of the different approaches

Markerless tracking is a challenging area of research because of the difficulty to recognize body parts and the movement of the person over the time. Nevertheless this area is becoming more and more important because of the drawbacks tracking with markers implicates. In the following the presented methods are summarized and evaluated starting with single camera tracking and leading to multiple camera approaches. When using a single camera despite of whether it is a 2D or 3D camera it is only possible to capture a single view at one time. This fact makes it impossible to be able to gather the information that lies beyond that view. In order to be able to track an object or a person in the 3D space the single camera has to be moved (see chapter 2.1) or more than one camera is needed. Examles of such methods where shown previously. All the presented methods used silhouette or visual hull techniques in order to extract to moving object from the images. The disadvantages of these methods are that they are prone to fast movement and that they have problems with distinguishing body parts which look similar. Using 3D cameras has the advantage that the depth information enhances the information a 2D camera delivers. With the additional depth information it is possible to retrieve results in a faster way with no need to process only the 2D information. Also the depth information can be used to distinguish between the moving object or person and the background which accelerates the process of background segmentation. As described before using only a single 3D camera has the same drawback as using a single 2D camera. The information that lies beyond can not be captured. When using a 3D camera mounted on a robot like in [SBSS08] only a static scene can be captured. As described in [SBSS08] a combination of a 2D and a 3D camera makes it possible to extract foreground information faster than using only 2D information but also only a single view can be captured. In order to be able to capture a moving object or person in 3D space multiple cameras are needed. Here it is possible to either use multiple 2D cameras, a combination of 2D and 3D cameras or a combination of multiple 3D cameras as presented later in this work.

Using multiple 2D cameras makes it possible to capture a three dimensional image of the observed scene. As described in [CMC$^+$06] silhouette and visual hull techniques are used to identify the moving subject and extract the foreground information. This has the same drawback as the single 2D camera method. The precise edges of the object can not be captured when using the visual hull method. When using the silhouette method the area between the different views can be very noisy due to the projection of the single views. Using multiple 3D cameras as presented in this work, makes it possible to use the information of the 2D image and the depth information for every single view. This makes it possible to create a 3D point cloud and to segment fore- and background with only the use of the depth information. Using multiple 3D cameras makes it also possible to deliver a more precise result for the edges of the captured object

In the following chapter the design of such a rig containing a set of two 3D cameras is described.

CHAPTER 3

Design

This chapter covers the design of the system in general, starting with the workflow, the used hardware and the camera setup. Additionally a short overview of the used libraries containing OpenNI and the multiple camera calibration software MIP-MCC is given. The last part of the chapter contains a detailed description of the particular steps starting from data acquisition and leading to the result: the merged point cloud.

3.1 Workflow

As described in chapter 2.2 this work deals with the first step of the object capturing pipeline called initialization. As mentioned before, initialization is an umbrella term and so it contains several steps each of them being dependent of the previous step. Figure 3.1 gives a coarse overview of the workflow. The first step of this pipeline is the Camera Setup which consists of a Microsoft Kinect and a PMD[vision] Camcube 3.0. A well-considered hardware setup makes it possible to get the best possible results, and so it is important to consider influences like incident solar radiation, the angle between the cameras, the working environment and the camera settings. Chapter 3.2 will deal with this step of the pipeline in more detail.

The next step in the pipeline treats camera calibration. As mentioned in chapter 2.1 the process of calibration delivers the intrinsic and extrinsic parameters of a camera setup making it possible to transform the world coordinates into pixel coordinates. In this work the calibration software MIP-MCC has been used. It uses a planar checkerboard pattern to find corresponding points in both pictures (for more information see subsection stereopsis in chapter 2.1). In order to get the necessary parameters of the calibration process the software needs calibration shots done with the calibration pattern which have to be taken with both cameras. The result of the calibration step is a calibration matrix for both cameras. The quality of the calibration process depends on the amount of shots that have been taken, on the lighting conditions and of the position of the calibration pattern in reference to the cameras. The calibration process is treated in chapter 3.4.

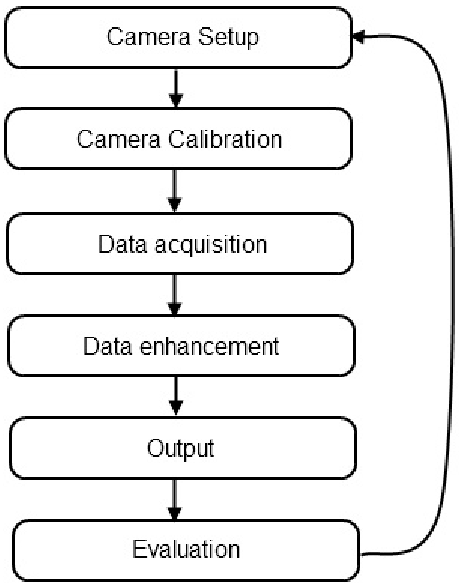


Figure 3.1: Workflow of the practical part of this work

After the camera setup has been defined, the data acquisition step can be initiated. Both cameras are connected to the computer using an USB connection. The interface between the cameras and the computer is a specific driver for each of the cameras. The communication with the cameras may be established by using the OpenNI library for the Kinect and the PMD plugin for the PMD camera. The result of this step is an array of depth values for the PMD and a depth map for the Kinect camera. For more information about this step see chapter 3.5.

The next step of the pipeline treats the enhancement and the segmentation of the gathered data. For the PMD camera it is possible to change the integration time and the modulation frequency leading to different qualities of the output data. The OpenNI framework makes it possible to change the resolution and the frame rate. Another important aspect of data enhancement that facilitates object tracking is background segmentation. In this work the Background Model Method is used. Data enhancement also includes noise reduction. Especially the PMD camera which uses the time-of-flight technology suffers from pixel noise that has to be attenuated in order to get good results. The chapters 3.6 and 3.7 treat these aspects in more detail.

The last step of the pipeline is the combination of the corrected and calibrated data which results in a single point cloud. The advantage of using two cameras is that the accuracy and the field of view is increased. Using more than two cameras makes it possible to capture the whole scene and so creating a complete 3D reconstruction of the viewed object. Chapter 3.8 discusses the last step of the pipeline.

In figure 3.2 the workflow is shown in more detail in order to make it clear how the specific steps of the pipelines of both cameras interact. It has to be noticed that the evaluation at the end of the pipeline can cause a new iteration of the pipeline in order to improve the results. Also, changing the camera setup during data acquisition makes it necessary to repeat the calibration process.

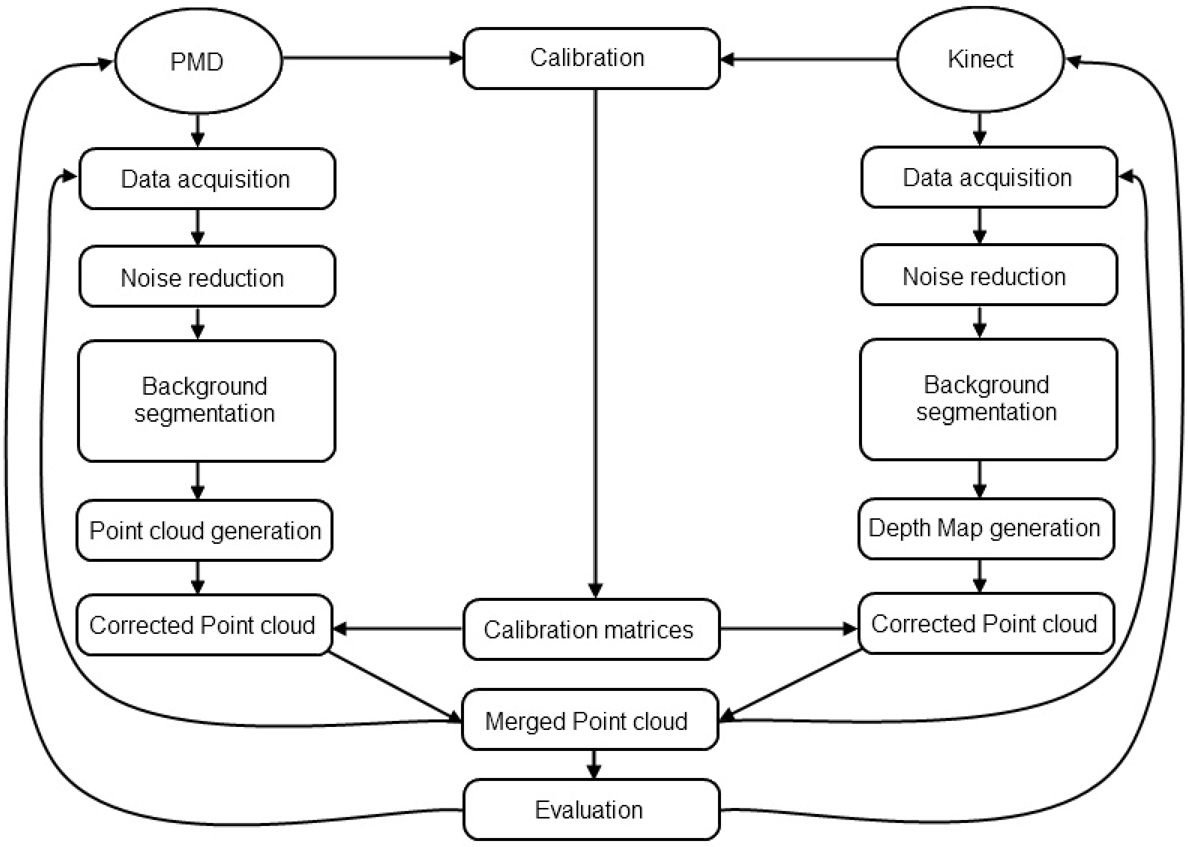


Figure 3.2: Workflow in detail

3.2 Hardware and Setup

In this chapter the technology of the used hardware is described focusing on the computer vision methods used to generate depth images. In addition, an overview of the working environment is given by specifying the camera setup and the spatial layout.

PMD[vision] Camcube 3.0

The first depth camera that has been used for this work is the PMD[vision] Camcube 3.0. This camera works with the time-of-flight principle using one PhotonICs ©PMD 41k-S2 sensor (see figure 3.3) and two illumination units (see figure 3.4), one on the left side and the other one on the right side of the sensor (see figure 3.5). The sensor is able to capture gray scale and

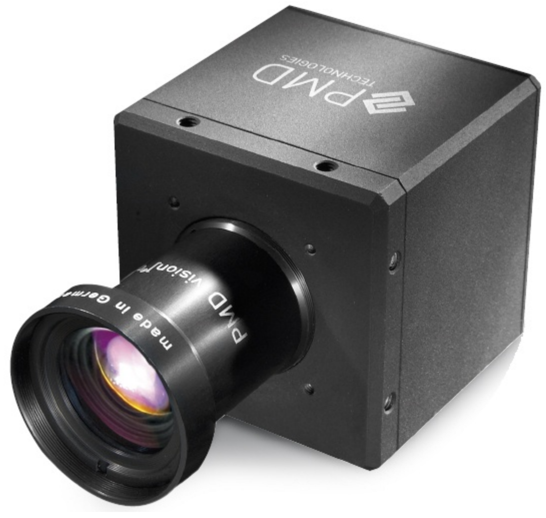


Figure 3.3: The PMD[vision] Camcube 3.0 sensor

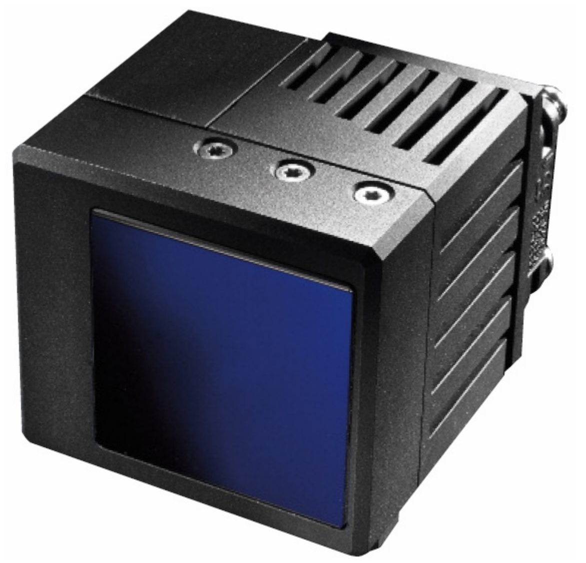


Figure 3.4: One of the two PMD[vision] Camcube 3.0 illumination units

depth images with a 200×200 pixel resolution simultaneously and features SBI (Suppression of Background Illumination) which reduces the sensibility with respect to illumination [Gmb11]. The measurement range of the camera lies between 0.3 and 7 meters with a field of view of $40^{\circ} \times 40^{\circ}$. The quality of the range data depends on factors like integration time (which can be determined programmatically) and lighting conditions. Increasing the integration time increases

Figure 3.5: The PMD[vision] Camcube 3.0 depth camera

the signal strength and so improves the quality of the data. In spite of the features that are used to suppress the disruption caused by incoming light, the sensor is prone to excessive amount of incoming sun light which leads to pixel errors (see chapter 3.6). The PMD camera uses continuous wave modulation with square waves for retrieving the range data. The output is a combination of the optical echo and the modulation voltage over integration time [Gmb09]. The ambiguity of the signal is avoided by phase shifting the periodical signals. To calculate the range value R the following formula is used [Gmb09]:

$$R = \frac{NAR\phi_0}{360°} \tag{3.1}$$

NAR is the non-ambiguity range and the parameter ϕ_0 is the phase difference which can be calculated by using the correlation function (the correlation of two phase shifted signals) ϕ_{corr} in the following way:

$$\phi_0 = \arctan(\frac{\phi_{corr}(270°) - \phi_{corr}(90°)}{\phi_{corr}(0°) - \phi_{corr}(180°)}) \tag{3.2}$$

The technical specifications of the PMD[vision] Camcube 3.0 are shown in table 3.1. For more information about the PMD[vision] Camcube 3.0 see [Gmb09] and [Gmb11].

Microsoft Kinect

The second depth camera that has been used for this work is the Microsoft Kinect. In contrast to the PMD depth camera which works with the time-of-flight principle, the Microsoft Kinect camera generates a pattern using infrared structured light and calculates the disparity between

Specification Parameter	**Value**
Sensor	PhotonICs ©PMD 41k-S2
Measurement range	0.3 - 7 meters
FOV (Field Of View)	$40° \times 40°$
FPS (frames per second)	40 fps for $200x200$ pixel
Connection to PC	USB 2.0
Focal length	12.8 millimeters

Table 3.1: Specification of the PMD[vision] Camcube 3.0

successive pixels [GRBB11]. The Microsoft Kinect consists of two CMOS sensors (one for capturing infrared light the other one for capturing RGB images) and one laser infrared projector (see figure 3.6). The Microsoft Kinect is able to capture an infrared image with a resolution

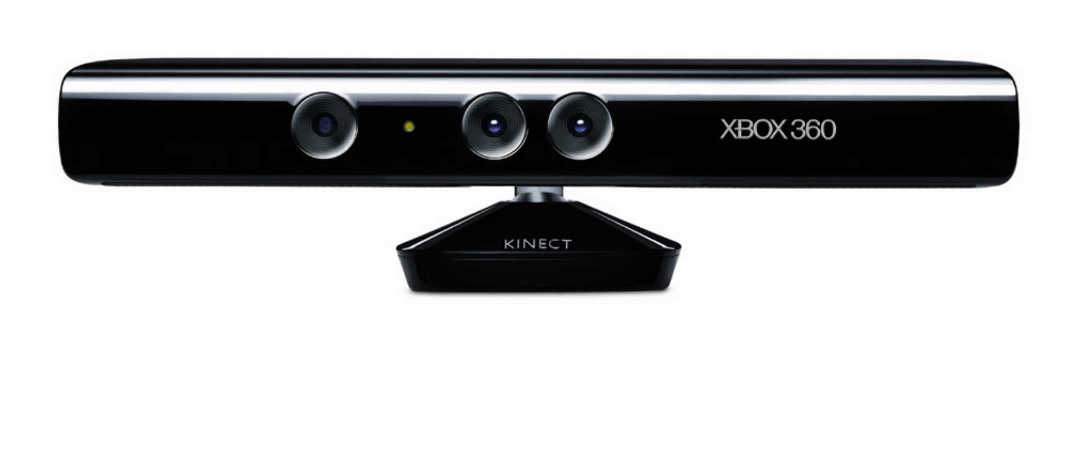


Figure 3.6: The Microsoft Kinect

of 320x240 pixel and a RGB image with a resolution of 640x480 pixel simultaneously. The measurement range of the camera lies between 1.2 and 3.5 meters with a horizontal field of view of 57° and a vertical field of view of 43° [Res11]. As mentioned before the projector of the Kinect works with a constant speckle IR pattern that is projected onto the scene. The pattern consists of 633×495 spots having brighter and darker subdivision groups of 211×165 spots so it results in a 3×3 IR point matrix. The center point of each 211×165 point sub-matrix is brighter than the other points. Figure 3.7 shows a photo of the pattern that has been taken with a standard camera. The depth value of a pixel can be estimated by comparing a memorized pattern of a pixel with the local pattern of a pixel. For this a correlation window is used comparing the pixel and its 64 neighboring pixels. The calculated offset between the compared pixels is called disparity. The actual depth value can then be calculated with triangulation. [Kho11]

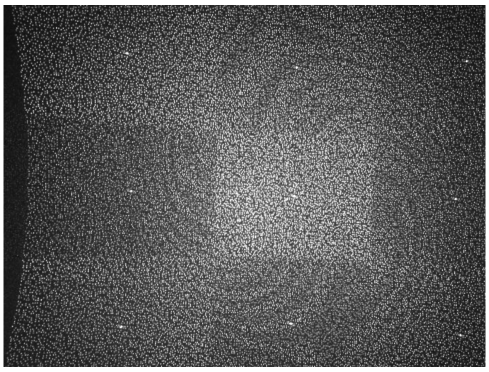


Figure 3.7: Pattern generated by the Microsoft Kinect IR projector

3.3 Libraries

For this work several libraries were used, either to establish the connection with the cameras or to visualize the result. In the following a short overview of the used plugins, frameworks and libraries is given containing an explanation of the application field in this work.

Open Natural Interaction (OpenNI) Framework

The OpenNI framework implemented by the non-profit organization OpenNI makes it possible to realize interaction between a human being and a device. The interaction between a device and a user is accomplished by using so called „Natural Interaction“ [Ope11b]. This term reflects the fact that no peripheral equipment like a mouse or a keyboard is used for interaction. The only way to communicate with the system is using hand gestures, voice commands or body motion. This makes it possible to interact with the system the same way human beings communicate with each other. In order to accomplish natural interaction with an application, OpenNI uses APIs (application programming interfaces) that enable access to audio and vision middleware like cameras. Figure 3.8 shows the interaction between the hardware, OpenNI, the middleware and the application and demonstrates that OpenNI communicates with every other layer of the process. The communication between the application and OpenNI is accomplished by using so called production nodes which provide the data retrieved by the used devices [Ope11b]. There are different types of production nodes that can be used for different application areas. For this work depth generator production nodes have been used that provide raw data from a depth sensor (in this case from the Kinect depth camera). The OpenNI framework collects the data and transfers it into a depth map (see chapter 3.5). In order to support offline analysis of the data received by the Kinect, the OpenNI framework supports recording and playback of data. The data is stored in a proprietary .oni file format that can be handled with OpenNI player production nodes. Apart from retrieving data the OpenNI framework can be used to provide other features like pose estimation, skeleton generation or detection of user position. One of the features that supports the implementation of applications using OpenNI is the production node error status. With this production node it is possible to support error detection during the process

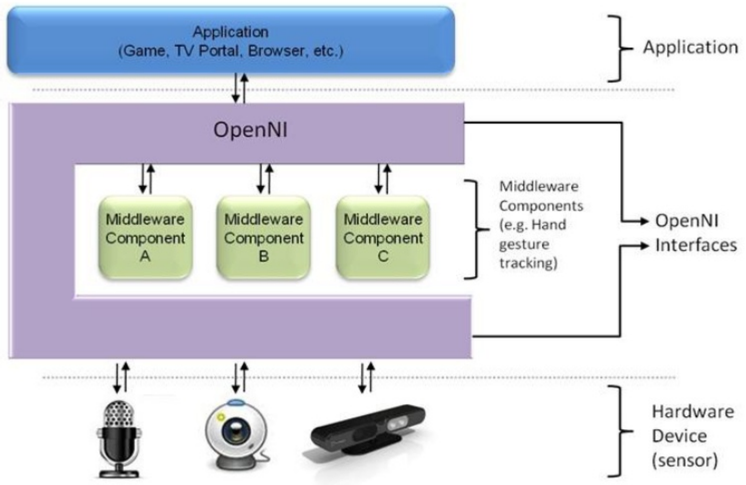


Figure 3.8: The functionality of OpenNI

of development and during the usage of the application. The usage of OpenNI in this work is explained in detail in chapter 4.

PMDSDK 2

The SDK (software development kit) implemented by the company PMDTec makes it possible to communicate with the PMD[vision] Camcube 3.0 and enables depth data acquisition [Pro09]. Apart from data acquisition the API permits the manipulation of certain camera parameters like integration time or modulation frequency. The SDK is included in the scope of delivery when purchasing the PMD[vision] Camcube 3.0. In order to provide communication with the camera and to use different functions like retrieving 3D data, the SDK uses two plugins. The first plugin takes care of the communication with the camera and the second one treats the processes that are needed to generate depth data [Pro09]. Like OpenNI the PMDSDK 2 supports error states which facilitates the implementation process. The usage of the PMDSDK 2 is described in more detail in chapter 4.

3.4 Calibration

The process of calibration is an elaborate operation especially when more than one camera is involved in the data acquisition process and each camera is produced by a different company. In order to calibrate each camera separately and in relation to the other cameras used, several computer vision methods have to be used in order to find corresponding points in all images and so gaining the intrinsic and extrinsic parameters of every camera. The result of the calibration process is the calibration matrix for every used camera. These matrices make it possible to transport the depth data into one combined coordinate system. For this work the calibration process is handled by a software named MIP-MCC (MIP-MultiCameraCalibration) developed by the Multimedia Information Processing Group of the technical faculty of the University of Kiel in Germany [Sch11]. For calibration, the MIP-MCC software uses OpenCV, a library that contains the implementation of computer vision algorithms like calibration or face detection [Ope11a]. The software makes it possible to calibrate either a single camera or multiple cameras and it calculates the intrinsic and extrinsic parameters. When calibrating a depth camera it is also possible to deal with measurement errors regarding the depth values. The MIP-MCC software calibrates the parameters of the cameras by using pictures made of a planar checkerboard pattern. In all images correspondent 2D-3D value pairs have to be determined in order to be able to use epipolar geometry. With these correspondent pairs it is possible to calculate the intrinsic and extrinsic camera parameters. The calibration process using the MIP-MCC software consists of the following steps:

- Taking a set of different types of pictures of a checkerboard pattern simultaneously with every camera in the rig.
- Building image lists for every format of every camera in the rig.
- Defining the corners of the checkerboard in every picture of every format.
- Approximation and refinement of the intrinsic parameters for every picture.

Image Generation

The first step of the calibration process deals with the image generation. In order to be able to calculate the intrinsic and extrinsic parameters of every used camera several pictures of the calibration pattern have to be taken. In order to produce accurate calculations the amount of pictures should lie between 20 and 80 pictures for every camera and every format. The different picture formats are essential to support 2D cameras as well as 3D cameras. When using only a set of 2D cameras in the rig only the generation of intensity images is possible so for the calibration process only images of this format are available. When using depth cameras the generation of intensity, amplitude and depth value images is possible and so the software has the possibility to calculate the calibration parameters and to estimate the depth errors. When taking the pictures of the checkerboard pattern it has to be assured that the whole checkerboard pattern is visible in every image. Also pictures from multiple distances and angles in respect to the cameras should be taken in order to be able to correct tangential and radial distortions and depth errors. A sample of a series of pictures taken with the PMD and the Kinect depth camera is shown in figure 3.9 and 3.10. For this work a series of 500 pictures were taken each in a different format.

Image Lists

The MIP-MCC software uses so called image lists for the calibration. Each image list contains a number of picture paths and must contain the same number of images like every other image list used for calibration. Also the order of the images has to be the same in every image list. To generate such lists the pictures have to be in the right format. The MIP-MCC software uses pictures that have the file extension '.mip'. Such files can be generated with a tool named biasShowCamWx which is a part of the BIAS library for computer vision developed by the Multimedia Information Processing Group at Kiel University [ea11]. The biasShowCamWx tool simultaneously grabs infrared, amplitude and depth images from all cameras used in the rig. Only the RGB pictures of the Kinect camera have to be taken separately. This limitation makes it necessary to first grab a set of infrared, amplitude and depth images at one time, then switch to RGB Kinect Mode and last grab the same image as before in RGB mode. For grabbing the RBG image it is important that the checkerboard pattern is not moved after taking infrared, amplitude and depth images to avoid calibration errors. For this work the Kinect RGB images were not needed and so it was possible to grab a continuous image stream which makes the handling much easier. The images are stored as '.mip' files grouped by the camera type and the recording mode (amplitude, infrared, depth) and can be stored as image lists.

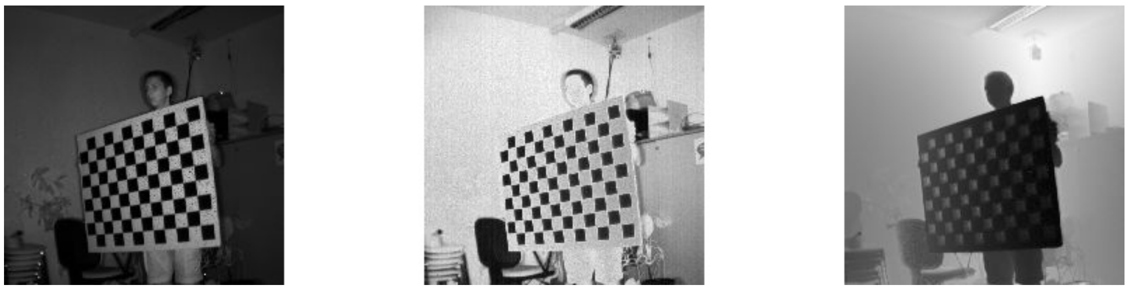


Figure 3.9: A picture series taken with the PMD depth camera. From left to right: amplitude image, intensity image, depth image

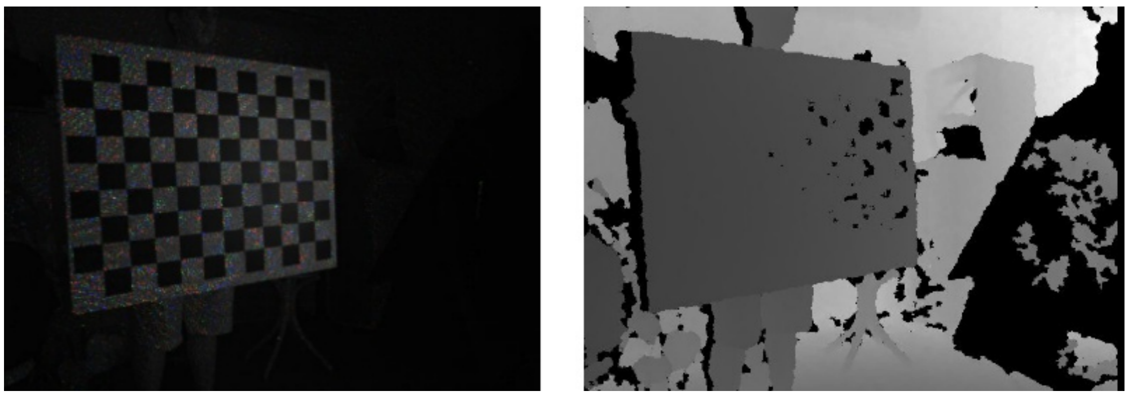


Figure 3.10: A picture series taken with the Kinect depth camera. From left to right: amplitude image and depth image

Corner Assignment

After generating the image lists the grabbed images have to be processed. For a successful calibration it is vital that the checkerboard pattern is visible in every image. Images that do not fulfill this condition have to be invalidated as well as all images that are associated with these images. As well as image invalidation the selection of the checkerboard corners is a necessary step. In every image a rectangle containing the inner corners of the checkerboard has to be selected. Figures 3.11 and 3.12 show a sample of a corner selection in an amplitude image of the PMD camera and in an infrared image of the Kinect. The corner selection has to be as precise as possible and it also is very important to always define the corners in the same order to avoid calibration errors.

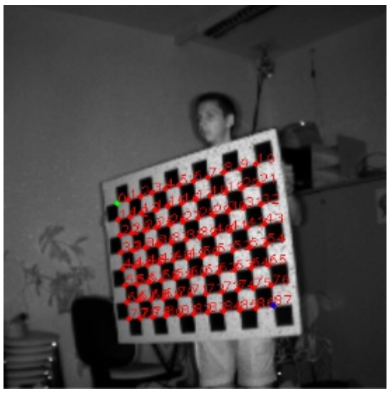


Figure 3.11: A sample of the corner selection with an amplitude image made with the PMD depth camera.

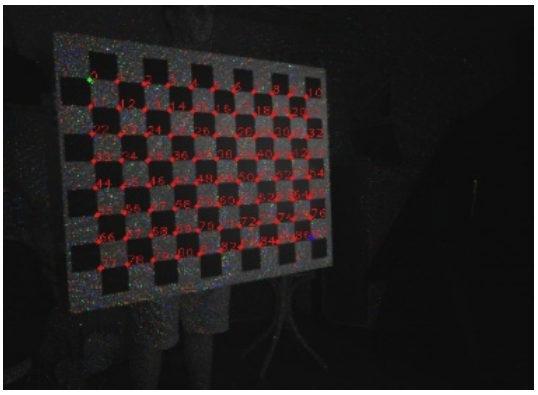
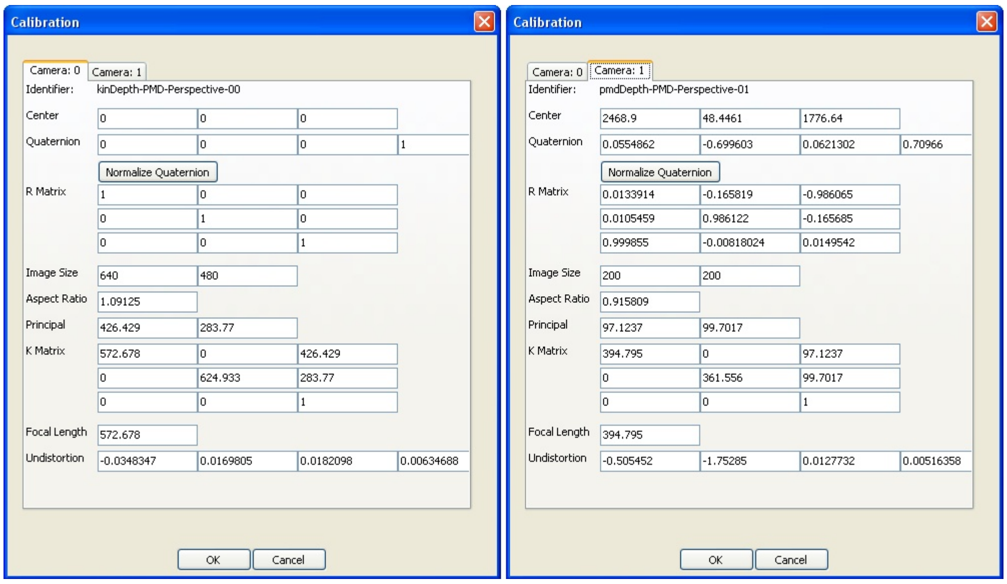


Figure 3.12: A sample of the corner selection with an infrared image made with the Kinect depth camera.

Approximation and refinement of the camera parameters

After the definition of the corners of the checkerboard, the MIP-MCC software generates an approximation for every camera position in all images and so approximates the intrinsic camera parameters for the PMD and the Kinect depth camera [Sch11]. Figure 3.13 shows an example of an approximation result of both cameras. The first row 'Center' determines the x, y and z



Identifier: kinDepth-PMD-Perspective-00

Center	0	0	0	
Quaternion	0	0	0	1
R Matrix	1	0	0	
	0	1	0	
	0	0	1	
Image Size	640	480		
Aspect Ratio	1.09125			
Principal	426.429	283.77		
K Matrix	572.678	0	426.429	
	0	624.933	283.77	
	0	0	1	
Focal Length	572.678			
Undistortion	-0.0348347	0.0169805	0.0182098	0.00634688

Identifier: pmdDepth-PMD-Perspective-01

Center	2468.9	48.4461	1776.64	
Quaternion	0.0554862	-0.699603	0.0621302	0.70966
R Matrix	0.0133914	-0.165819	-0.986065	
	0.0105459	0.986122	-0.165685	
	0.999855	-0.00818024	0.0149542	
Image Size	200	200		
Aspect Ratio	0.915809			
Principal	97.1237	99.7017		
K Matrix	394.795	0	97.1237	
	0	361.556	99.7017	
	0	0	1	
Focal Length	394.795			
Undistortion	-0.505452	-1.75285	0.0127732	0.00516358

Figure 3.13: The result of the approximation of the camera parameters

value of the translation of the coordinate system whereas the quaternion and the rotation matrix both define the rotation of the coordinate system referring to the extrinsic parameters of the calibration. The image size describes the height and width of the pictures grabbed by the camera and the aspect ratio specifies the ratio between image width and height. The row 'Principal' describes the x and y value of the principal point of the image. The K-matrix is the composition of focal length, aspect ratio and principal point and can be defined as follows:

$$\mathbf{K} = \begin{pmatrix} f & 0 & p_x \\ 0 & f * a & p_y \\ 0 & 0 & 1 \end{pmatrix}$$

The undistortion vector defines the parameters for correcting the radial and tangential distortion. After the estimation of the intrinsic and extrinsic parameters the actual calibration can be started. For every picture used in the calibration process a rig and calibration parameters are estimated by using the checkerboard pattern and epipolar geometry. At the end of the calibration process the final calibration parameters are calculated and stored in a .xml file.

3.5 Data Acquisition

After the calculation of the extrinsic and intrinsic camera parameters every action that cause a change of the parameters should be avoided. For example the movement of a camera would cause a change in the extrinsic camera parameters or the change of the focus would cause a change of the intrinsic camera parameters. Every change in either the intrinsic or extrinsic parameters makes it necessary to recalculate the camera parameters.
For data acquisition it is necessary that the cameras are placed in a certain position to each other that makes it possible to capture the user without causing too much shadowing effects. It is vital to find a good rig setting before calibration because changes in the rig setting causes a change in the extrinsic parameters. An appropriate rig setting covers overlapping areas of the images in order to be able to find corresponding image areas in the merge process later on. Inappropriate rig settings mostly cover either an area that overlaps too much or not at all. An image merge using an inappropriate rig setting having too few overlapping areas would produce an unusable calibration result because no or not enough corresponding points could be found. Using too much overlapping areas however would cover an area that is too small and so the merged views would not be usable for later applications like 3D tracking or 3D reconstruction. Also the distance from the captured object or user to the cameras is an important factor. Objects that are too near or too far away from the sensors can not be captured accurately and increase image errors. Using the Kinect the ideal distance from the captured user to the camera lies between 1.2 and 3.5 meters. Objects that are captured beyond this range produce image acquisition failures that result in shadows in the resulting images. Using the PMD camera wrong distances result in flying pixels and so cause image errors. Besides the position of the cameras relative to each other the condition of the room or area used for image acquisition has to fulfill certain criterias. In principal an empty room with no incoming sunlight would be the optimum for image acquisition. Sunlight causes, dependent on the intensity and the incident, angular, speckle noise in the resulting image. Also reflecting objects like mirrors or objects made of metal cause image distortions and noise which increases the effort of data enhancement after acquisition.
After having found the right area and position for the rig the configuration of the cameras should be evaluated. Settings like integration time or modulation frequency using the PMD depth camera can be changed and produce different results. The increase of the integration time produces images with less noise and so less pixel errors. The change of frequency makes it possible to use more than one camera of the same construction type at the same time by avoiding the ambiguity of signals. In contradistinction to the PMD depth camera the Kinect has no possibility to change any settings. The images taken with both cameras can either be stored as files or handled as real time data. The PMD depth camera has the possibility of changing the image resolution which causes a reduction of the frame rate. The Kinect only offers an image resolution of 640x480 at a frame rate of 30 fps. The output of both cameras is determined by the calculations of the used middleware. The PMD plugins deliver an array of depth values, the OpenNI framework delivers a depth map of the grabbed values. In the following the particular data types are described in more detail.

Data Acquisition with the PMD[vision] Camcube 3.0

Using the PMD depth camera the first step in the data acquisition process is the establishment of the connection. In order to do so the PMD camera needs the provided plugins as well as the provided driver to be able to transfer the data. As an interface to the computer the camera needs USB 2.0 and as an additional power supply it needs 12V power. After connecting the camera to the computer the data transfer may be started. For every pixel on the sensor the camera gets an intensity and a depth value by using the calculations described in section 3.2. The quality of the data depends on the integration time which determines the signal strength. The range of the camera is determined by the modulation frequency which is by default set to 20 MHz. The output of the PMD depth camera is a greyscale and a depth image.

Data Acquisition with the Microsoft Kinect

Like the PMD depth camera the Microsoft Kinect has an USB 2.0 interface and an additional power supply. For this work two different Kinect drivers were used: The NUI Kinect driver and the PrimeSense driver. The NUI driver was used to be able to capture images for the calibration process. The PrimeSense driver was used to capture the .oni file used for creating the resulting 3D images. As described before the Microsoft Kinect uses a speckle IR pattern and creates an infrared image and a RGB image. The actual depth information is retrieved by using triangulation as described in chapter 3.2.

3.6 Noise Reduction

As every camera, depth cameras suffer from noise which is caused by lighting conditions or the condition of the captured area. There are two possibilities for enhancing the quality of the captured data which can be used individually or in combination with each other:

1. Changing the setting of the cameras in order to improve the quality of the captured frames.
2. Using image processing methods in order to eliminate the noise.

Using the PMD[vision] Camcube 3.0 it is possible to change the integration time while capturing the image. The integration time defines the strength of the signal and so increases or decreases the quality of the data. In figure 3.14 a captured frame using a low integration time can be seen. Especially at the edges of objects the concentration of flying pixels is very high. This effect is mainly caused by the difference of the depth value between an edge and the environment. The light emitted by these surfaces vary and so two different illumination values influence the depth value of a pixel at an edge. The depth values near an edge lie in the middle of the two captured objects and seem to be „flying around“ in the captured space [SK10]. This effect can be reduced by increasing the integration time when using the PMD[vision] Camcube 3.0. Figure 3.15 shows the result of the captured frame when increasing the integration time. In order to eliminate the remaining flying pixels a Median Filter is used. A Median Filter compares the actual pixel value with his neighboring values. The values are then sorted according to size. The value of the actual pixel is then replaced by the middle value of the sorted value collection.

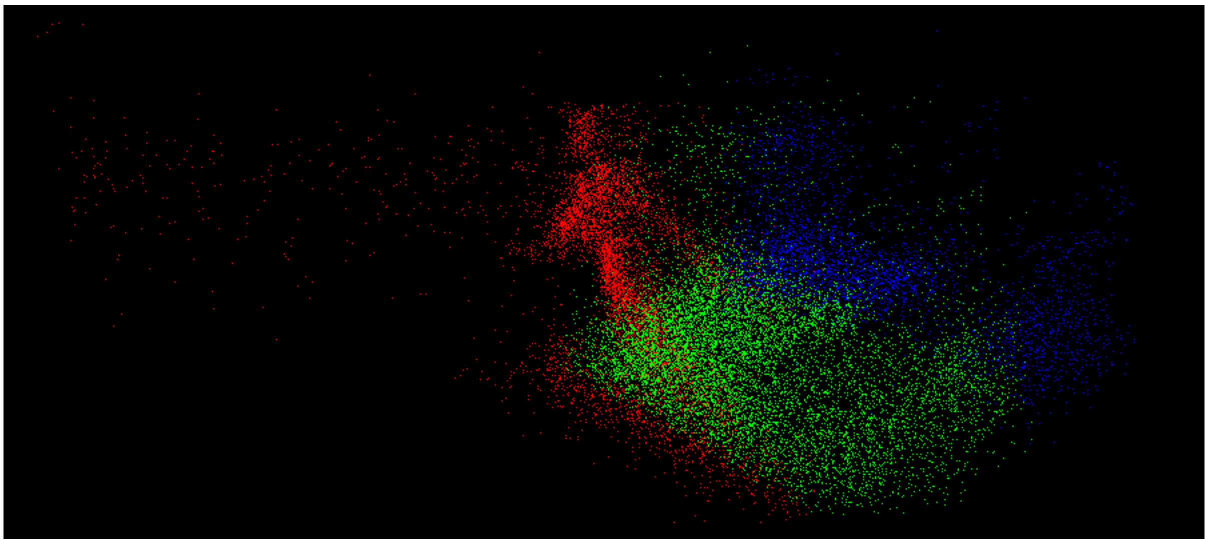


Figure 3.14: A frame captured with a low integration time. The visualization was done with OpenGL

The result of a Median Filter is a smoothed image with a reduction of outliers. The Microsoft

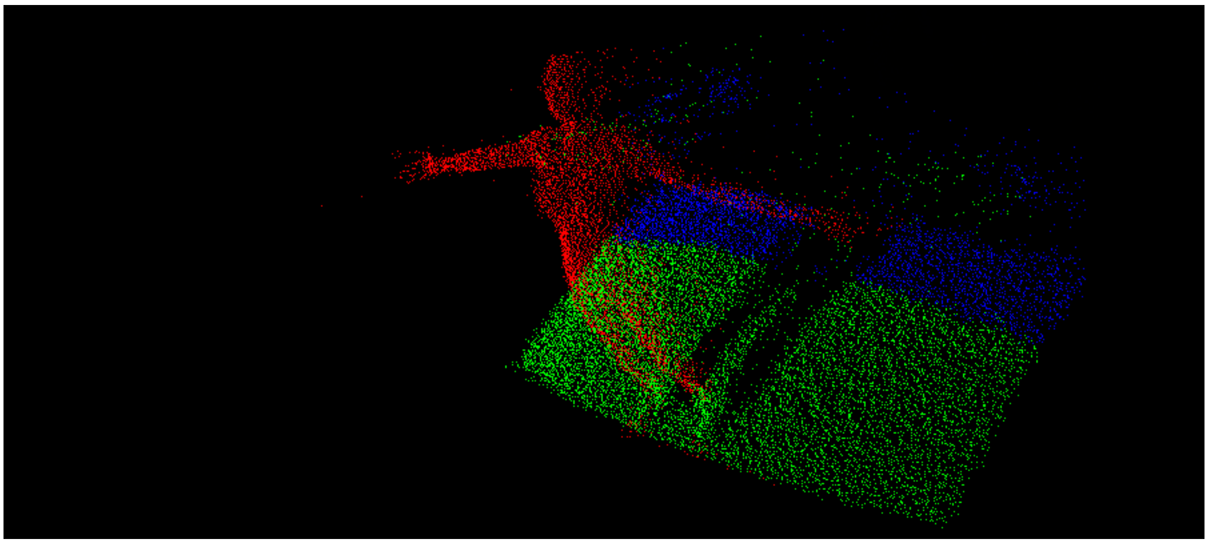


Figure 3.15: A frame captured with a high integration time. The visualization was done with OpenGL.

Kinect also suffers from image noise but with this camera there is no possibility of changing any settings externally. Since the Microsoft Kinect only has one IR emitter the camera suffers more easily from image occlusions than the PMD[vision] Camcube. When the observed object gets too close to the camera the region illuminated by the IR emitter cannot be captured by the camera anymore and so is marked as an unknown area. In the resulting image this area looks like a shadow or a double image. Figure 3.16 shows this effect.

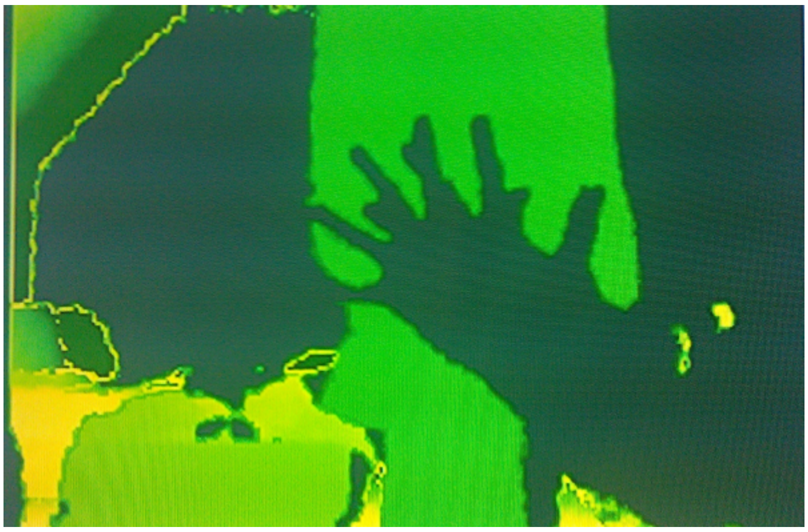


Figure 3.16: This image shows the shadow effect occurring when a captured object is located too near in front of the Microsoft Kinect. The captured hand can be seen twice.

3.7 Background Segmentation

The final step before combining the point clouds is background segmentation. Here the relevant foreground information is separated from the irrelevant background information in order to further process the data. When using point clouds for gesture recognition or tracking, the complete information gathered by the cameras can be disturbing. Also rendering and processing the additional amount of data is memory consuming. Therefore the process of background segmentation is an important step of the workflow.
In literature various techniques for background removal are described. Most of them use the color information contained in the frame in order to distinguish the foreground from the background. Other algorithms use a gradient or change in luminance to separate both areas [BHH11]. All techniques mentioned suffer from the restriction of having only two dimensional data available. For this work the depth information is used in order to pre-process the recorded data. The removal of the background is done in a two step approach:

1. Definition of a threshold for removing the information having a very high z-value.

2. Construction of a background model that is used for the remaining data.

The first step of the process is used to remove the main part of unneeded information. Mostly the range of the camera is limited by obstacles like walls, office utensils or doors. These objects generate a lot of unwanted information. When using a threshold this information can be removed easily. Depending on the distance between the foreground object and the obstacles the result of this method various heavily. When capturing an object that moves closely in front of a wall it is difficult to choose a suitable threshold. There are also difficulties when the captured object enters the area beyond the chosen threshold. In this case the foreground information is lost. For this reason a second method for background removal is implemented. This method uses a predefined background model which is compared with the actual frame. Each depth value in the actual frame is subtracted from the depth value in the background model. If the difference between the values lies beyond a predefined threshold, the pixel in the actual frame is marked as background. In order to generate a suitable background model the scene has to be captured before the moving object appears in front of the camera. The frame is stored and then used during the whole image capturing process which makes this method suitable only for static backgrounds. The result of this step are two point clouds containing only the foreground information.

3.8 Merged Views

The last step of the working process is the generation of a single point cloud being combination of the data gathered from both cameras. When capturing the data, both cameras deliver a two dimensional array of data. The PMD[vision] Camcube 3.0 delivers a 200 x 200 pixel point array and the Microsoft Kinect delivers a depth map with a resolution of 640 x 480 pixel. Both images have their own coordinate system. When trying to create a merged point cloud the image coordinates have to be transformed into 3D coordinates and then the two different coordinate systems have to be transformed to result in the same coordinate system. This process was previously described in chapter 2.1. First of all the calculation of the three dimensional data has to be performed. Since the depth value is known, the following formula can be used [Ope11a]:

$$X = \frac{u \cdot z}{fx} - \frac{z \cdot cx}{fx} \tag{3.3}$$

$$Y = \frac{v \cdot z}{fy} - \frac{z \cdot cy}{fy} \tag{3.4}$$

$$Z = depth(u, v) \tag{3.5}$$

Where (X, Y, Z) are the three dimensional coordinates of the captured point, u and v are the image coordinates, $c = (cx, cy)$ is the principal point, $f = (fx, fy)$ is the focal length and z is the depth value captured by the depth camera. As described in chapter 2.1 capturing a scene with a camera produces lens distortions. Once estimated the correction parameters for radial and tangential distortion can be integrated in the formula used above as follows:

$$x'' = \frac{X}{Z} \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 \cdot \frac{X}{Z}\frac{Y}{Z} + p_2(r^2 + 2 \cdot \left(\frac{X}{Z}\right)^2) \tag{3.6}$$

$$y'' = \frac{Y}{Z} \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1 \cdot (r^2 + 2\left(\frac{Y}{Z}\right)^2) + 2p_2 \frac{X}{Z}\frac{Y}{Z} \tag{3.7}$$

$$\text{with} \tag{3.8}$$

$$r^2 = \left(\frac{X}{Z}\right)^2 + \left(\frac{Y}{Z}\right)^2 \quad \text{and} \quad Z = depth(u, v) \tag{3.9}$$

As soon as the three dimensional coordinates are calculated for every camera used in the rig the transformation into the world coordinate system can be done with the following formula:

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = R \begin{bmatrix} x'' \\ y'' \\ z \end{bmatrix} + t$$

By using the rotation matrix and the translation vector estimated during calibration, the cameras are transformed in a way that both of them lie in the same coordinate system. The depth values captured by the camera are reprojected from the estimated reference points resulting in a point cloud that contains the captured values of both cameras. This combination process of multiple cameras makes it possible to reconstruct a three dimensional object.

CHAPTER 4

Implementation

This chapter covers the implementation of the combination of the depth cameras and the visualization of the results. In addition to the program a DLL (dynamic link library) has been implemented in order to make it possible to use the results of the calibration process for future work. For this work the IDE (integrated development environment) Visual Studio and the programming language C++ were used. The visualization of the combined views was done with the help of OpenGL (Open Graphics Library), the extension library GLEW (OpenGL Extension Wrangler Library) and the utility toolkit GLUT (OpenGL Utility Toolkit). The implementation of the camera handling was done with the libraries mentioned in chapter 3.

The program consists of multiple sections all handled by the Main class which is responsible for establishing the connection, handling the background subtraction and visualizing the results. The classes KinectConnection and PmdConnection treat the connection with the cameras and grab the frames on demand. The ConnectionHandler class is responsible for grabbing the actual frame from both cameras which makes it possible to update both cameras at the same time. The FrameHandler class lets the user save the active frame and the background image of all used cameras into files. It also applies the background model to the data and so calculates the foreground of the scene. The Matrix class handles all matrix operations like returning an identity matrix or matrix multiplication. The View class provides and calculates all parameters needed for the calibration of the cameras. The DisplayImage class contains all parts needed to display the result with OpenGL. In order to be able to use the results of the MIP-MCC software a configuration file is needed that contains all parameters calculated by the multiple camera calibration software. Finally the visual output and the resulting DLL are explained in the last section of this chapter.

4.1 Main class

The Main class is the starting point of the program calling up the DisplayImage class and the ConnectionHandler class. The main class contains instances of all other classes and handles a global variable that contains information about the visualization state which can be with and without showing the background. In contradistinction to the DLL the application contains a call on the DisplayImage class which contains the part that makes it possible to visualize the result of the merge. The methods used in this class are as follows:

- **InitializeConnection:** This methods initializes the ConnectionHandler instance and calls the initialization methods of the KinectConnection and PMDConnection classes.
- **GetNextFrame:** In order to grab a frame this method has to be called. It uses the update methods from the camera connection classes and returns the point cloud from the PMD depth camera and the point values from the Kinect depth camera.
- **TerminateConnection:** When ending the application it has to be assured that the connection of both depth cameras are being terminated in order to prevent malfunctions when starting the application again.

4.2 KinectConnection class

This class contains all functions that assure the correct execution and termination of the data acquisition process for the Kinect depth camera. The first method to call when establishing a connection with the Kinect camera is the **KinectInitialize** method which creates the connection between the camera and the application. In order to create a connection the OpenNI Framework has to be used creating a so called 'Context Object'. This object makes it possible to combine the production chains that are used for the application. Each production chain is a set of different production nodes used to create the required data. It is possible to create multiple instances of OpenNI context objects and so use different sensors at the same time as long as the OpenNI sensor module was implemented for the device. To use a context object it has to be initialized using the function **context.Init()**. Since the OpenNI framework not only allows to retrieve data from the sensor in real time but also to process previously recorded data streams, it is possible to load sensor data from a file having the file extension **.oni**. Loading data from a file the function **context.OpenFileRecording(filename)** has to be used. Instead of creating production nodes programmatically it is possible to load production nodes from a .xml file. Listing 4.1 shows an example of a .xml file created to produce a depth production node and an image production node. The tag **<Log>** configures the logging information which is set to 'Error' by default which means that for this configuration only errors are logged. The production node tags define all nodes used by this initialization. In doing so it is possible to define also specific configurations for each node like mirroring or cropping.

```
<OpenNI>
      <Log writeToConsole="false" writeToFile="false">
            <LogLevel value="3"/>
      </Log>
      <ProductionNodes>
            <Node type="Depth" name="DepthNode">
                  <Configuration>
```

```
                        <Mirror on="false"/>
                  </Configuration>
            </Node>
            <Node type="Image" name="ImageNode" stopOnError="
               false">
                  <Configuration>
                        <Mirror on="false"/>
                  </Configuration>
            </Node>
      </ProductionNodes>
</OpenNI>
```

Listing 4.1: The configuration file for production nodes

After initializing the context it is possible to access a specific node by calling the function **context.FindExistingNode()**. Calling the function it is possible to define the type of node to be found and saving the node into a generator object which generates a depth map or an image map. In addition to the data generated by the generator object additional information concerning the gathered data is provided. This metadata is saved into a so called 'Metadata object' and can be accessed with the help of the generator object. The metadata object contains information like resolution or the total number of the grabbed frames. In order to retrieve the depth map of the current frame it is necessary to call an update function. There are different update functions depending on the use of the application. If multiple context nodes are used in one application it can be desirable to update only specific context nodes. In order to update all used context nodes the function **context.WaitAndUpdateAll()** has to be used. It updates all production nodes as soon as the data is available. Each operation conducted by the production node returns a status value which indicates, if the desired function call has been successful or not. This status value can be used to implement error handling and so communicate the problem to the user. The last step in the process of data retrieving is the termination of the connection which is conducted by the function **context.Shutdown**. Listing 4.2 shows a simple example of the data acquisition process using the OpenNI framework:

```
context.InitFromXmlFile();

context.FindExistingNode(NodeType , depthgenerator );

context.WaitAndUpdateAll();

depthgenerator.GetMetaData();

depthgenerator.GetDepthMap();

context.ShutDown();
```

Listing 4.2: Connection and frame grabbing with the Kinect depth camera

Further information about the features of the OpenNI framework can be found in [Ope11b].

4.3 PmdConnection class

The PmdConnection class like the KinectConnection class contains all functions to ensure the correct connection handling and data acquisition. To initialize the connection with the PMD depth camera the function **pmdInitialize()** is called which opens a connection between the camera and the application. In order to do that a 'PMDHandle' object has to be created which handles the connection and all other functions concerning the depth data. The first function that has to be called is **PMDOpen()** which expects a pointer to a PMDHandle object and the plugin parameter. Like the OpenNI Framework the PMD plugins allow to record data and save it into a file that has a **.pmd** file extension. Such files can afterwards be loaded and handled in the application. In contrary to OpenNI the PMD plugin only requires a variable, passed during the process of opening a handle, that declares whether the data should be loaded from a file or be streamed during a real time connection with the camera. In order to grab a frame the **pmdUpdate()** function has to be called. This function has to be called every time a new frame is required. The actual frame can be received by calling the function **pmdgetSourceData**() which requires the handle and the size data which is returned by the function **pmdGetSourceDataDescription**. To get all distances captured in a frame the function **pmdGetDistances()** can be used. It returns a float variable with all distance values of the current frame. The last step is the termination of the connection which can be done with the function **pmdClose()**. It terminates the connection with the camera and deletes the handle. In order to be able to implement exception handling the PMD plugin supports multiple error descriptions. Each function returns an integer value that determines whether the current operation was successful or not. Listing 4.3 shows an example of a PMD Connection and the process of retrieving a frame.

```
pmdOpen(*PMDHandle,*CamcubePluginPath,*VideoPath,*
   CamcubeprocPluginPath);

pmdUpdate(PMDHandle);

pmdGetSourceDataDescription(PMDHandle, *PMDDataDescription);

pmdGetDistances(PMDHandle, *data, size);

pmdClose(PMDHandle);
```

Listing 4.3: Connection and frame grabbing with the PMD depth camera

Further information about the PMD functions can be found in [Gmb09].

4.4 FrameHandler class

The FrameHandler class is a helper class and makes it possible for the user to have a glance at the data by saving the depth values into a file. This can be very helpful if no data is visible for the user during the visualization . In addition it contains the determination of the background model and returns the edited frame to the DisplayImage class. The following functions are part of the FrameHandler class:

- **saveFrameToFile:** This function opens a file stream and iterates over the data. Every depth value is saved in a separate row.
- **saveKinFrametoFile:** In order to save depth data retrieved from the Kinect depth camera a file stream has to be opened. The data from the DepthMetaData object is stored in a two dimensional data structure. So a two dimensional iteration has to be done to grab all values.
- **calculateKinForeground:** This function calculates the foreground of the actual frame by applying the background model on it.
- **calculatePmdForeground:** In this function the active area of the actual PMD frame is calculated by using the background model.
- **MedianFilter:** This functions performs a filter operation of the current image using a Median Filter.

4.5 Matrix class

The Matrix class contains all operations that are necessary to operate with matrices and reduces the complexity of the code structure. A 4x4 matrix is a two dimensional array having each four values. The following functions are part of the Matrix class and describe matrix operations:

- **initalizeIdentity:** This function initializes an identity matrix by setting the diagonal values to one and the rest of the matrix to zero.
- **initializeNull:** This function creates a matrix object whose values are all set to zero.
- **initializeTranslation:** In order to be able to perform a translation, a translation matrix has to be created. As described in chapter 2.2 the translation part of the matrix is the last column of the matrix.
- **initializeRotationX:** Initializes a rotation around the x-axis about a certain angle.
- **initializeRotationY:** Initializes a rotation around the y-axis about a certain angle.
- **initializeRotationZ:** Initializes a rotation around the t-axis about a certain angle.
- **initializeScale:** This function defines a matrix that performs a scaling transformation. The scaling parameters are positioned in the diagonal of the matrix.

- **matrixMultRight:** Performs a matrix multiplication with the passed matrix on the right hand side of the multiplication.
- **matrixMultLeft:** Performs a matrix multiplication with the passed matrix on the left hand side of the multiplication.
- **transposeMatrix:** Transposes the passed matrix.
- **invertMatrix:** Calculates the inverted matrix by using the determinant of the passed matrix.
- **matrixPointMult:** Transforms a specific point by multiplying it with a matrix.

4.6 View class

This class contains all operations that are needed to perform the merging of the views. It contains the generation of the matrices for the intrinsic and extrinsic parameters and various conversion functions. All functions are listed below:

- **anglesToRotationMatrix:** Makes a conversion between angles and rotation matrices as described in the appendix.
- **getRotationMatrix:** Creates the rotation matrix used by the external transformation.
- **getIntrinsics:** Creates the matrix with the intrinsic parameters.
- **createRoll:** Calculates the roll angle from the quaternion delivered by the calibration software.
- **createPitch:** Calculates the pitch angle from the quaternion delivered by the calibration software.
- **createYaw:** Calculates the yaw angle from the quaternion delivered by the calibration software.

4.7 DisplayImage class

This class contains all OpenGL functions and makes it possible to visualize the results. It consists of the initialization of GLUT which handles the window generation, the keyboard, the mouse movement and all OpenGL operations like rotations or the visualization of points. The following list shows all functions of the DisplayImage class:

- **initialize:** In this function GLUT is initialized and used for determining window properties, keyboard , mouse and display functions.
- **draw:** This function visualizes the point clouds and a global reference coordinate system for better orientation.

- **handleKeyboard:** All operations that can be controlled by the user are implemented in this function including rotations, turning the background on and off, translations and the termination of the application.
- **handleMouse:** The control of the mouse is implemented in this function. The mouse is used to rotate the global view and to zoom in or out.
- **getMergedView** This function brings together all calculations and operations that are necessary to merge the two views generated by the depth cameras. The transformation from world coordinates into image coordinates is done by this function as well as the merge of the two different camera coordinate systems into one global coordinate system. The calculations from this function are the precondition for the draw function.

4.8 Configuration File

The configuration file is needed to be able to use the results of the MIP-MCC software. It contains all intrinsic and extrinsic parameters calculated by the software. Also it contains the thresholds for the background model and the paths for the PMD plugins and the OpenNI configuration .xml file. Listing 4.4 shows the structure of the configuration file using offline mode and listing 4.5 shows the structure of the configuration file in online mode.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <appSettings>

    <add key="Plugins" value="pmdfile.W32.pcp"/>

    <add key="Param" value="filename.pmd"/>

    <add key="PluginProc" value="camcubeproc"/>

    <add key="KinnectConnectionMode" value="offline"/>

    <add key="KinnectFile" value="filename.oni"/>

  </appSettings>

</configuration>
```

Listing 4.4: The configuration file for the offline application

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

```

```
  <appSettings>
    <add key="Plugins" value="camcube3"/>
    <add key="Param" value=""/>
    <add key="PluginProc" value="camcubeproc"/>
    <add key="KinnectConnectionMode" value="online"/>
  </appSettings>
</configuration>
```

Listing 4.5: The configuration file for the online application

4.9 DepthCamCon Library

The purpose of the dynamic link library named 'DepthCamCon' created for this work is the usage of the data in future work. The merged point cloud can be used for motion tracking or reconstruction purposes. The library delivers the transformed point clouds separately so the user has the possibility to work with the individual views as well as with the merged view. In addition it is possible to control the process of grabbing the frames at specific time intervals. This functionality is very useful for computationally intensive applications that do not run in real time. In order to support this feature an additional class was implemented named 'ConnectionHandler'. This class handles the connections of both cameras, the timing of the acquisition of the new frame and the termination of the connections. In addition it provides meta data of both data streams that could be useful for the user. The library delivers only the calibrated data with the background removed from the scene. The following list shows all functions of the ConnectionHandler class:

- **InitializeKinect:** Initializes the connection with the Kinect depth camera.
- **InitializePmd:** Initializes the connection with the PMD depth camera.
- **getNextFrameKinect:** Updates the context node and grabs the next frame.
- **getNextFramePMD:** Updates the PMDHandle and grabs the next frame.
- **TerminatePmdCon:** Terminates the connection with the PMD depth camera.
- **TerminateKinCon:** Terminates the connection with the Kinect depth camera.
- **getPmdWidth:** Gets the amount of values in x direction for the PMD depth camera.
- **getPmdHeight:** Gets the amount of values in y direction for the PMD depth camera.

- **getKinWidth:** Gets the amount of values in x direction for the Kinect depth camera.
- **getKinHeight:** Gets the amount of values in y direction for the Kinect depth camera.

The usage of the library is very simple and done in a few lines of code as listing 4.6 shows with help of a simple example of grabbing a frame from both cameras.

```

connectionHandler->InitializePmd();
connectionHandler->InitializeKinect();


//for every new frame these two functions have to be called

float* pmdFrames= connectionHandler->getNextFramePMD();
float* kinFrames= connectionHandler->getNextFrameKinnect();


//the x,y and z values of the first pixel are stored one after another
//the size of the arrays is height*width*3 so 3 values for every pixel are stored
//these variables can be used for a loop in order to get all coordinates for the actual frame

int allPmdCoordinates=con->getPmdWidth()*connectionHandler->getPmdHeight() *3;
int allKinCoordinates=con->getKinWidth()*connectionHandler->getKinHeight() *3;

//getting the first x,y,z-coordinate of the actual PMD frame

int i=0;
float pmdFramex1=pmdFrames[i];
float pmdFramey1=pmdFrames[i+1];
float pmdFramez1=pmdFrames[i+2];

//getting the first x,y,z-coordinate of the actual Kinect frame

int j=0;
float kinFramex1=kinFrames[j];
float kinFramey1=kinFrames[j+1];
float kinFramez1=kinFrames[j+2];


connectionHandler->TerminatePmdCon();
connectionHandler->TerminateKinCon();
```

Listing 4.6: The usage of the DepthCamCon library

CHAPTER 5

Results

In this chapter the result of every single work step is presented, showing that every step has a significant influence on the successive step. In the first section the calibration of the depth cameras is described which influences the results of all other steps. Without precise intrinsic and extrinsic parameters a successful merging process of the depth data cannot be made. Because of some difficulties arising from the calibration of the depth cameras, the final result of the merging process was not as expected. This section describes the problems that occurred during calibration and the influence this calibration troubles had on the resulting point cloud. The second section describes the capturing process with the cameras and the resulting raw data without applying any image processing tasks. The third section briefly covers the influence of varying input settings and the process of eliminating the background information. In the last section the result of the merging process is presented by visualizing the point cloud using OpenGL.

5.1 Calibration

The calibration process consists of three steps each of them building on the previous one. The first step of the calibration is the image generation. As described in chapter 2.1, intensity and depth images have to be taken from each camera. The system configuration and components used for image generation are shown in table 5.1. In order to produce meaningful calibration data the checkerboard has to be visible in each image. Also the checkerboard has to captured from different poses, distances and angles in order to get more accurate results. In order to receive results as precise as possible the intrinsic parameters of the cameras have to be calibrated separately. This makes it possible to generate various checkerboard poses without paying attention to occlusions regarding the viewing angle of the second camera. The following pictures with corresponding image lists where made in order to calibrate the intrinsic camera parameters:

- **Infrared image list for the Microsoft Kinect:** In order to generate an image list for infrared images twenty pictures where taken from different poses. In order to find an

System Component	Used Component
Operating system	Windows XP 32bit
Kinect driver	Windows Kinect Driver
Kinect Plugin	OpenNI 1.5.2.23
PMD driver	pmdcamcube
PMD plugin	camcube.W32.pap, camcubeproc.W32.ppp
Software for image generation	biasShowCamWx.exe
Software for calibration	MIP-MCC 1.0.0

Table 5.1: System configuration and used components for Microsoft Kinect and PMD[vision] Camcube 3.0

image having sparse interferences but enough light intensity to be able to find the corners of the checkerboard, the lighting condition has to be taken into account.

- **Depth image list for the Microsoft Kinect:** The image list for the depth images corresponds to the image list for the infrared images and so it is important that both lists contain the same image from the same pose. In order to achieve that the checkerboard should not be moved during image generation. Also the same amount of images has to be taken and saved into the image list. Figure 5.1 shows a sequence of the image lists generated for the Microsoft Kinect containing infrared images and the corresponding depth images. It can be seen that defining the corners of a single image can be very difficult because the illumination of the image can be quite unsatisfactory.

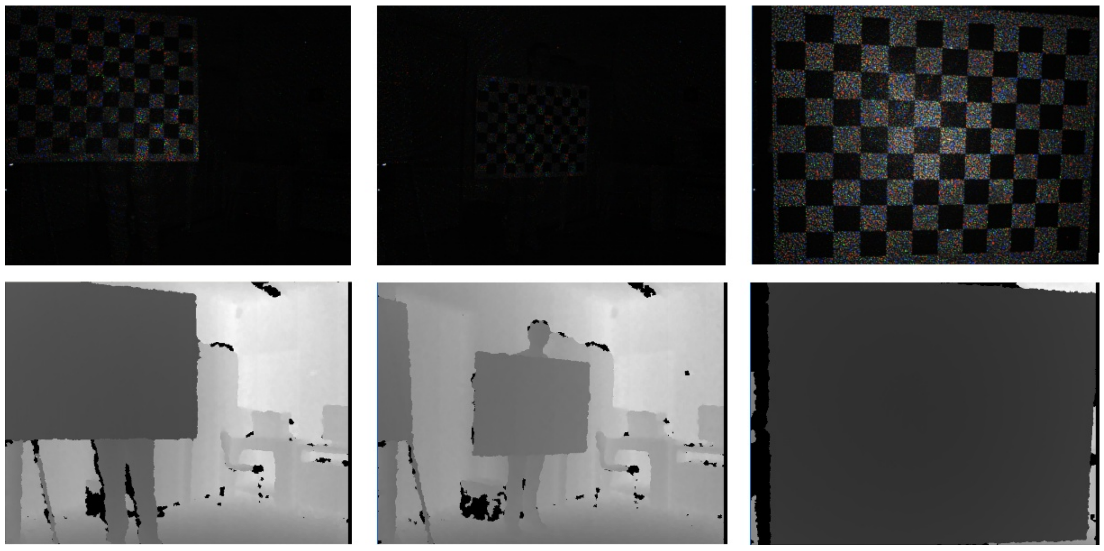


Figure 5.1: A sequence of infrared and depth images taken with the Microsoft Kinect

- **Infrared image list for the PMD[vision] Camcube 3.0:** Taking images with the PMD[vision]

Camcube 3.0, four kinds of images are created namely amplitude images, grey value images, intensity images and depth images. For the calibration only intensity (or grey value) and depth images are needed. In order to calibrate the intrinsic parameters of the PMD[vision] Camcube 3.0 twenty pictures are taken from various angles and distances. Compared with the Microsoft Kinect the PMD[vision] Camcube 3.0 produces images with less resolution but makes the identification of the corners more comfortable because they can be identified easier in the intensity image than in the infrared image the Microsoft Kinect produces.

- **Depth image list for the PMD[vision] Camcube 3.0:** The depth image list for the PMD[vision] Camcube 3.0 is produced in the same way as for the Microsoft Kinect so twenty depth images corresponding to the intensity images have to be taken. Figure 5.2 shows a sequence of intensity and depth images made with the PMD[vision] Camcube 3.0.

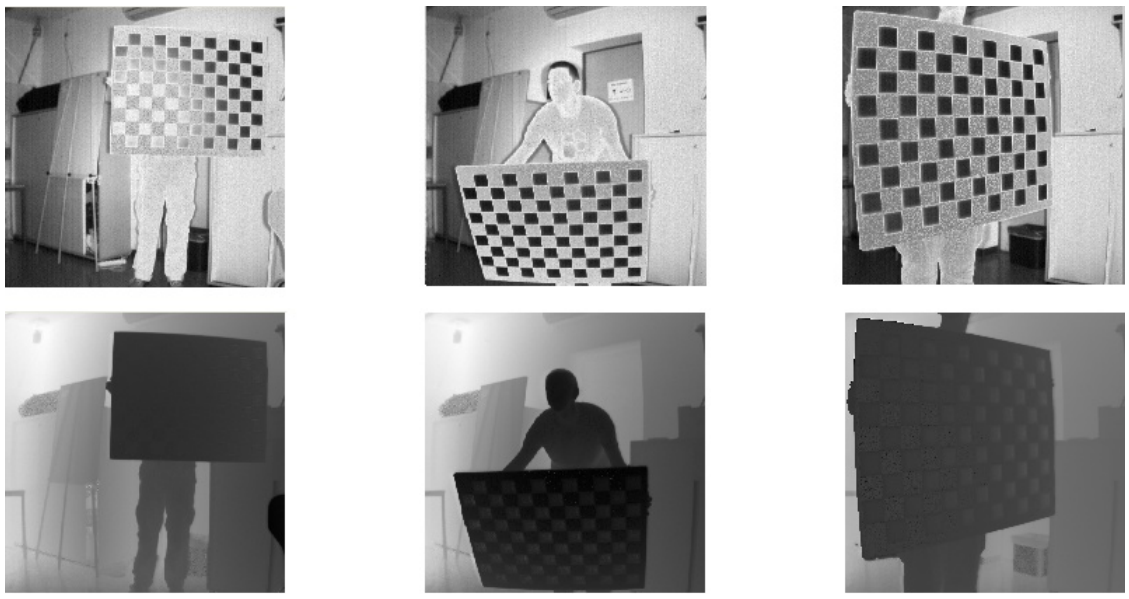


Figure 5.2: A sequence of intensity and depth images taken with the PMD[vision] Camcube 3.0

After creating the image lists the structure of the checkerboard pattern has to be specified. For the calibration of the depth cameras a checkerboard pattern consisting of 12x9 squares is used. Each square has a size of 80 millimeters. Table 5.2 and table 5.3 show the result for the calibration of the intrinsic parameters of the Microsoft Kinect and the PMD[vision] Camcube 3.0 depth cameras:

In figure 5.4 and 5.3 reprojection images of the calibration process of the PMD[vision] Camcube 3.0 and the Microsoft Kinect created by the MIP-MCC software are shown.

After calibrating the intrinsic parameters for both cameras the extrinsic parameters of the rig can be calibrated. This is being done by starting the calibration process with fixed intrinsic

Intrinsic Parameter	**Result**
Focal length	589.87
Aspect Ratio	0.99
Principal Point $C = (C_x, C_y)$	(342.77,248.40)

Table 5.2: Example results of the calibration process of the intrinsic parameters using the Microsoft Kinect

Intrinsic Parameter	**Result**
Focal length	301.40
Aspect Ratio	0.99
Principal Point $C = (C_x, C_y)$	(107.75,103.88)

Table 5.3: Results of the calibration process of the intrinsic parameters using the PMD[vision] Camcube 3.0

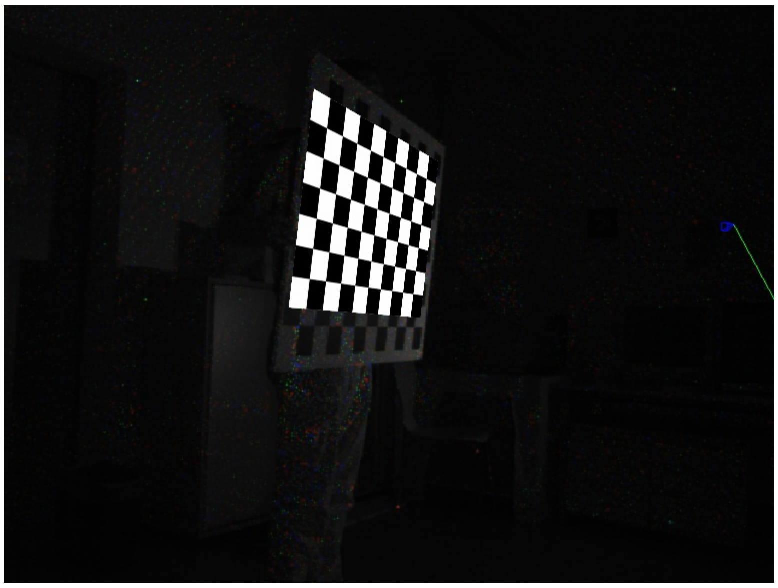


Figure 5.3: This image shows the reprojected calibration pattern for the Microsoft Kinect

parameters from the previous step. For this calibration step four image lists are used namely one intensity image list captured with the PMD[vision] Camcube 3.0, one infrared image list captured with the Microsoft Kinect and two depth image lists captured with both cameras. When calibrating the extrinsic cameras it is vital that the calibration pattern is visible in both images and that during the image retrieval process the calibration pattern stays at the same place. Figure 5.5 shows an example of an image pair taken with both cameras simultaneously. Here it can be seen that taking pictures with both cameras is quite challenging because of the different resolutions of the cameras. This makes it difficult to assure that the checkerboard pattern is visible in every

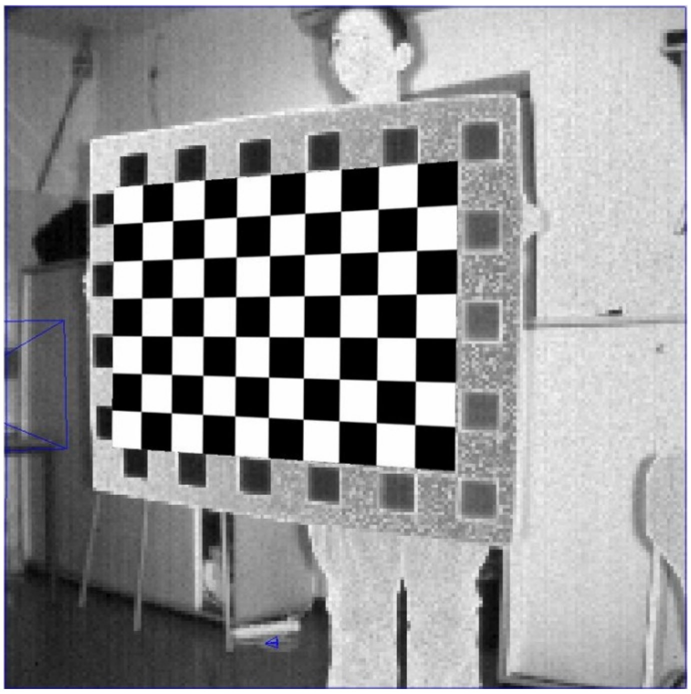


Figure 5.4: This image shows the reprojected calibration pattern for the PMD[vision] Camcube 3.0

image. After creating the image lists for both cameras the corners have to be assigned in order to be able to calibrate the extrinsic parameters. Table 5.4 shows the result of the calibration process and in figure 5.6 the reprojection of the calibration pattern from every depth camera is shown. The x, y and z values describe the translation in relation to the first camera and the rotation of one camera in respect to the other is described as a quaternion.

Extrinsic Parameter	**Result**
tx	2136.47 mm
ty	207.76 mm
tz	3034.87 mm
s	0.018
a	-0.738
b	0.027
c	0.67

Table 5.4: Results of the calibration process of the extrinsic parameters using the PMD[vision] Camcube 3.0 and the Microsoft Kinect. The unit of the translation part of the parameters is millimeter. The rotation part of the parameters is given as a quaternion (for more information see 6.3)

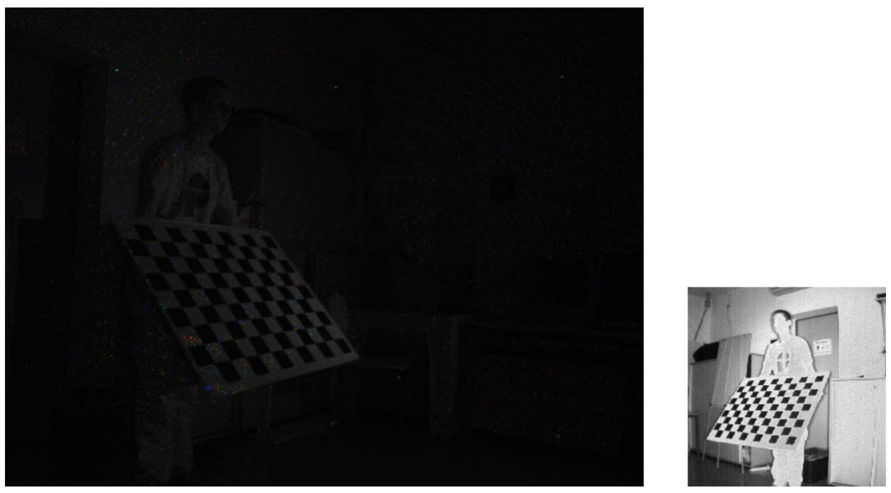


Figure 5.5: This image shows a scene captured simultaneously by both cameras

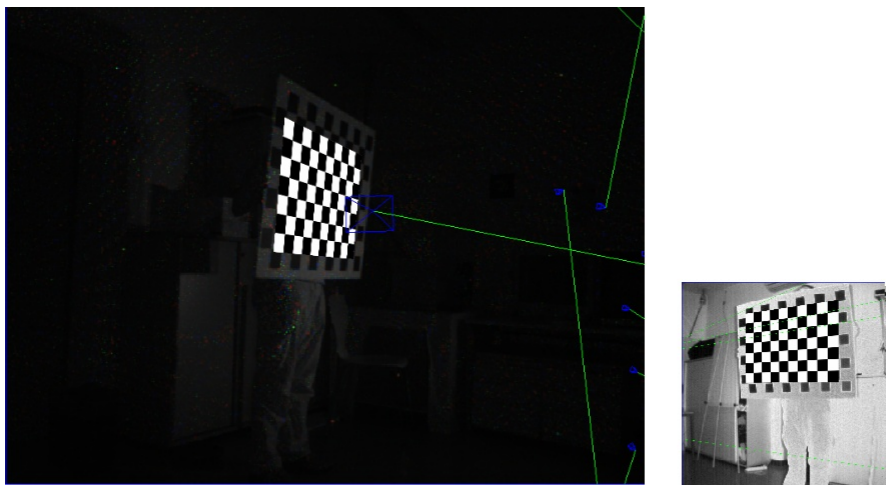


Figure 5.6: This image shows the reprojected calibration pattern for the same scene captured by both depth cameras

The calibration process has been a very challenging part of this work. Several tries to calibrate both cameras have been made producing different results for the extrinsic and intrinsic parameters although the same pictures have been used. Not knowing the exact parameters complicates error handling in the subsequent steps because the error source cannot be spotted certainly. Figure 5.7 shows an example of a failed reprojection.

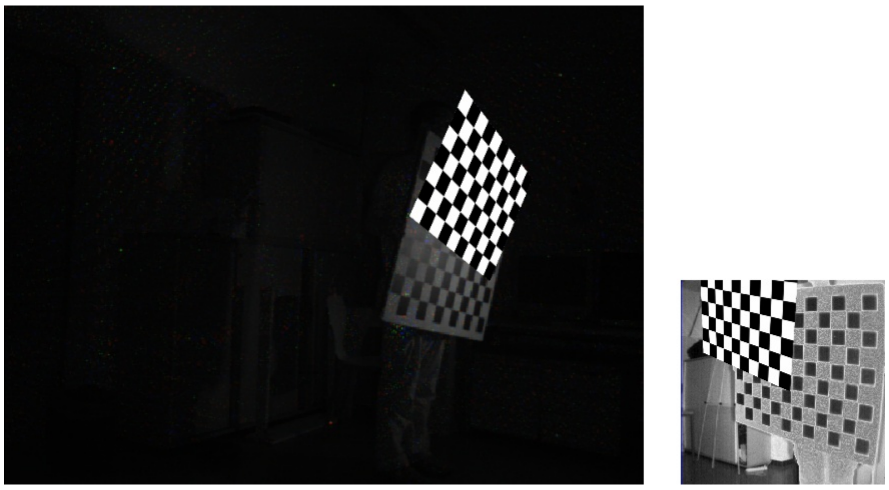


Figure 5.7: This image shows the reprojected calibration pattern for the same scene captured by both depth cameras. The calibration pattern is out of alignment and so falsifies the result of the calibration.

5.2 Raw Data

In this section the process of data acquisition and the output of this step is described. As mentioned in chapter 3.5 every camera has its own plugin for capturing data. During the process of data capturing different system configurations have to be used in order to retrieve images for the calibration process and a video stream for the reconstruction process. The components and system configurations used in order to be able to capture data with both cameras are shown in table 5.5: With the used system configuration it is possible to either use a recorded video stream

System Component	Used Component
Operating system	Windows XP 64bit
Kinect driver	SensorKinect from Primesense
Kinect Plugin	OpenNI 1.5.2.23
PMD driver	pmdcamcube
PMD plugin	camcube.W32.pap, camcubeproc.W32.ppp, pmdfile.W32.pcp

Table 5.5: System configuration and used components for Microsoft Kinect and PMD[vision] Camcube 3.0

for both cameras or to use a live stream. The software used for capturing a video stream was Camvis 3.0 for the PMD depth camera and OpenNI NiViewer for the Microsoft Kinect depth camera. The depth data was then retrieved with the algorithms described in chapter 4. The result of this step is a two dimensional image per camera with a depth value assigned to each pixel. Figures 5.8 and 5.9 show the result of this process.

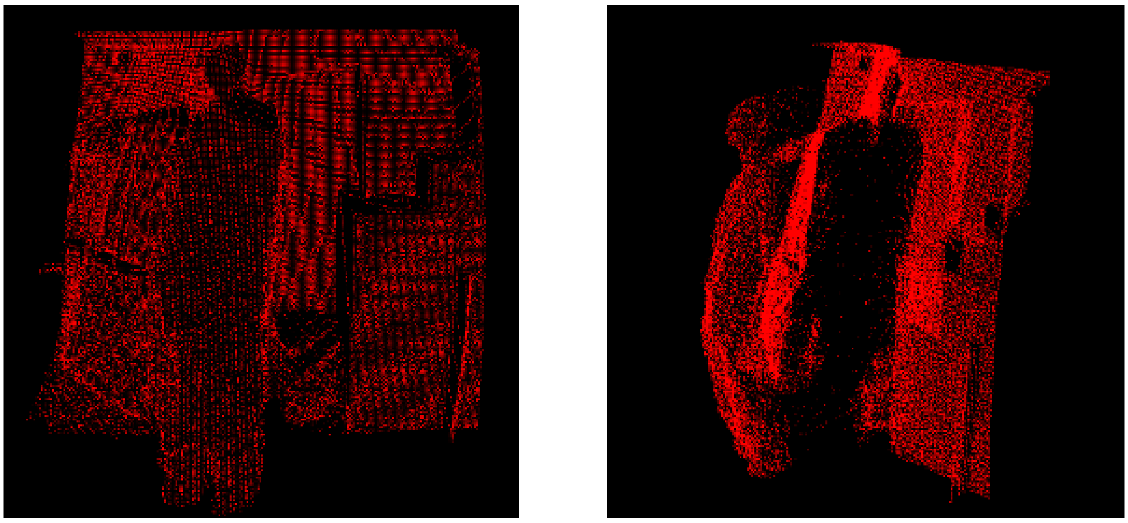


Figure 5.8: The raw output of the PMD[vision] Camcube 3.0 visualized from different viewing angles with OpenGL.

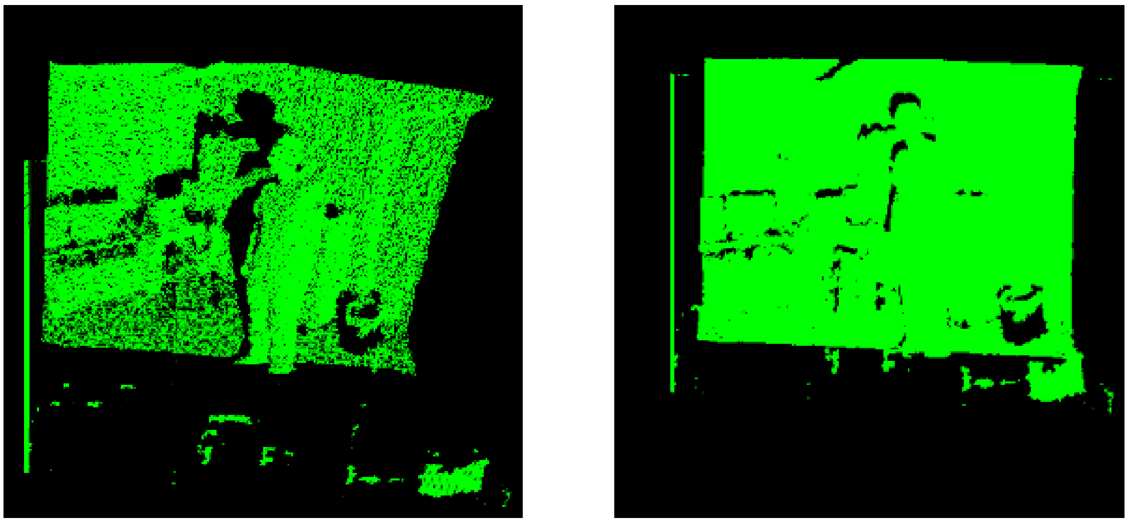


Figure 5.9: The raw output of the Microsoft Kinect visualized from different viewing angles with OpenGL.

5.3 Noise Reduction and Background Subtraction

This section shows the result of the used background subtraction method and the noise reduction using a Median Filter. As described in section 3.7 a background model is generated in order to subtract the background information from the foreground information. This background information is generated by capturing the first frame of the stream and storing it. For this method it is important to first capture the empty room because the model acts as a mask. A background model containing foreground information would distort the resulting image. After creating the background model the actual frame is subtracted from the model. For this operation a threshold is used which makes is possible to adjust the result of the subtraction by taking the distance between the observed object and the background into account . Figure 5.10 and 5.11 show the result of the background subtraction method for the PMD[vision] Camcube and the Microsoft Kinect. The removal of the background cuts away a large part of unwanted information but as

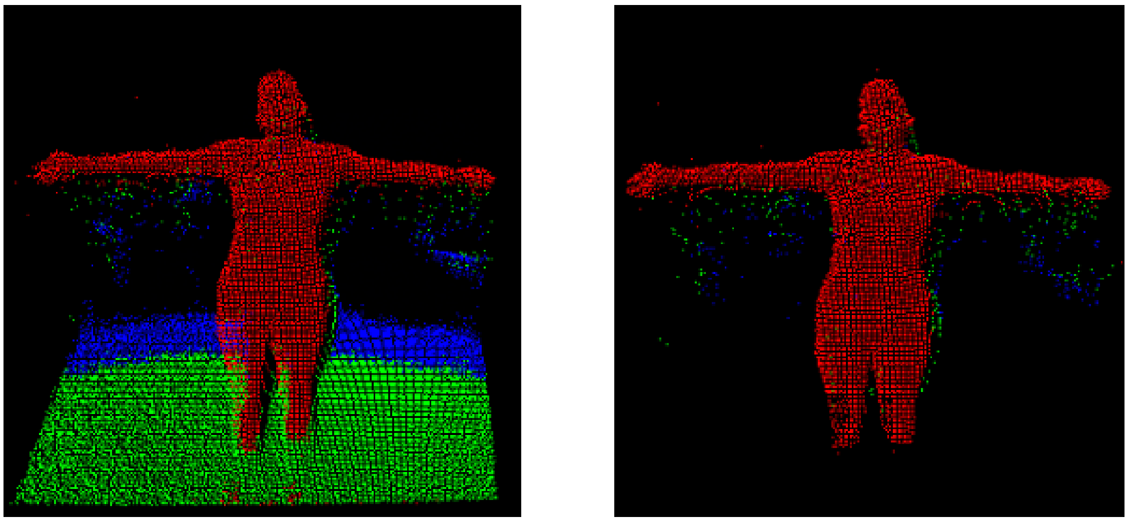


Figure 5.10: Background subtraction using the PMD[vision] Camcube.

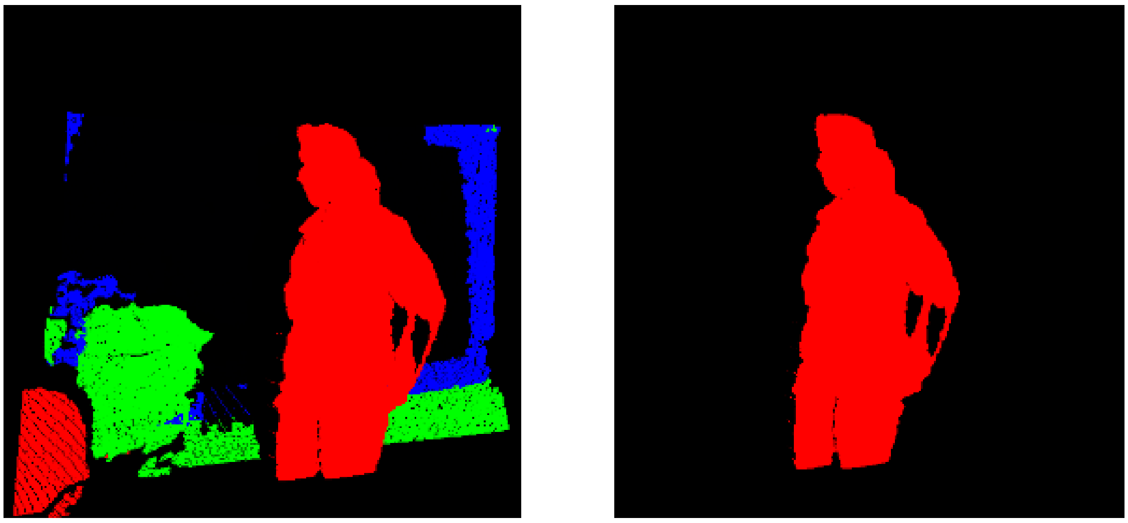


Figure 5.11: Background subtraction using the Microsoft Kinect.

mentioned before in section 3.6 especially the PMD[vision] Camcube suffers from noise caused by incoming sunlight. A simple and effective method to reduce outlier is the appliance of a Median filter. For this work a Median Filter having a $9x9$ filter window is used for sorting the neighboring values. After that the middle value of the sorting operation is used for the actual pixel value. In order to handle the edges of the input image correctly the value of the edge is copied to the left and the right of the value respectively otherwise the resulting image would be distorted. The result of this step is a smoothed image with reduced noise. Figure 5.12 shows the result of this step for the PMD[vision] Camcube.

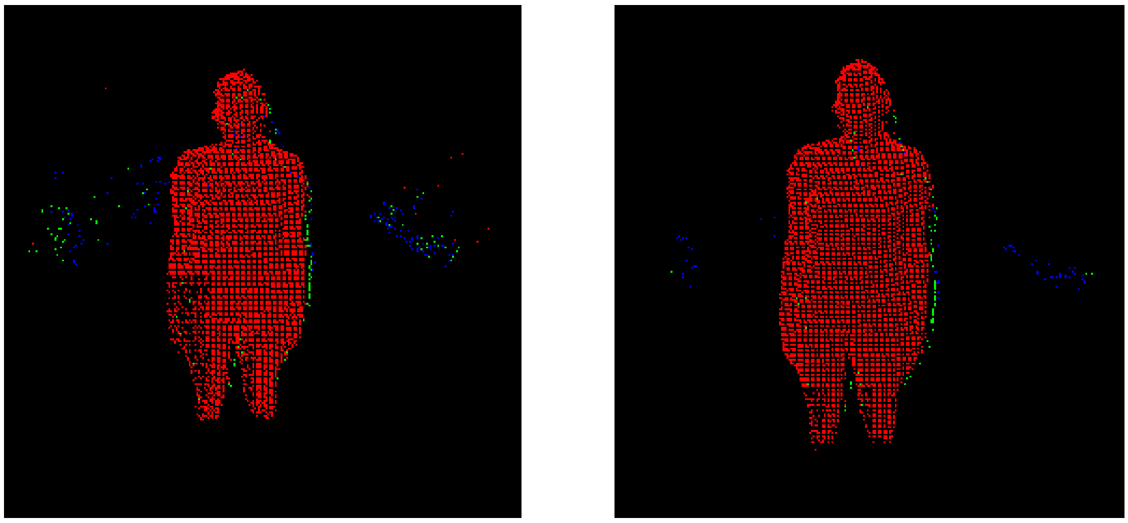


Figure 5.12: Applying a $9x9$ Median Filter on a frame captured with the PMD[vision] Camcube.

5.4 Merged Point Cloud

After having found appropriate values for generating input data and subtracting the background from the foreground information, the result are two 2D images containing depth information for each pixel value. The images have different resolutions namely $200x200$ pixel for the PMD[vision] Camcube and $640x480$ pixel for the Microsoft Kinect. Also the images are captured with different frame rates because the PMD[vision] Camcube is able to capture 40 images per second whereas the Microsoft Kinect only captures 30 frames per second. The intrinsic and extrinsic parameters estimated during calibration are used in this working step considering the different image resolution and frame rate as mentioned before. When merging the point clouds using the formulas described in 3.8, it could be seen that the problems occurred during calibration severely influence the resulting merged point cloud. While the result of the three dimensional reconstruction using the intrinsic parameters gained during calibration seems to produce quite good results the transformation into world coordinates fails. The result of the extrinsic calibration produces very imprecise values with divergences up to 1.3 meters. Various attempts were made using different lighting conditions and input images always resulting in unusable values for the extrinsic parameters. Since the source code of the used calibration software was not available at the time the experiments were made, there was no possibility of analyzing the source of error in order to be able to correct it. Another difficulty using different types of depth cameras is the different frame rate. Since there is no possibility of changing the frame rate it was not possible to synchronize both cameras. Figure 5.13 shows the result of the intrinsic calibration and figure 5.14 shows the result of the extrinsic calibration step. As it can be seen the two individual point clouds do not match which is a result of the imprecise extrinsic parameters.

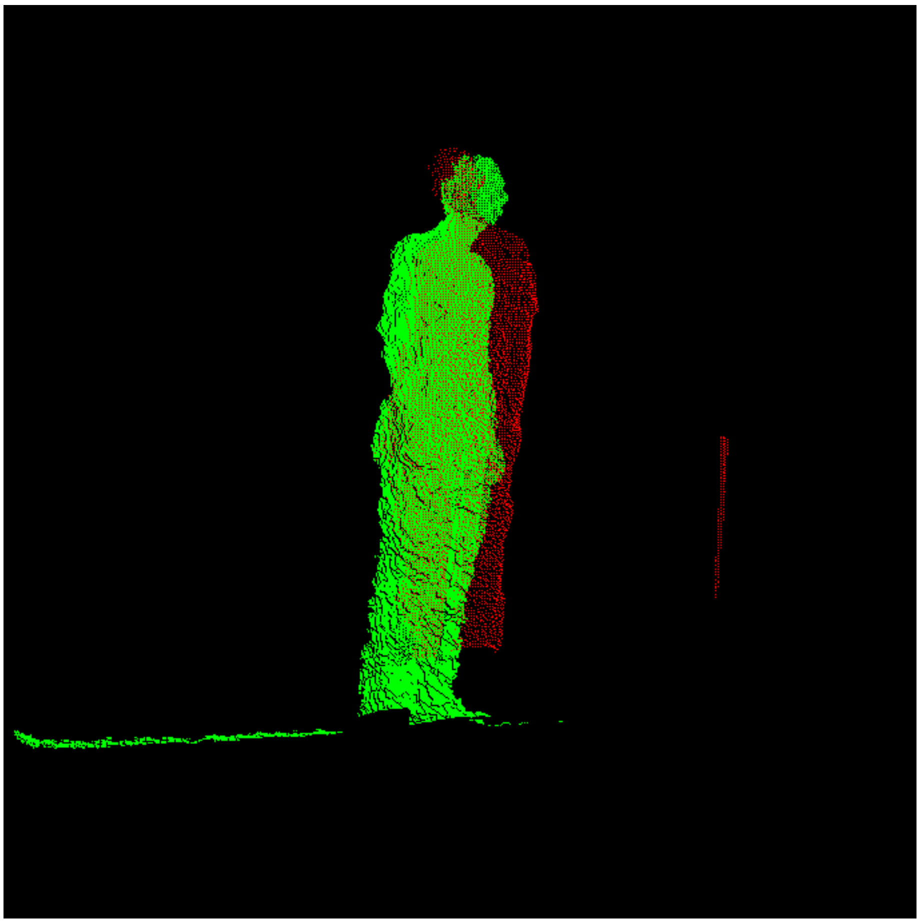


Figure 5.13: This image shows the result of the intrinsic calibration performed on both point clouds

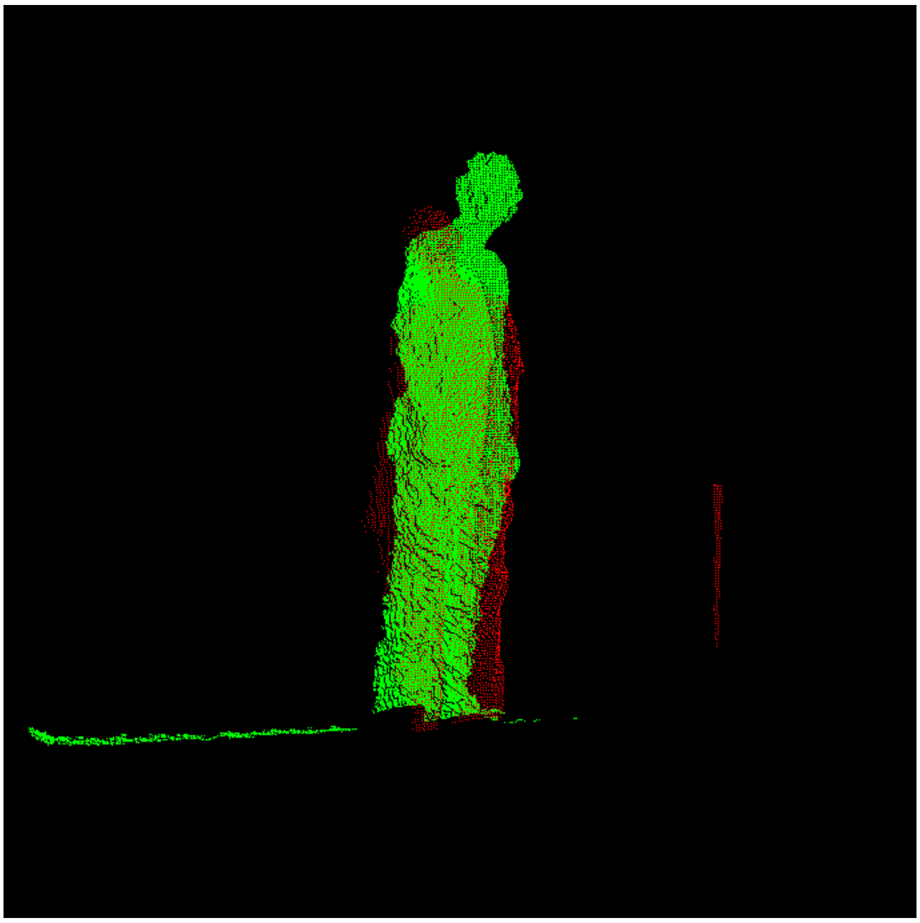


Figure 5.14: This image shows the result of the extrinsic calibration performed on both point clouds

CHAPTER 6

Conclusion and Future Work

6.1 Improvements and Future Tasks

As described in chapter 5 the result of the calibration have a grave influence on the following steps of the process. Without having precise calibration data the combination of the depth data can not be done. In order to improve the results of this step another calibration tool should be evaluated or an algorithm that handles this step of the working process should be implemented. In order to achieve better reconstruction results merging algorithms like ICP (iterative closest point) should be explored as a future work. Because these steps of the process are very vital and labor intensive it should be seen as stand alone fields of research.
Another problem is the use of different camera types and different capturing technologies. Every capturing technology has its own drawbacks. When using different types of cameras all these drawbacks have to be taken into account and the developer runs the risk of loosing the sight of the actual application by only handling fault correction. On the other hand using the same capturing technology can also cause severe problems. For example the usage of multiple time of flight cameras can cause signal overlapping and so falsify the result of the capturing process. The same effect occurs when using a depth camera like the Microsoft Kinect where an IR pattern is projected onto the scene.When using multiple cameras of this type the projected pattern overlaps and so can cause a misinterpretation of the data. As a future work this aspect should be explored.
In order to improve background segmentation other features like intensity values or gradients could be taken into account. The effectivity of the algorithm depends of the used capturing technology so an algorithm that takes this into account would be useful.

6.2 Using Multiple Depth Cameras for Tracking

The application fields of using multiple depth cameras are versatile but besides reconstructing a three dimensional object the intended use of this work is three dimensional tracking. Once calibrated a stable camera rig can be used for various purposes. The camera stream can either be recorded to use it for motion analysis or the movement of the captured object can be used as a basement for creating three dimensional objects. Especially in computer graphics or the movie industry where realistic movements of characters play an important role this three dimensional records can be very useful. For these application fields the performance of the system is not so important because the quality of the results of the tracking algorithms and the results of the motion analysis come to the fore. There are other fields of application where both the performance of the tracking algorithms and the performance of the data enhancement algorithms play a very important role. Considering applications where the tracked movement is used as an input for a successive action the performance of the system is vital. Such tracking system could be used for the gaming and entertainment industry where the tracked movements can be used for steering a character in a computer game or for turning on the television with a special gesture. The benefit of using multiple depth cameras is that the observed person can move freely across the room without paying attention to the viewing angle of a single camera. Also the additional depth information can be easily used to cut away unwanted information.

6.3 Conclusion

Combining multiple depth cameras is a challenging but interesting work. In this work the mathematical and technological background for combining depth cameras has been explained. The whole working process from the connection of the cameras to the combination of the processed data has been specified and analyzed. On base of this work an application that gathers the data and combining them to a single point cloud has been developed with the possibility of using it not only for offline but also for real time applications. Unfortunately it was not possible to merge the point clouds due to bad calibration results but using the proposed improvements satisfying results certainly can be found in future.

Appendix

This section gives an short description of the mathematical background that is relevant for the implementation.

Quaternions

Quaternions are an extension of complex numbers into higher dimensions containing one real part s and three imaginary parts i,j and k [DDH04]:

$$q = s + ia + jb + kc \quad \text{with} \quad a, b, c \in \Re \tag{6.1}$$

The imaginary parameters can be defined as followed [DDH04]:

$$i^2 = j^2 = k^2 = -1, \quad ij = -ji = k \tag{6.2}$$

Instead of using euler angles for the representation of rotation transformation quaternions can be used because of their simplicity. A rotation using quaternions can be represented by using a unit quaternion q containing the unit vector u:

$$q = (s, v) \tag{6.3}$$

$$s = \cos\frac{\theta}{2}, \quad v = u \sin\frac{\theta}{2} \tag{6.4}$$

The unit vector u refers to a specific rotation axis and θ describes the rotation angle around the selected axis. Using quaternions the rotation of a point P that results in the rotated point p' can be formulated as below [DDH04]:

$$P' = (0, p') \tag{6.5}$$

$$p' = s^2 p + v(p \cdot v) + 2s(v \times p) + v \times (v \times p) \quad \text{with} \quad p = (x, y, z) \tag{6.6}$$

This term can be evaluated by using the following rule for quaternion multiplication [DDH04]:

$$q_1 q_2 = (s_1 s_2 - v_1 \cdot v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2) \tag{6.7}$$

Euler angles

Euler angles can be used to describe rotations in any direction. Basically there are three angles of rotation called pitch, roll and yaw [DP02]. Yaw defines the rotation around the y-axis, pitch describes the rotation around the x-axis and roll measures the rotation around the z axis. The three vectors can be defined as followed [Sab08]:

$$\mathbf{R}_\phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$$\mathbf{R}_\theta = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$\mathbf{R}_\psi = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Where ϕ refers to pitch, θ refers to yaw, and ψ refers to roll.

Conversion between euler angles and rotation matrices

A single rotation matrix that describes the rotation of an object in x-, y- and z-direction can be constructed by building up three independent rotation matrices for each direction and then combining them into the final rotation matrix [DP02]. The multiplication of matrices is not commutative and thats why multiplying the individual matrices in a different order yields to different rotation matrices. For this work, the OpenGL convention is used where all rotations are performed counterclockwise, the up vector is the positive y-axis and the viewing direction is the negative z-axis. Using this convention a rotation around first the x axis then the y axis and last the z axis is defined as followed:

$$R = R_\phi R_\theta R_\psi \tag{6.8}$$

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.9}$$

$$R = \begin{pmatrix} \cos\Theta\cos\Psi & -\cos\Theta\sin\Psi & \sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi + \cos\Phi\sin\Psi & -\sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & -\sin\Phi\cos\Theta \\ -\cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi + \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{pmatrix} \tag{6.10}$$

Conversion between quaternions and rotation matrices

The conversion from a quaternion to a rotation matrix can be performed by using the rules of quaternion multiplication defined in chapter 6.3. The resulting matrix $R_{p'}$ that rotates around a chosen axis can be seen in the following:

$$\mathbf{R_{p'}} = \begin{pmatrix} 1-2b^2-2c^2 & 2ab+2sc & 2ac-2sb \\ 2ab-2sc & 1-2a^2-2c^2 & 2bc+2sa \\ 2ac+2sa & 2bc+2sa & 1-2a^2-2b^2 \end{pmatrix} \tag{6.11}$$

The terms (a, b, c) are the vector part and s is the scalar part of the quaternion. The final rotation matrix can be constructed by substitution of the s, a, b, c parts of the quaternion resulting in the following matrix [DDH04]:

$$\mathbf{R_{p'}} = \begin{pmatrix} u_x^2(1-\cos\theta)+cos\theta & u_xu_y(1-cos\theta)-u_z\sin\theta & u_xu_z(1-cos\theta)+u_ysin\theta \\ u_yu_x(1-cos\theta)+u_zsin\theta & u_y^2(1-cos\theta)+cos\theta & u_yu_z(1-cos\theta)-u_xsin\theta \\ u_zu_x(1-cos\theta)-u_ysin\theta & u_zu_y(1-cos\theta)+u_xsin\theta & u_z^2(1-cos\theta)+cos\theta \end{pmatrix} \tag{6.12}$$

The vector $u = (u_x, u_y, u_z)$ is the unit vector used in the quaternion representation. The matrix $R_{p'}$ is constructed using the following substitutions [DP02]: $w = \cos\frac{\theta}{2}$
$x = u_x \sin\frac{\theta}{2}$
$y = u_y \sin\frac{\theta}{2}$
$z = u_z \sin\frac{\theta}{2}$

Conversion between quaternions and euler angles

The conversion from quaternions to euler angles can be conducted by using the following equations using the terms from the matrix $R_{p'}$ [DP02]:

$$\phi = \arcsin(-(2bc+2sa)) \tag{6.13}$$

$$\theta = \begin{cases} \arctan(2ac-2sb, 1-2a^2-2b^2) & \text{if } \cos(\phi) \neq 0, \\ \arctan(-2ac-2sb, 1-2b^2-2c^2) & \text{otherwise} \end{cases} \tag{6.14}$$

$$\psi = \begin{cases} \arctan(2ab+2sc, 1-2a^2-2c^2) & \text{if } \cos(\phi) \neq 0, \\ 0 & \text{otherwise} \end{cases} \tag{6.15}$$

For more information about derivations and conversion of different rotation representations see [DP02].

Homogeneous coordinates

Homogeneous coordinates are meaningful representations and can be used to express transformations as matrix multiplications [DDH04]. To accomplish that a three dimensional point $P = (x, y, z)$ has to be expanded to a four dimensional point $P' = (xw, yw, zw, w)$, where $w \neq 0$ is called the homogeneous parameter. This parameter has to be chosen in a way that the following condition is fulfilled [DDH04]:

$$x = \frac{xw}{w} \quad , \quad y = \frac{yw}{w} \quad \text{and} \quad z = \frac{zw}{w} \tag{6.16}$$

For reasons of simplification $w = 1$ is often chosen.

Bibliography

[BHH11] S. Brutzer, B. Hoferlin, and G. Heidemann. Evaluation of background subtraction techniques for video surveillance. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1937 –1944, june 2011.

[CBS00] Frank Chen, Gordon M. Brown, and Mumin Song. Overview of three-dimensional shape measurement using optical methods. *Optical Engineering*, 39(1):10–22, 2000.

[CMC$^+$06] S. Corazza, L. Mündermann, A.M. Chaudhari, T. Demattio, C. Cobelli, and T.P. Andriacchi. A markerless motion capture system to study musculoskeletal biomechanics: Visual hull and simulated annealing approach. *Annals of Biomedical Engeneering*, 34:1019–1029, 2006.

[CTPD08] Ryan Crabb, Colin Tracey, Akshaya Puranik, and James Davis. Real-time foreground segmentation via range and color imaging. *Computer Vision and Pattern Recognition Workshop*, 0:1–5, 2008.

[DDH04] M. Pauline Baker Donald D. Hearn. *Computer Graphics with OpenGL*. Pearson international edition. Pearson Prentice Hall, 3 edition, 2004.

[DP02] F. Dunn and I. Parberry. *3D math primer for graphics and game development*. Wordware game math library. Wordware Pub., 2002.

[ea11] Koch R. et al. Bias - basic image algorithms library, 2011.

[FLP01] Olivier Faugeras, Quang-Tuan Luong, and T. Papadopoulou. *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications*. MIT Press, Cambridge, MA, USA, 2001.

[Gmb09] PMDTechnologies GmbH. A performance review of 3d tof vision systems in comparison to stereo vision systems. Technical report, PMDTechnologies GmbH, 2009.

[Gmb11] PMDTechnologies GmbH. A performance review of 3d tof vision systems in comparison to stereo vision systems. Technical report, PMDTechnologies GmbH, 2011.

[GRBB11] Matthias Greuter, Michael Rosenfelder, Michael Blaich, and Oliver Bittel. Object Detection with the 3D-Sensor Kinect. In *International Conference on Research and Education in Robotics*, pages 130–143. Springer, 2011.

[Hor00] Berthold K. P. Horn. Tsai's camera calibration method revisited. Technical report, Massachusetts Institute of Technology, 2000.

[HS97] Janne Heikkila and Olli Silven. A four-step camera calibration procedure with implicit image correction. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 1106–, Washington, DC, USA, 1997. IEEE Computer Society.

[Ike89] *Shape from shading*. MIT Press, 1989.

[Kho11] K. Khoshelham. Accuracy analysis of kinect depth data. Volume XXXVIII-5/W12, 2011:1–6, 2011.

[LB08] Guido Maria Cortelazzo Luca Ballan. Marker-less motion capture of skinned models in a four camera set-up using optical flow and silhouettes. *Proceedings of 3DPVT'08 - the Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, pages 1–8, 2008.

[Li01] Ming Li. Correspondence analysis between the image formation pipelines of graphics and vision. Technical report, May 2001.

[MG01] Thomas B. Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding*, 81:231–268, 2001.

[MHS05] Yukiko Kenmochi Masatoshi Hamanaka and Akihiro Sugimoto. Discrete epipolar geometry. *Lecture Notes in Computer Science*, Volume 3429/2005:323–334, 2005.

[MSB99] Vaclav Hlavac Milan Sonka and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. PWS Publishing, 1999.

[Ope11a] OpenCV, 2011.

[Ope11b] OpenNI. *OpenNI User Guide*. OpenNI, 2011.

[Pro09] Frd Pro. *PMDSDK 2 Programming Manual*. PMDTec GMBH, 2.1.1 edition, 6 2009.

[PYGC08] Pekelny, Yuri, Gotsman, and Craig. Articulated object reconstruction and markerless motion capture from depth video. *Computer Graphics Forum*, 27(2):399–408, 2008.

[Res11] Microsoft Research. Programming guide: Getting started with the kinect for windows sdk beta. Technical report, Microsoft, 2011.

[Sab08] Robert Sablatnig. 3d vision maschinelles sehen in 3d. Skriptum zur Lehrveranstaltung, 2008.

[SBK08] Ingo Schiller, Christian Beder, and Reinhard Koch. Calibration of a pmd camera using a planar calibration object together with a multi-camera setup. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume Vol. XXXVII. Part B3a, pages 297–302, Beijing, China, 2008. XXI. ISPRS Congress.

[SBKK07] Birger Streckel, Bogumil Bartczak, Reinhard Koch, and Andreas Kolb. Supporting structure from motion with a 3d-range-camera. In *Proceedings of the 15th Scandinavian conference on Image analysis*, SCIA'07, pages 233–242, Berlin, Heidelberg, 2007. Springer-Verlag.

[SBSS08] Agnes Swadzba, Niklas Beuter, Joachim Schmidt, and Gerhard Sagerer. Tracking objects in 6d for reconstructing static scenes. *Computer Vision and Pattern Recognition Workshop*, 0:1–7, 2008.

[SC06] Aravind Sundaresan and Rama Chellappa. Multi-camera tracking of articulated human motion using motion and shape cues. In *IN ASIAN CONFERENCE ON COMPUTERVISION*, pages 131–140. Springer, 2006.

[Sch05] Oliver Schreer. *Stereoanalyse und Bildsynthese*. Springer-Verlag Berlin Heidelberg, 2005.

[Sch11] Ingo Schiller. Mip - multicameracalibration., 2011.

[SK10] Alexander Sabov and Jörg Krüger. Identification and correction of flying pixels in range camera data. In *Proceedings of the 24th Spring Conference on Computer Graphics*, SCCG '08, pages 135–142, New York, NY, USA, 2010. ACM.

[ZTCS99] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Shape from shading: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21:690–706, August 1999.