Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Gregor Falk

Verfahren zur Personenverfolgung und -zählung in stark frequentierten Durchgangsbereichen mit Hilfe des Kinect 3D-Bildsensors

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Gregor Falk

Verfahren zur Personenverfolgung und -zählung in stark frequentierten Durchgangsbereichen mit Hilfe des Kinect 3D-Bildsensors

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Studiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Hans Peter Kölzer
Zweitgutachter : Prof. Dr.-Ing. Stephan Hußmann

Abgegeben am 17. Januar 2014

Gregor Falk

Thema der Masterarbeit

Verfahren zur Personenverfolgung und -zählung in stark frequentierten Durchgangsbereichen mit Hilfe des Kinect 3D-Bildsensors

Stichworte

Microsoft Kinect™, 3D-Bildsensor, Kamera Kalibrierung, libfreenect, OpenCV, Bildanalyse, Gesichtsdetektion, Viola und Jones, SURF/SIFT-Algorithmus, Tracking, Partikelfilter, Personenzählung

Kurzzusammenfassung

Diese Arbeit umfasst die Analyse von Verfahren zur Personenverfolgung mit anschließender Personenzählung unter Verwendung des Microsoft Kinect™ 3D-Bildsensors. Schwerpunkt dabei bildet die Gesichtsdetektion sowie das Tracking von Personen mittels Partikelfilter unter Verwendung der gleichzeitig zur Verfügung stehenden Farbbild- und Tiefeninformationen. Es werden die Randbedingungen der involvierten Systeme vorgestellt, analysiert und bewertet. Durch die Erforschung verschiedener Lösungsansätze konnte im Rahmen dieser Arbeit ein, an die Erfordernisse angepasstes, optimales Ergebnis erarbeitet, realisiert und umfangreich getestet werden. Eine Bewertung der Realisierung sowie eine Auswertung der Testergebnisse sind ebenfalls Gegenstand dieser Thesis.

Gregor Falk

Title of the master thesis

Methods for person tracking and counting in busy transit areas using the Kinect 3D image sensor

Keywords

Microsoft Kinect ™, 3D image sensor, camera calibration, libfreenect, OpenCV, image analysis, face detection, Viola and Jones, SURF/SIFT algorithm, tracking, particle filter, person counting

Abstract

Inside this report the analysis of methods for person tracking with following counting of people using the Microsoft Kinect ™ 3D image sensor is described. The emphasis is seen in the face detection and tracking of persons using particle filter and concurrently available color image and depth information. Boundary conditions of the involved systems are presented, analyzed and evaluated. By exploring different solution approaches in this work, which are drawn to the needs, an optimal result could be developed and implemented as well as tested extensively. An evaluation of the implementation and the evaluation of the test results are also subject of this thesis.

Inhaltsverzeichnis

Bilderverzeichnis

Tabellenverzeichnis

1 Einführung

Im Rahmen dieser Masterarbeit an der Hochschule für Angewandte Wissenschaften (HAW) in Hamburg sollen Verfahren für die Personenverfolgung und –zählung mit Hilfe des Microsoft Kinect 3D-Bildsensors untersucht und ein Programm entwickelt werden, welches die als Aufgabenstellung beschriebene Problemstellung löst. Dieses Kapitel dient der Darstellung der Motivation und formuliert, darauf aufbauend, diese Problem- bzw. Aufgabenstellung. Das Kapitel schließt mit Erläuterungen zur Struktur der Arbeit und einigen allgemeinen Hinweisen.

1.1 Motivation

Die Bildverarbeitung hat in den letzten Jahren vielfältige Anwendungsmöglichkeiten im Bereich der Wissenschaft und Technik für sich eingenommen. Jeder Besitzer eines modernen Smartphones verfügt über z.B. eine Vielzahl von bildanalysierenden Elementen, welche softwareseitig in diese integriert sind. Die Gesichtsdetektion und die Möglichkeit, das Mobiltelefon nur mit seinem eigenen Gesicht zu entsperren seien zwei Beispiele hierfür.

Dennoch bleibt der Überwachungsbereich der wohl größte Anwender von Videotechnik. Kameras werden zur Bewachung von Gebäuden oder Räumlichkeiten eingesetzt, aber auch für die Langzeitspeicherung von Daten, die auf diese Weise aufgenommen werden. Seit einiger Zeit werden Überwachungssysteme mittels unterschiedlichster Kameras auch zur Analyse der erfassten Bilddaten direkt eingesetzt. Hier sind vor allem die Kontrolle von Personengruppen über Trackingverfahren sowie die daraus abgeleitete Analyse von Bewegungen oder Bewegungsrichtungen oder gar die Identifikationen einzelner Personen hieraus zu nennen.

Weiterhin werden Kameras aber auch im Sicherheitssektor einsetzt. So besitzen viele Autos moderner Baureihen Kameras als Einparkhilfe oder für Anti-Kollisions-Warnsysteme. Diese detektieren zum Beispiel Personen, die in den Fahrbereich eindringen, und geben, über die eingebetteten und die Situation analysierenden Programme, Warnungen an den Fahrer aus.

Kameras oder Sensoren, die den betrachteten Bildbereich auch in der dritten Dimension räumlich auflösen sind die jüngsten Errungenschaften im Bereich der bildverarbeitenden Systeme. Mit ihren unterschiedlichen, meist auf Stereoanalysen basierenden, Verfahren sind verschiedenste Anwendungen im Bereich der Analyse von Räumlichkeiten oder Strukturen von Objekten denkbar. Häufig werden sie für die Vermessung oder Steuerung von Gegenständen bzw. Automatisierung von Robotersystemen eingesetzt.

Mit der Microsoft Kinect™-Kamera wurde die 3D-Kameratechnik 2010 auch in den häuslichen Bereich für Jedermann erschwinglich eingeführt. Die Xbox 360® Spielekonsole war damit Vorreiter in der Anwendung kamerabasierter Steuerungsmöglichkeiten in der Unterhaltungstechnik, d.h. in der Lenkung von Spielen durch Bewegungen und Gesten.

1.2 Aufgabenstellung

Im Rahmen dieser Masterarbeit ist es nun das Ziel, die in der Motivation in Ansätzen beleuchteten Verfahren der Bildverarbeitung auf den Kinect™-Bildsensor als Hardwarekomponente anzuwenden. Dabei soll ein Programm entwickelt werden, welches Möglichkeiten zur Personenverfolgung und –zählung beinhaltet. Ziel ist es, Personen, die einen Durchgang entlang gehen, zu Detektieren, zu Tracken (d.h. zu Verfolgen) und schließlich zu Zählen. Dabei gilt es insbesondere die durch die Kinect™ zur Verfügung stehenden, zusätzlichen 3D-Daten zu nutzen.

Eingeschränkt wird diese grundlegende Beschreibung der Problemstellung zunächst auf einen Personenkreis von maximal zwei Personen, welche die gleichzeitig für die Erfüllung der Aufgabe betrachteten Objekte darstellen. Diese Definition beschreibt dennoch einen stark frequentierten Durchgangsbereich, denn die Frequenz der erscheinenden Personen ist dabei nicht eingegrenzt. Eine Analyse für den Umgang mit größeren Personenmengen soll aber auch Gegenstand dieser Thesis sein.

Als erste Ziele seien jedoch zunächst die Abwägung der Machbarkeit und die Entwicklung eines Konzeptes aufgrund gängiger Verfahren der Bildverarbeitung und Bildanalyse angesetzt. Auch der Arbeitsbereich und die herrschenden Umgebungsvariablen sollen festgelegt werden. Abschließend gilt es, das entwickelte Gesamtkonzept als Anwendungsprogramm zu realisieren und weitere Untersuchungen durchzuführen, welche diese Thesis abrunden. Aussagen über die reale Einsetzbarkeit und Eindeutigkeit der Personenzählung stellen das, mit den Erkenntnissen dieser Arbeit gewonnenen Erfahrungen, bedeutendste Endziel dar.

1.3 Aufbau der Arbeit

Diese Thesis ist in verschiedene Kapitel unterteilt. Nach dieser kurzen Einführung werden in Kapitel 2 zunächst der Ausgangszustand, d.h. der Kinect-Sensor als solcher, beschrieben und Grundlagen zum Thema Kinect, Kamera-Kalibrierung, Linux und OpenCV vorgestellt, welche für den Verlauf dieser Arbeit von Bedeutung sind.

Kapitel 3 beinhaltet verschiedene Analysen der Nutzungsmöglichkeiten und Randbedingungen sowie zur Merkmalsextraktion und Klassifikation. Auch eine Analyse von Objektverfolgung mittels Partikelfiltern ist enthalten. Am Ende wird ein Konzept vorgestellt, welches anschließend in Kapitel 4 umgesetzt wird.

Nachdem die mögliche Realisierung als Programm in Kapitel 4 beschrieben, abgeschlossen und abgewogen wurde, folgt in Kapitel 5 die Evaluation dieser.

Zum Abschluss dieser Masterarbeit werden noch einmal die wesentlichen Ergebnisse zusammengefasst und bewertet. Außerdem folgt ein Ausblick auf mögliche zukünftige Schritte und weitere Anwendungsmöglichkeiten.

1.4 Konventionen

Für das bessere Verständnis dieser Masterarbeit ist es sinnvoll und notwendig einige Konventionen zu vereinbaren. Diese sind nachfolgend aufgeführt und für den gesamten Umfang dieser Thesis gültig.

Aus Gründen der besseren Lesbarkeit werden im Verlauf der Arbeit Marken- und Urheberrechtliche Bezeichnungen und Zusätze ausschließlich bei der ersten Verwendung aufgeführt. Außerdem wird der Kinect-Bildsensor zur Vermeidung von Eintönigkeit unterschiedlich bezeichnet – Kinect-Sensor, Kinect-Kamera, aber auch nur Kinect, Kamera oder Kamerasystem sind die häufig verwendeten Begrifflichkeiten.

Des Weiteren sind häufig verwendete Abkürzungen, Fremdworte und Begriffe im Glossar zusammengestellt.

Außerdem werden Quellenangaben für einzelne Bilder, welche nicht den üblicherweise Verwendeten Quellen des Literaturverzeichnisses entstammen, als Fußnote direkt am Objekt kenntlich gemacht. Beziehen sich hingegen ganze Abschnitte sinngemäß auf eine Quelle, so wird dieses zu Beginn der jeweiligen Betrachtungen nur einmal bekannt gegeben. Ausführungen, welche nur im weitesten Sinne einer Quelle entstammen, werden hingegen nicht explizit benannt, sondern die Quellen lediglich als allgemeine Quellen dieser Thesis im Literaturverzeichnis aufgeführt.

Zu guter Letzt sei noch darauf hingewiesen, dass die Nummerierung der Bilder und Tabellen zwar fortlaufend erfolgt, jedoch die jeweilige Kapitelnummer führend mit angegeben wird. Bild 4.9 wäre also das neunte Bild dieser Thesis und befindet sich im Kapitel 4.

2 Grundlagen

Dieses Kapitel stellt die grundlegenden Randbedingungen dieser Arbeit vor, welche für die Konzeptionierung sowie die schlussendliche Realisierung der Aufgabenstellung benötigt werden. Hierzu gehören die Vorstellung des eigentlichen Kinect-Bildsensors sowie die verwendeten Betriebssysteme und Softwares. Weiterhin seien an dieser Stelle die Grundlagen der Kalibrierung von (3D-)Kameras und für diese Arbeit wichtige Verfahren der Bildverarbeitung dargestellt.

2.1 Kinect-Sensor

Der Kinect™-Sensor der Firma PrimeSense™ wurde für die Microsoft Xbox 360® entwickelt und 2009 unter dem Namen „Projekt Natal“ vorgestellt. Die Markteinführung des, auf dem PrimeSensor™ basierenden, Sensors folgte 2010. PrimeSense ist einer der führenden Hersteller für 3D-Mess- und Erkennungstechnologien mit Sitz in Israel. Ziel war es, eine neuartige Schnittstelle für den Xbox-Nutzer zu schaffen, welche es erlaubt auf einfache und intuitive Art und Weise die Konsole bzw. deren Spiele zu kontrollieren und natürlich mit diesen Geräten zu interagieren. Dabei sollte der Sensor gleichzeitig für Jedermann erschwinglich bleiben.

Bild 2.1: Xbox 360® Kinect™-Sensor[1]

Das Ergebnis ist der in Bild 2.1 dargestellte und schließlich als Kinect™ benannte Sensor, welcher heute für ca. 100€ zu bekommen ist. Er stellt bis heute einen Meilenstein in der kontrollerlosen Spielesteuerung dar.

Die Kamera erfasst dabei zurzeit maximal zwei Spieler als aktive Interaktionspartner, welche über ihre Bewegungen bzw. Gesten und Sprache Spiele steuern können.

Seit 2011 stellt Microsoft Research eine Windows-SDK für Windows 7 zur Verfügung, welche es Entwicklern erlaubt nicht-kommerzielle Anwendungen außerhalb der Xbox zu entwickeln. Parallel hat die Firma ASUS mit dem WAVI Xtion einen eigenen 3D-Sensor für den Personal Computer vorgestellt, welcher als alternative zum Kinect-Sensor gilt.

[1] Quelle: Wikipedia: *Die freie Enzyklopädie*, `http://en.wikipedia.org/wiki/File:Xbox-360-Kinect-Standalone.png` (abgerufen am 02.01.2014)

2.1.1 Komponenten des Kinect-Sensors

Der Kinect-Sensor besteht aus einem neigbaren Kopf, welcher die Mikrofone und Kameras enthält, sowie einem Standfuß. Nachfolgend seien die einzelnen Komponenten im Bild 2.2 dargestellt und spezifiziert.

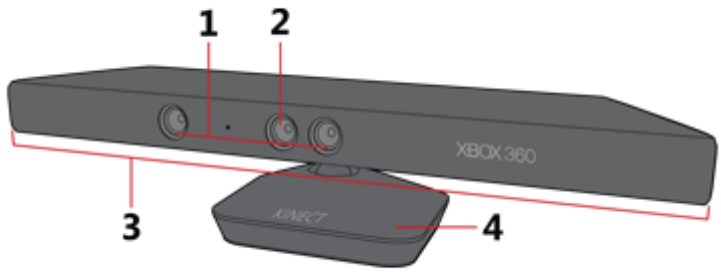

Bild 2.2: Wichtige Komponenten des Kinect™-Sensors, Quelle: [8]

(1) 3D-Tiefensensor: Ein Infrarot-Laser-Projektor (links) und ein monochromer CMOS-Sensor (rechts) arbeiten zusammen, um unabhängig von den Lichtverhältnissen ein dreidimensionales Sehen zu ermöglichen. Der Sensor bietet eine Ortsauflösung von 640 x 480 Pixel bei einer Tiefenauflösung von bis zu 11 Bit (2048 Quantisierungsstufen) und einer Wiederholrate von 30 Hz. Die genaue Funktionsweise ist in Abschnitt 2.1.2 vorgestellt.

(2) RGB-Farbkamera: Der RGB-Bayer-Sensor[2] erfasst ein 640 x 480 Pixel großes, 8 Bit Farbbild, ebenfalls bei einer Wiederholrate von 30 Hz.

(3) Mikrofon-Array: Der integrierte Verbund von vier Mikrofonen zu einem Array ermöglicht die Erfassung und Ortung von Sprache bzw. Sprachbefehlen eines einzelnen Spielers.

(4) Motorisierte Neigung: Im Standfuß ist ein mechanischer Antrieb enthalten, welcher eine vertikale Neigung des Kinect-Sensors um ±30° ermöglicht. Diese Neigung kann wiederum von 3G-Beschleunigungssensoren im Kopf erfasst und ausgelesen werden.

Status-LED: Im Kopf befindet sich zwischen (1) und (2) weiterhin eine Status-LED, welche in grün und rot, aber auch gelb den aktuellen Arbeitszustand anzeigt.

An das Endgerät wird der Sensor über ein spezielles USB-Kabel mit externer Stromversorgung angeschlossen. Es wird das USB 2.0 Protokoll verwendet.

[2] RGB-Bayer-Sensor: Fotosensor mit einem Farbfilter, welcher zu 50% aus grünen und je 25% aus roten und blauen Pixeln besteht.

2.1.2 Der 3D-Tiefen-Sensor

Es gibt heutzutage verschiedenste Möglichkeiten Tiefeninformationen mit Hilfe von Kameras zu erfassen. Der dabei gängigste, weil auch kostengünstigste Weg stellt dabei die Verwendung eines **Stereo-Kamerasystems** dar. Hierbei werden lediglich zwei gut auflösende Kameras in einem definierten Abstand parallel oder in einem leichten Winkel zueinander platziert und aus den beiden aufgenommenen Bildern die Pixelverschiebung anhand von Referenzpunkten bestimmt. Mit Hilfe einfacher triangulierender Berechnungen wird dann die Entfernung vom Kamerasystem bestimmt.
Eine weitere Möglichkeit stellen die so genannten **TOF[3]-Kameras** dar. Diese messen über modulierte Lichtimpulse im Infrarot-Bereich die Signallaufzeit, welche auf dem Weg vom Sender über das angestrahlte Objekt zurück zum Empfänger in der Kamera entsteht. Über diese Laufzeitmessung kann schließlich direkt auf die Distanz geschlossen werden.

PrimeSense hat bei dem Kinect-Sensor jedoch auf ein neues Prinzip gesetzt, welches in dieser Form von PrimeSense patentiert wurde. Grundsätzlich basiert es auf der Methode des „**structed light**“. Hierbei wird mit Hilfe eines Infrarotprojektors ein pseudozufälliges Interferenzpunktemuster erzeugt und in den Raum ausgestrahlt. Mit Hilfe einer Infrarot-Kamera wird dieses Muster dann wieder aufgenommen und daraus die Tiefeninformationen ermittelt. In Bild 2.3 sind noch mal die Hardware und Position von Projektor und Sensor in dem unverkleideten Kinect-Sensor abgebildet. Die genaue Funktionsweise der Tiefenmessung sei in den nachfolgenden Absätzen erklärt.

Bild 2.3: Komponenten des unverkleideten Kinect-Sensors, Quelle [10]

Wie bereits erwähnt erzeugt der Infrarot-Projektor ein permanentes, bekanntes Punktemuster, welches in den betrachteten Raum ausgestrahlt wird, und wo es als Vielzahl an Reflexionspunkten erscheint. Hierfür befinden sich vor einem Infrarot-Laser zwei optische Diffraktionsgitter, welche ein zunächst regelmäßiges Punkteraster erzeugen. Jedoch werden ferner nur die hellsten und damit für den Infrarot-Sensor später am besten wieder zu erkennenden Punkte durch gelassen. Auf diese Weise entsteht eines von neun Untermustern, welche untereinander alle identisch sind und in der Mittel jeweils einen noch helleren Referenzpunkt zur Lokalisation von Ausschnitten aus dem Gesamtmuster enthält. Ein ungestörtes Interferenzmuster sowie

[3] TOF: englisch: time of flight

dessen Erzeugung ist in Bild 2.4 dargestellt, wobei die Referenzmarken sowie die neun Untermuster in rot hervorgehoben sind.

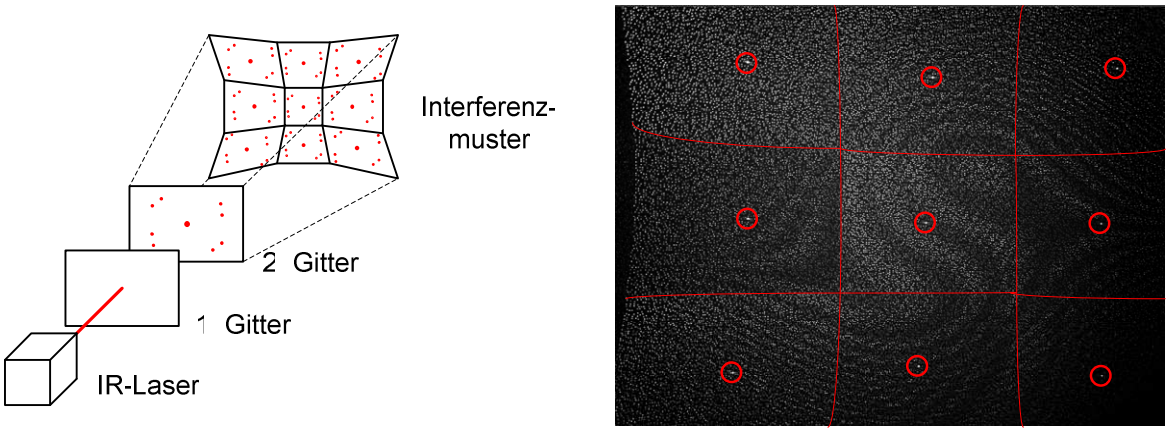

Bild 2.4: Ungestörtes pseudozufälliges Interferenzpunktmuster des Infrarot-Projektors

Befindet sich nun ein Objekt im Raum oder bewegt sich durch diesen, so werden die Lichtpunkte anders reflektiert und das bekannte Muster verzerrt. Diese Veränderungen im Muster werden von dem Infrarot-Sensor aufgenommen und ein nach geschaltetes Chip-Set errechnet hieraus 3D-Informationen. Diese Informationen werden bei dem Kinect-Sensor als 11-Bit Grauwertbild aufbereitet und mit einer Auflösung von 640 x 480 Pixel zur Verfügung gestellt.

2.1.2.1 Auflösung und Sichtfeld

Das Sichtfeld der Infrarot-Kamera beträgt horizontal 57° und vertikal 43° (siehe [11]). Die für die 3D-Berechnung notwendige Stereobasis beträgt dabei 7,5 cm. Da der Kinect-Sensor jedoch nur die notwendige eine Infrarot-Kamera besitzt entsteht ein unangenehmer Nebeneffekt bei der Erfassung und Berechnung der Referenzpunkt. Wie in Bild 2.5 schematisch dargestellt entsteht hinter einem angestrahlten Objekt ein Infrarot-Schatten in dem abgestrahlten Muster.

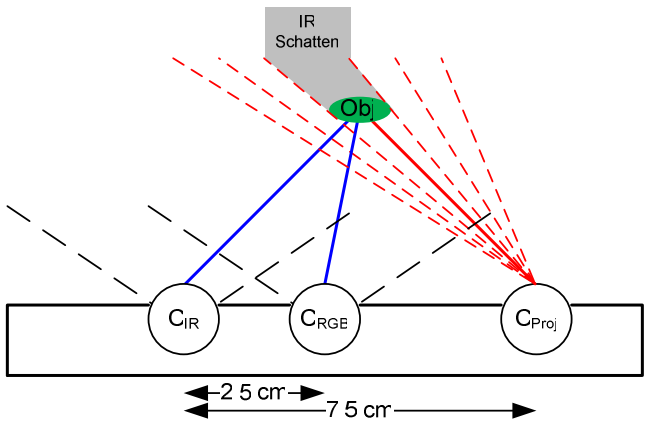

Bild 2.5: Schematische Darstellung des Kinect-Sensors

Durch den Schatten befinden sich hinter dem Objekt keine Interferenzpunkte sodass hier keine Aussagen über Tiefeninformationen getroffen werden können. In dem späteren Grauwertbild sind diese Bereiche dann nicht definiert bzw. werden mit dem maximalen Quantisierungswert belegt.

Weiterhin lässt sich unter anderem aus Bild 2.5 schließen, dass sich die Anzahl an Infrarot-Punkten auf einem Objekt mit der Entfernung zum Projektor ändert. Ist ein Objekt nah, so fallen viele Interferenzpunkte auf dieses und die Oberfläche bzw. Struktur kann detaillierter erfasst werden. Mit steigender Entfernung ändert sind die Dichte und Anzahl an Punkten und damit gleichzeitig die Auflösung in der Tiefe geringer (vgl. Bild 2.6).

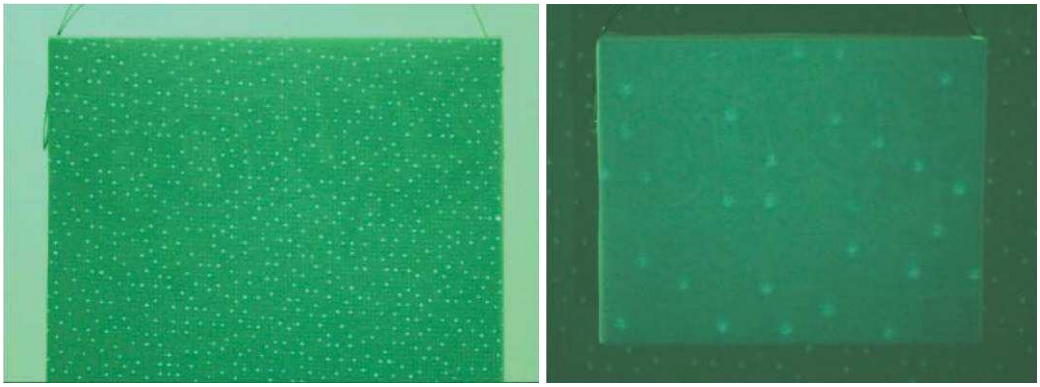

Bild 2.6: Interferenzmuster auf Objekt in 1m (links) und 6m (rechts), Quelle: [12]

Aus [12, Tab.2] ergeben sich folgende Werte für die Tiefen- und Querauflösung:

Distanz [m]		**0,5**	**1,0**	**1,5**	**2,0**	**2,5**	**3,0**	**3,5**	**4,0**	**5,0**
Auflösung	**Tiefe [mm]**	1,3	4,0	8,0	13,2	19,0	27,0	38,8	50,8	75,0
	Quer [mm]	1,3	2,2	3,3	4,1	4,8	6,1	7,0	7,9	9,7

Tabelle 2.1: Tiefen- und Querauflösung des Kinect-Sensors

Neben der sinkenden Auflösung mit der Entfernung gibt es auch eine minimale Entfernung für die Berechnung von Tiefeninformationen mit Hilfe des Kinect-Sensors. Diese liegt bei ca. 45 cm. Unterhalb dieser Distanz können ebenfalls keine Tiefeninformationen ermittelt werden und im Grauwertbild bilden sich erneut weiße, undefinierte Bereiche aus (vgl. Infrarot-Schatten).

2.1.2.2 Entfernungsbestimmung und Arbeitsbereich

Um später aus dem Grauwert-Tiefenbild (8 Bit oder 11 Bit) ggf. explizite Entfernungen bestimmen zu können, lässt sich nachfolgende Gleichung [aus 13] approximieren:

$$f(q) = 123{,}6 \cdot \tan\left(\frac{q_{11Bit}}{2842{,}5} + 1{,}1863\right) \tag{1}$$

Wird nun für *q* ein entsprechender Pixelwert eingesetzt ergibt sich die Entfernung in der Tiefe in mm. Die Grafik in Bild 2.7 veranschaulicht noch einmal die Beziehung von qunatisiertem Pixelwert und der Entfernung als Umkehrfunktion von *f(q)*.

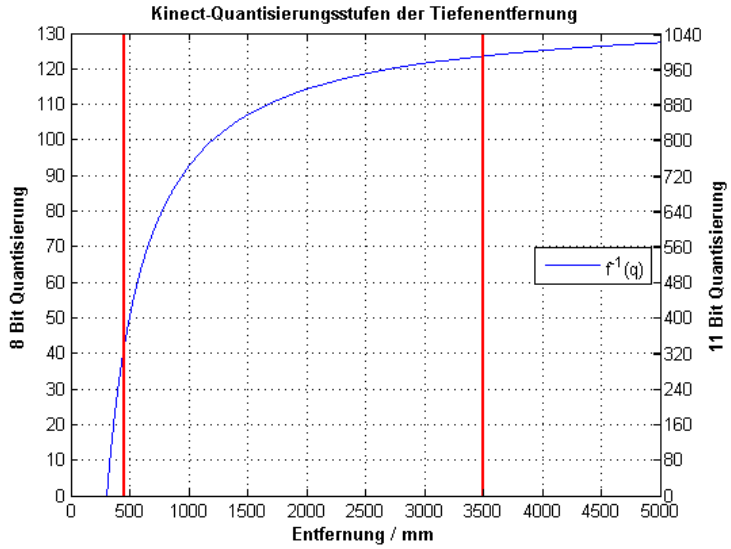

Bild 2.7: Beziehung von Entfernung und Pixelwert im Tiefenbild (8 und 11 Bit)

Eingezeichnet ist außerdem die minimale Grenze für den Arbeitsbereich bei 450 mm. Gemäß der Spezifikation des Kinect-Sensors [11 ; 12] liefert der Kinect-Sensor nur bis 3,5 m akzeptable Tiefeninformationen. Diese Grenze ist ebenfalls in Bild 2.7 eingezeichnet und bestimmt zusammen mit der minimalen Grenze einen sinnvollen **Arbeitsbereich von 450 – 3500 mm**. Die maximale Entfernung für diesen Bereich lässt sich ebenfalls aus dem Verlauf von $f^{-1}(q)$ erschließen, da ab einem Pixelwert von ca. 120 bei 8 Bit Quantisierung die Unterscheidbarkeit von Entfernungen zunehmend sinkt. Damit ergeben sich für die spätere Verwendung von Tiefeninformationen ein maximaler Pixelwert von 122 und ein minimaler Wert von 43 bei 8 Bit.

2.2 Kalibrierung von RGB- und Tiefenbild

Wie bereits in Abschnitt 2.1 vorgestellt, erfasst der Kinect-Sensor zwei verschiedene Bilder mit Hilfe zweier unterschiedlicher Bildsensoren (Kameras). Dabei handelt es sich um einen RGB-Farbsensor sowie einen Infrarot-Tiefensensor, welcher ein Grauwertbild liefert. Beide Bilder haben schlussendlich eine Auflösung von 8 Bit.

Weiterhin ist bereits aus den Bildern Bild 2.3 und Bild 2.5 ersichtlich, dass diese beiden Kameras eine örtliche Verschiebung von 2,5 cm zueinander innehaben – logisch, denn die Sensoren können zur gleichen Zeit nicht denselben Platz im Raum einnehmen. Mit dem Kinect-Sensor liegt also ein quasi Stereo-Kamerasystem vor, wobei die Kameras nicht zugleich für die triangulierende Tiefenberechnung benötigt werden.

Doch überlagert man die beiden Bilder einer ausgewählten Szene (Bild 2.8 oben) ohne weitere Bearbeitung ergibt sich nachfolgendes Bild:

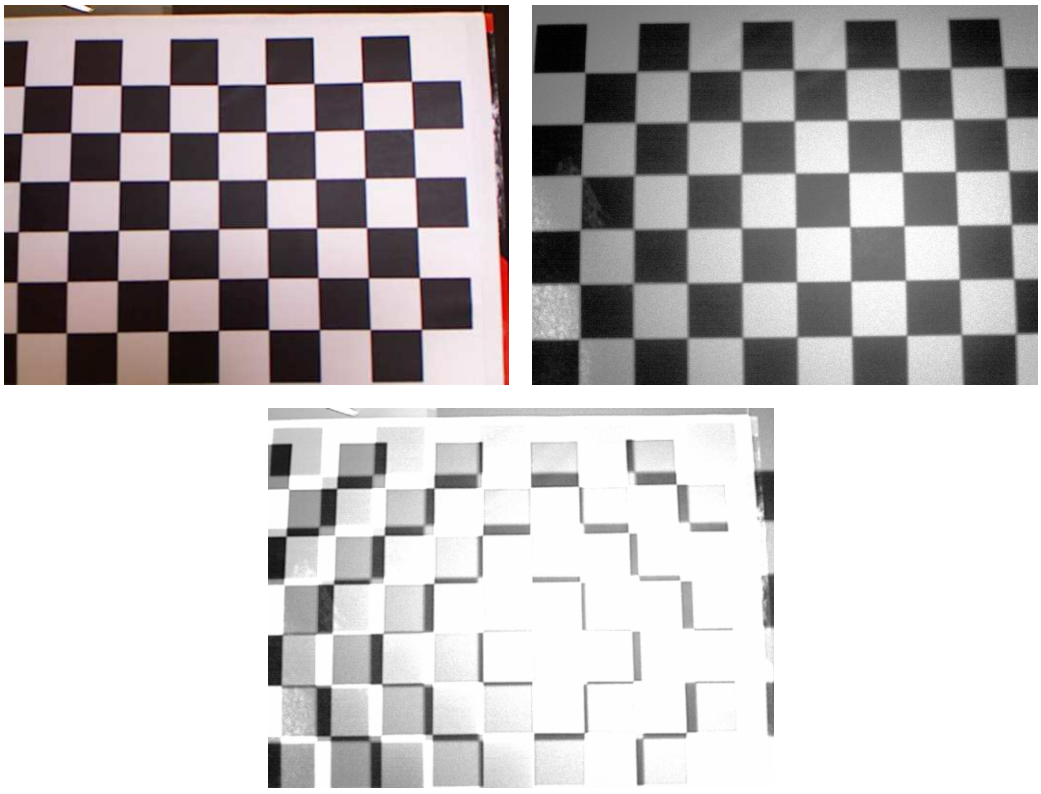

Bild 2.8: Unkalibrierte Aufnahmen: RGB (o.links), IR (o.rechts), überlagert (unten)

Das untere Bild der Bildergruppe zeigt deutlich, dass die Überlagerung der beiden Einzelbilder eine Verschiebung von größtenteils einem Schachbrettrechteck (entspricht ca. 2,5 cm) aufweist. Außerdem ist eine Linsen bzw. Fokusweiten bedingte Verzerrung zu vermuten.
Die Grundlagen für die Kalibrierung und damit Beseitigung dieser Fehler sind in Anlehnung an [3, S.370ff ; 11] in den nachfolgenden Abschnitten dargestellt.

2.2.1 Intrinsische Parameter (Kamera-Modell, Verzerrung)

Zunächst sei das Kamera-Modell mit der Kamera-Matrix *M* für die intrinsischen Parameter definiert. Jede Kamera besitzt spezifische Eigenschaften, die sie definieren. Hierzu gehören die Brennweite *f* (focal length) sowie der Bildmittelpunkt *c* (principal point). Diese Einflussgrößen ändern sich nicht sofern sich die Kamera(-Eigenschaften) nicht ändern, z.B. durch einen Zoom, und sind somit von der aufgenommenen Szene unabhängig.

In der Matrix *M* angeordnet ergibt sich für die, jeweils in x als auch in y Richtung des Bildes wirkenden, Parameter folgende Definition:

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Mit Hilfe dieser Matrix können die Koordinaten der physikalischen Welt in homogene Kamerakoordinaten projiziert werden. Es ergibt sich folgende Gleichung, wobei *Q* die realen und *q* die homogenen Koordinaten repräsentieren:

$$q = M \cdot Q \text{ , mit } q = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \text{ und } Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3}$$

Nachdem nun eine gemeinsame Grundlage mit einem spezifischen Koordinatensystem geschaffen wurde, folgt der Blick auf das erste Problem, welches im vorangegangenen Abschnitt benannt wurde – die Linsenverzerrung.

Wie in Bild 2.9 visualisiert, erzeugt die Linse einer Kamera eine Verzerrung von Objekten, sodass diese auf der Bildebene deformiert erscheinen. Auch diese Eigenschaft ist für jede Kamera bzw. Linse fest und lässt sich über weitere intrinsische Parameter beschreiben.

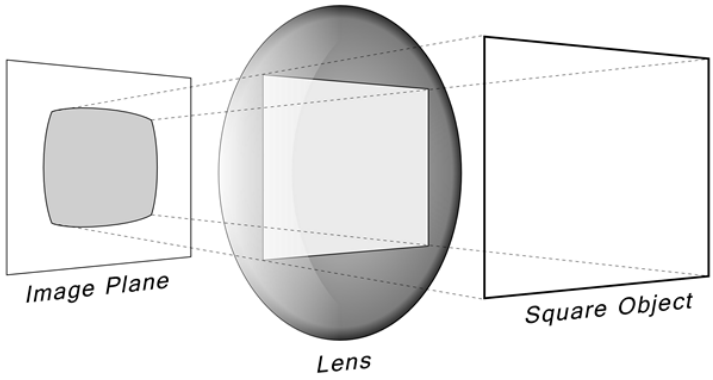

Bild 2.9: Prinzip der Linsenverzerrung, Quelle: [3]

Bei den Linsenverzerrungen werden radiale und tangentiale Verzerrungen unterschieden, wobei erstere auf den Linsenschliff zurück zu führen sind, zweitere auf die Montage der Kamera als Ganzes.

Die häufigsten radialen Verzerrungen sind die Tonnen- oder Kissenverzerrung, welche durch zu oder abnehmende Vergrößerungen zum Rand der Linse hin entstehen. Die tangentialen Verzerrungen entstehen, wenn Linse und Bildebene nicht parallel zueinander angeordnet sind. Alles in allem ergeben sich bis zu fünf Verzerrungskoeffizienten, welche zumeist in einem Verzerrungs-Vektor *d* angegeben werden:

$$d = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{4}$$

Die *k*-Koeffizienten repräsentieren dabei die radiale, die zusätzlichen *p*-Koeffizienten die tangentiale Verzerrung.

Um die beschriebenen Verzerrungen zu beseitigen sind insgesamt vier Gleichungen notwendig, welche die originalen Pixel (x und y) an die korrekte Position verschieben. Die obere Gleichung errechnet die Umskalierung der radialen Verzerrung, die untere Gleichung die der tangentialen Verzerrung.

$$\begin{aligned} x_{korrigiert} &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ y_{korrigiert} &= y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \end{aligned} \tag{5}$$

$$\begin{aligned} x_{korrigiert} &= x + \left(2 p_1 y + p_2\left(r^2 + 2x^2\right)\right) \\ y_{korrigiert} &= y + \left(p_1\left(r^2 + 2y^2\right) + 2 p_2 x\right) \end{aligned} \tag{6}$$

Mit Hilfe der später in Abschnitt 4.2 beschriebenen Kalibrierung lassen sich die vollständigen intrinsischen Parameter (Matrix *M* und Verzerrungskoeffizienten) für beide Kameras des Kinect-Sensors ermitteln.

2.2.2 Extrinsische Parameter (Rotation, Translation)

Um das zweite und durchaus gravierendere Problem der gegeneinander verschobenen Bilder aufgrund der Basisdistanz von 2,5 cm zu beseitigen, werden die so genanten extrinsischen Parameter benötigt. Diese stellen sich als Rotations-Matrix *R* und Translations-Vektor *t* dar, sind jedoch nur für die jeweils aktuelle räumliche Situation gültig. Erst zwei Parameter machen die relative Transformation allgemein gültig.

Vorausgesetzt, zwei Bildebenen verwenden das gleiche homogene Koordinatensystem als Grundlage, so kann nun mit Hilfe der Rotation und Translation (Drehung und Verschiebung zwischen den Ursprüngen zweier Koordinatensysteme) die Konvertierung von Objekt-Koordinatensystem auf das homogene Kamerakoordinatensystem abgeschlossen werden. In Anlehnung an Bild 2.10 gilt:

$$P_C = R \cdot \left(P_O + t \right) \tag{7}$$

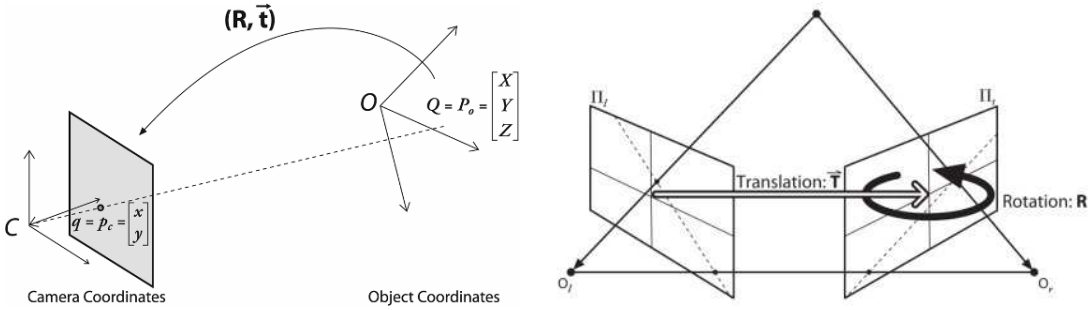

Bild 2.10: Rotation und Translation, Quelle: [3]

Bezogen auf den Kinect-Sensor ergeben sich somit die nachfolgenden Gleichungen für die prinzipielle Transformation von Infrarotbild (IR) in das Farbbild (RGB).

Zunächst werden die Bildpunkte des Infrarotbildes mit Hilfe der invertierten Kamera-Matrix M_{IR} in einen 3D-Punkt überführt. Anschließend erfolgt die Transformation auf das Koordinatensystem des Farbbildes mittels Rotation und Translation mit darauf folgender zurück Projektion mittels der RGB-Kamera-Matrix in das neue Kamera-Koordinatensystem.

$$\begin{aligned} &1.) \quad P_{IR} = M_{IR}^{-1} \cdot p_{IR} \\ &2.) \quad P_{RGB} = R \cdot P_{IR} + t \\ &3.) \quad p_{RGB} = M_{RGB} \cdot P_{RGB} \end{aligned} \tag{8}$$

Auf diese Weise befindet sich jeder existierende Pixel des Infrarotbildes an derselben Bildposition wie der analoge Pixel des Farbbildes. Somit können die drei Farb-Layer des RGB-Bildes mit dem einen Layer des Tiefenbildes z.B. zu einem RGBD-Bild mit vier Layern kombiniert und als kompaktes 3D-Bild weiter verarbeitet werden.

2.2.3 Kalibrierung (mittels Homography)

Mit der Verwendung von homogenen Koordinaten lassen sich hauptsächlich die in Abschnitt 2.2.2 getätigten Überlegungen und Formeln weiter vereinfachen. Hierzu wird die so genannte „homography transformation“ betrachtet, welche sich auf eine 3x3-Matrix H reduzieren lässt. Diese Transformation besteht aus zwei Teilen, der physikalischen Transformation, welche die Bildebenen lokalisiert und ineinander überführt, sowie der Projektion, welche auf die intrinsischen Parameter zurückgeht.

Einfach ausgedrückt lässt sich in Bezug auf Bild 2.11 folgende Gleichung aufstellen:

$$q = HQ \text{ , mit } H = MW \text{ und } W = \begin{bmatrix} R & t \end{bmatrix} \tag{9}$$

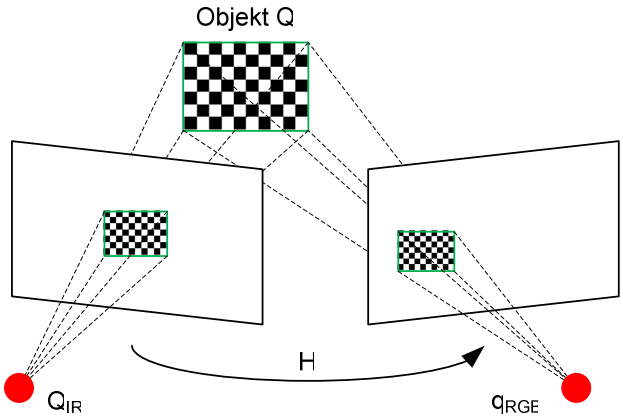

Bild 2.11: Prinzip der „homography transformation“

M entspricht der Kamera-Matrix, *R* der Rotations-Matrix und *t* dem Translations-Vektor. *W* besteht aus der 3x3 Rotations-Matrix mit angefügtem Translations-Vektor, sodass diese Matrix eine Größe von 3x4 hat.

Matrix *W* lässt sich jedoch noch mal vereinfachen, da als Objektebene die Bildebene angenommen werden kann. Somit ergibt sich in *Q* für $Z = 0$, sodass die 3. Spalte der Rotations-Matrix vernachlässigt werden kann. *R* besteht somit nur noch aus zwei Spaltenvektoren r_1 und r_2 (jeweils 3x1), sodass für *W* als 3x3-Matrix nun gilt:

$$W = \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \text{ mit } q = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ und } Q = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{10}$$

Mit Hilfe der intrinsischen Parameter für die Infrarot- sowie die Farbkamera d_{IR}, M_{IR} bzw. d_{RGB}, M_{RGB} und die Homogenitäts-Matrix *H* lässt sich somit die Kalibrierung der beiden Bildsensoren durchführen. Die Bilder werden dabei entzerrt und überlagernd ausgerichtet, sodass die Grundlage für die dreidimensionale Arbeit mit dem Kinect-Sensor geschaffen ist.

2.3 Kinect-Treiber und Entwicklungsumgebung

Zu Beginn einer jeden Programmierarbeit stellt sich die Frage nach dem zu verwendenden Betriebssystem. Im Falle dieser Arbeit ergibt sich ferner die Auswahl eines geeigneten Treibers zur Ansteuerung des Kinect-Sensors. Die beiden folgenden Abschnitte sollen die Entscheidung für den openkinect-Treiber „(lib)freenect“ auf dem Linux basierten Betriebssystem Ubuntu darlegen und begründen. Außerdem werden die wesentlichen Eigenschaften, weiterhin verwendeten Anwendungen sowie die Entwicklungsumgebung als Ganzes vorgestellt.

2.3.1 Kinect-Treiber

Prinzipiell ist der Kinect-Sensor ausschließlich für den Betrieb an einer Xbox gedacht. Aus diesem Grund gab es zu Beginn keinen offiziellen Gerätetreiber von Microsoft, welcher z.B. eine Verwendung unter Windows ermöglicht. Deshalb hat Adafruit, eine Firma für „open source“-Hardwareprojekte, mit der Veröffentlichung der Microsoft Kinect im November 2010 einen dotierten Wettbewerb ins Leben gerufen, um einen quelloffenen Treiber zu entwickeln, welcher den Zugriff auf den Kinect-Sensor ermöglicht. Binnen einem Monat hat die „OpenKinect“-Gemeinschaft [13] erste Erfolge erzielt und somit erste Treiber unter, bis heute, völliger OpenSource-Lizenz zur Verfügung gestellt.

Auf die große Nachfrage aufbauend, brachte 2010 auch PrimeSense mit „OpenNI“ eine eigene Software zur Steuerung des Kinect-Sensors heraus, welche über die grundlegenden Möglichkeiten der Informationsverwaltung hinaus unter anderem auch die Skelettierung und damit akkurate Positions- und Gestenbestimmung von Personen beinhaltet. Aus diesem Grund ist OpenNI zurzeit der bevorzugte Kinect-Treiber, jedoch ist die Verwendung aufgrund der undurchsichtigen Lizenzrechte des „OpenNI NITE“ Tracking-Features differenziert zu betrachten.

Schließlich brachte 2012 auch Microsoft ein SDK[4] für die Kinect heraus, welches einen Einsatz des Kinect-Sensors ab Windows 7 möglich macht.

Im Rahmen dieser Thesis ist es sinnvoll vollständig auf OpenSource-lizenzierte Treiber, Programme und Betriebssysteme zurück zu greifen. Aus diesem Grund fiel die Wahl schnell auf den Kinect-Treiber „**libfreenect**“ der OpenKinect-Gemeinschaft. Dieser ist z.Z. in der Version 0.2 für Windows, Linux und OS X verfügbar und unter „Apache v2/GPL v2“ lizenziert. Er beinhaltet unter anderem Wrapper-Klassen für **C++**, weshalb diese als Programmiersprache gewählt wurde.

Die Entscheidung für das Betriebssystem fiel aufgrund der guten Kompatibilität mit dem Kinect-Treiber auf das Linux basierte Betriebssystem **Ubuntu**.

[4] SDK: Software Development Kit

2.3.2 Betriebssystem und Programmierung

Wie im vorangegangenen Abschnitt bereits erwähnt fällt die Wahl für das Betriebssystem im Rahmen dieser Arbeit auf **Ubuntu**, welches zurzeit als **Langzeitversion 12.04 LTS**[5] verfügbar ist. Betrieben wird dieses auf einem Pentium DualCore E5200 @ 2x2,5GHz Computer mit 8GB RAM der HAW Hamburg. Die teilweise umfangreichen Vorgehensweisen bei der Installation von Programmen und Treibern unter Linux sind auch hier nicht zu unterschätzen. Eine Zusammenfassung der wichtigsten Schritte zur Installation von libfreenect, aber auch den nachfolgend beschriebenen Komponenten ist im Anhang A zusammengestellt. Somit wird an dieser Stelle der besseren Übersicht halber auf explizite Installationsbefehle verzichtet.

2.3.2.1 Programmierumgebung Code::Blocks

Als Programmierumgebung für die objektorientierte Hochsprache C++ fiel die Entscheidung auf **Code::Blocks** als Komponente von openFrameworks[6]. Es liegt für diese Arbeit in der Version 10.05 vor und verwendet den GCC-Compiler 4.6.3.

Die Einbindung der jeweils benötigten Bibliotheken für den Compiler und Linker erfolgt über das interne Linuxprogramm „pkg_config“, sodass in Code::Blocks lediglich folgende Angaben in den Projekteintellungen (build options) gemacht werden müssen:

Compiler settings (unter „other options“):

```
'pkg-config opencv --cflags'
'pkg-config libfreenect --cflags'
```

Linker settings (unter „other options“):

```
'pkg-config opencv --libs
'pkg-config libfreenect --libs'
```

Weiterhin ist die Optimierungs-Option ausgeschaltet. Mit der so konfigurierten Entwicklungsumgebung wird die unter Kapitel 4.1 umgesetzte Realisierung der Aufgabenstellung nunmehr möglich.

2.3.2.2 Erste Schritte

Erste Erfahrungen und Erfolge mit dem Kinect-Sensor können jedoch auch mit den Demo-Anwendungen, welche mit dem freenect-Treiber mitgeliefert werden, gemacht werden. Hier ist vor allem „freenect-glview“ zu nennen, mit dessen Hilfe die korrekte Funktion des Kinect-Sensors über das Terminal verifiziert werden kann. Außerdem können hiermit alle drei Bilder (Farb-, Infrarot- und Tiefenbild), welche die Kinect zur Verfügung stellt, visuell geprüft werden.

[5] LTS: long time service
[6] OpenFrameworks ist ein auf Creative Coding ausgelegtes Open-Source-Toolkit in C++

Bild 2.12: freenect-glview – Farb-, Infrarot- und Tiefenbild

Weiterhin ist es möglich, den Kinect-Sensor über den gspca-Treiber unter Linux als Web-Cam einzubinden, was nach dem Anstecken der Kamera an einem USB-Port ggf. automatisch geschieht. Der aktuelle Zustand kann gemäß der Angaben im Anhang A.4 überprüft und verändert werden.

Ist die Kinect angeschlossen, so können über `lsusb` im Terminal die drei USB-Devices für den ansteuerbaren Motor, das Mikrofonarray und natürlich die Kameras geprüft werden. Es sollte sich eine, der nachfolgenden Ausgabe ähnliche Auflistung ergeben:

```
$ lsusb
...
Bus 002 Device 007: ID 045e:02b0 Microsoft Corp. Xbox NUI Motor
Bus 002 Device 008: ID 045e:02ad Microsoft Corp. Xbox NUI Audio
Bus 002 Device 009: ID 045e:02ae Microsoft Corp. Xbox NUI Camera
```

Über die nachfolgenden beiden Befehlen kann mit Hilfe von gStreamer[7] sowohl das Farb- also auch das Infrarotbild ausgelesen werden. Wichtig ist dabei die Angabe der korrekten Device-Nummer, welche über den ersten Befehl in Erfahrung gebracht werden kann.

```
ls /dev/video*

gst-launch-0.10 v4l2src device=/dev/video1 ! video/x-raw-yuv,width=640,height=480
! ffmpegcolorspace ! xvimagesink

gst-launch-0.10 v4l2src device=/dev/video1 ! video/x-raw-yuv,width=640,height=488
! ffmpegcolorspace ! xvimagesink
```

Das Farbbild hat dabei eine Größe von 640x480 Pixel, das Infrarotbild jedoch 640x488 Pixel. Es ergeben sich analoge Bilder zu denen in Bild 2.12 dargestellten, jedoch ohne das Tiefenbild.

[7] gStreamer: Ein freies Multimedia-Framework zur Verarbeitung von Multimedia-Datenströmen

2.4 OpenCV-Bibliothek

Bei der Suche nach einer Bibliothek für die eigentliche Bildverarbeitung fiel das Mittel der Wahl schnell auf die freie OpenCV[8]-Bibliothek, welche zurzeit am weitesten verbreitet ist und damit das Maß der Dinge darstellt. Von Intel im Jahre 2006 entwickelt existiert OpenCV seit 2009 in der weiter entwickelten Version 2.x. Im Rahmen dieser Thesis wird die, Anfang 2012 aktuelle und stabile, Version 2.4.4 verwendet.

Die Stärke von OpenCV liegt neben der großen Menge der Algorithmen aus neuesten Forschungsergebnissen in der Rechengeschwindigkeit. Sie eignet sich somit für eine Vielzahl von Anwendungsbereichen, somit besonders für die Echtzeit-Bildverarbeitung. Features wie grundlegende bildverarbeitende Operationen, Bildmanipulation und Filterung, Kontur und Kantendetektion, Segmentierung und Klassifikation, aber auch Tracking und Bewegungserkennung sowie Kamera-Modelle und Kalibrierung sind Bestandteile der Bibliothek. Mit dieser Auflistung wird nochmals deutlich, warum diese Bibliothek für diese Arbeit so vorteilhaft ist.

Die OpenCV-Bibliothek ist in C bzw. C++ konzipiert und läuft unter den gängigsten Betriebssystemen (Linux, Windows und OS X). Neben den Büchern [3] und [4], welche Erklärungen und Anwendungsbeispiele für die wesentlichsten Features von OpenCV bieten, existiert eine mehr oder weniger detaillierte, der aktuellen Bibliotheksversion angepasste, Online-Dokumentation [5].

2.5 Camera Calibration Toolbox für MATLAB

Für die bessere Visualisierung der Kalibrierergebnisse in Kapitel 4.2.2 wird die frei lizenzierte „Camera Calibration“-Toolbox für MATLAB verwendet, welche von Jean-Yves Bouguet am California Institute of Technology entwickelt wurde. Bezogen werden kann diese unter [14], wo ebenfalls eine begrenzte Dokumentation vorliegt.

Mit Hilfe dieser Toolbox können die, im Rahmen dieser Thesis und für die Kalibrierung des Kinect-Sensors verwendeten, Aufnahmen nachträglich und tiefer gehend analysiert und bewertet werden. Die C Implementation dieser Toolbox ist stellt weiterhin die Basis für die Kamerakalibrierung in OpenCV dar, sodass sie die Realisierungen in dieser Thesis analytisch gut ergänzt.

[8] OpenCV: Open Source Computer Vision

3 Analyse und Konzeptionierung

Das dritte Kapitel betrachtet im Rahmen einer ausgiebigen Analyse der Aufgabenstellung die Konzeptionierung eines Lösungsvorschlags für die im nächsten Kapitel umgesetzte Realisierung. Hierbei werden die Randbedingungen diskutiert, die wesentlichen Elemente und Schritte im Vorgehen entwickelt und mögliche Ansätze für die Lösung der Aufgabe definiert.

Im Rahmen der Analyse werden auch Ideen diskutiert und vorgestellt, welche sinnvolle Teillösungen darstellen, jedoch später nicht umgesetzt werden. Dennoch bieten sie Möglichkeiten für nachfolgende Arbeiten und werden zum Teil im Ausblick noch mal aufgegriffen.

3.1 Vorüberlegungen

Zu Beginn der Analyse stehen Vorüberlegungen zum prinzipiellen Vorgehen und zur Positionierung der Kamera im Raum, welche die Basis für die weiteren Ansätze bilden.

Gemäß dem Titel und der Aufgabenstellung dieser Thesis sollen Personen in stark frequentierten Durchgangsbereichen verfolgt und gezählt werden. Hier ist die Vorstellung von einem Flur oder Eingang anzunehmen, d.h. Personen die normal den Gang entlanggehen oder durch eine Tür gehen sollen mit Hilfe des Kinect-Sensors erfasst werden. Dabei sind ausschließlich Personen (Menschen) zu berücksichtigen – andere sich bewegende Objekte sollen also ignoriert werden.

3.1.1 Prinzipielle Vorgehensweise

Die prinzipielle Vorgehensweise zur Lösung der Problemstellung ergibt sich somit wie folgt:

Zunächst müssen die Kinect-Bilddaten empfangen und in gültige 3D-Informationen konvertiert werden. Ist dieses geschehen gilt es Personen in dem Bild zu detektieren. Diese müssen dann über bestimmte Merkmale weiter klassifiziert und von einander unterschieden werden. Somit besitzt jede detektierte Person eine eigene Identität, welche über die Merkmale bestimmt wird. Zu guter Letzt soll die Person noch mittels Tracking verfolgt und gezählt werden. Ein geeignetes Datenmanagement ist hier entscheidend.

Das nachfolgende Bild 3.13 stellt die einzelnen Stufen, welche es abzuarbeiten gilt, noch einmal schematisch dar.

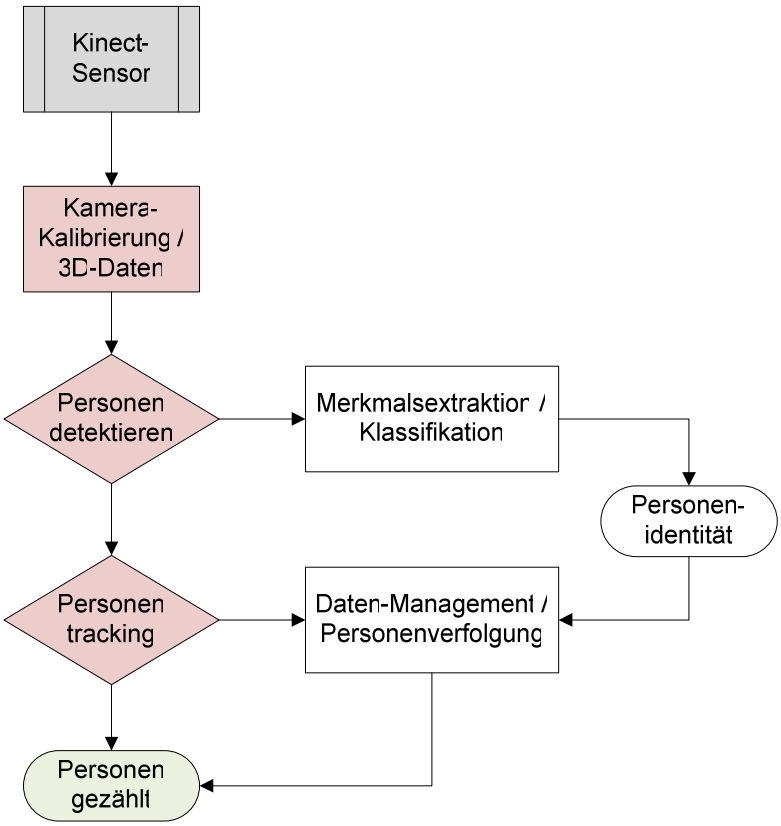

Bild 3.13: Schematische Darstellung der Problemstellung

Dieses Flussdiagramm bildet nunmehr die Arbeitsgrundlage für sämtliche weitere Analysen und die schlussendlich Erstellung des Konzepts.

Den Anfang der tiefer gehenden Vorüberlegungen bildet jedoch zunächst die Frage der Positionierung des Kinect-Sensors.

3.1.2 Positionierung des Kinect-Sensors

Aufgrund der Tatsache, dass vorübergehende Personen erfasst werden sollen und gleichzeitig eine feste Umgebung vorliegt, ist eine geeignete Position des Kinect-Sensors zu definieren.

Eine Möglichkeit wäre es, die Kamera von oben Senkrecht auf den (Durch-)Gang blicken zu lassen (vgl. Pos.1 in Bild 2.1). Eine zweite Möglichkeit ergibt die Positionierung seitlich an einer Wand auf einer Höhe von ca. 160cm (durchschnittliche Augenhöhe) mit Blickrichtung auf die entgegenkommenden Personen (Pos.2). Die dritte sinnvolle Möglichkeit ist die Positionierung mittig über dem Durchgang (angenommene Höhe 2m) auf einer Höhe von durchschnittlich 220cm (Pos. 3).

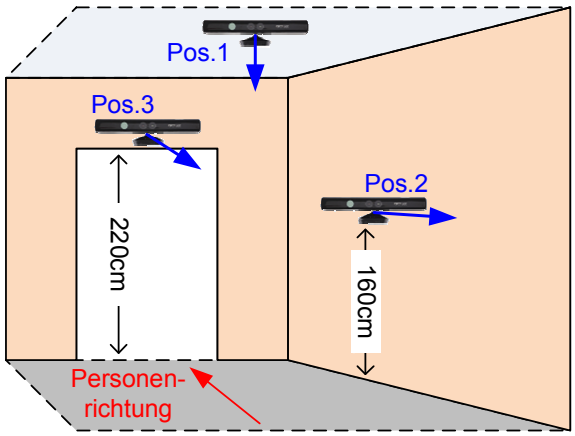

Bild 3.14: Möglichkeiten der Kamerapositionierung

Position 1 ermöglicht die lückenlose Detektion von oben. Alle vorüber ziehenden Personen oder Gegenstände können gegenüber dem Fußboden erfasst werden. Nachteilig wäre hier zu nennen, dass die Tiefeninformationen der Kinect nicht sinnvoll verwendet werden können – denn von oben „sehen alle Menschen gleich aus“ bzw. sind alle Menschen nur anhand Ihrer Körpergröße zu differenzieren. Auch sind Merkmale wie Bekleidung, Gesicht, Oberkörper,… im Farbbild nicht wirklich erkennbar und können somit nicht sinnvoll verwendet werden.

Position 2 hingegen ist von vorne auf die Personen gerichtet. Somit ergeben sich Möglichkeiten in der Unterscheidung der oben genannten Merkmale wie Gesicht oder Oberkörper. Auch ist die Differenzierung von Personen über ihre Entfernung möglich, sofern die Tiefeninformationen des Kinect-Sensors mit verwendet werden. Die Positionierung auf ca. 160cm Höhe eignet sich z.B. besonders gut, um die Gesichter der Personen zu fokussieren, da die Kamera relativ frontal darauf gerichtet ist. Nachteilig an dieser Position ist jedoch, dass Personen sich gegenseitig überlagern können, sodass ggf. dadurch verdeckte Personen nicht erkannt werden können.

Diese Problematik ist bei der Position 3 etwas relativiert, da die Personen zwar immer noch recht frontal, aber mehr von oben herab erfasst werden. Zumindest die Köpfe sind somit fast immer gut sichtbar, außer die Personen gehen direkt hintereinander. Diese Position ermöglicht es also gleichzeitig auf die Entfernung der Person zum Sensor schließen und die Personen anhand Ihrer Gesichter zu differenzieren, sofern diese nicht mit stark gesenktem Kopf den Durchgang entlang gehen.

Nach Aufführung der Vor und Nachteile der einzelnen Positionen für den Kinect-Sensor zur Erfüllung der Aufgabenstellung ergibt sich die **Position 3** als optimale Lösung und verbesserte Variante von Position 2. Position 1 würde sich nur dann eignen, wenn reine 2D-Daten zur Verfügung stehen und die Differenzierung der Personen voneinander nicht von entscheidender Bedingung wären.

3.1.3 Nutzungsmöglichkeiten der 3D-Bilddaten

Wie bereits mehrfach betont liegt der große Vorteil des Kinect-Sensors darin, dass er sowohl ein 2D-Farbbild aber auch ein Tiefenbild zur Verfügung stellt. Kombiniert ergibt sich somit ein dreidimensionaler Raum (RGB-D[9]) mit einer „Pixelwolke“ bzw. ein 2D-RGB-Farbbild, mit einem zusätzlichen Layer für die Tiefeninformationen.

Doch wie lassen sich diese 3D-Informationen nun sinnvoll und lösungsorientiert nutzen?

Detektierte Personen erhalten mit Hilfe der Tiefeninformationen somit eine Position (x,y,z) im Raum und damit ein einfaches Merkmal zur Unterscheidung dieser untereinander. Schließlich können sich zwei Personen nicht zur gleichen Zeit am selben Ort aufhalten. Außerdem ergibt sich nachfolgender angenehmer Nebeneffekt: Tauschen z.B. zwei Personen in einem zweidimensionalen Bildablauf die Positionen, so befinden sie sich in der Mitte kurzzeitig scheinbar an derselben Position. Ist jedoch vorher bekannt, welche Person vorne und welche weiter hinten steht bzw. bei dem Tausch entlang geht, so ist auch im Falle der Überlagerung die Differenzierung noch möglich.

Rechenaufwendigere Möglichkeiten zur Unterscheidung von Personen wären z.B. der Histogramm-Vergleich oder der Vergleich markanter Punkte (vgl. Abschnitt 3.2.1.5). Die Verwendung dieser Verfahren hat mit den 3D-Daten eine hervorragende Alternative erhalten.

Außerdem kann über die Tiefeninformationen der Bereich außerhalb des Arbeitsbereiches über 3,5m (vgl. Abschnitt 2.1.2.2) ausgeblendet werden (siehe Abschnitt 3.2.1.4) und somit zu weiteren möglichen Vereinfachungen führen.

Mit den bis hier getätigten Vorüberlegungen lassen sich nun die einzelnen Stufen der Problemlösung aus Bild 3.13 explizierter analysieren und erste konkrete Konzeptansätze erstellen.

[9] RGB-D: RGB-Farbbild, wobei jeder Pixel zusätzliche Tiefeninformationen (Depth information) besitzt

3.2 Bilderanalyse

Die Bildanalyse – hier aufgrund der Mehrzahl an zur Verfügung stehenden Bilder (Farb- und Tiefenbild) Bilderanalyse genannt – bezeichnet den fundamentalen Prozess des „maschinellen Sehens“. Sie besteht aus den folgenden Punkten:

- ⇒ **Bild(er)aufnahme einer Szene**
- ⇒ **Bildvorverarbeitung**
- ⇒ **Segmentierung** (Erzeugung zusammenhängender aber abgegrenzter Bildregionen – Zerlegung des Bildes in Objekte und Hintergrund)
- ⇒ **Merkmalsextraktion** (Extraktion bestimmter, segmentierter Merkmale als symbolische Informationen)
- ⇒ **Klassifizierung** (Zusammenfassung von Merkmalen zu Objektklassen) **bzw. Mustererkennung** (Verfahren zur automatischen Einordnung von Merkmalen in Kategorien)
- ⇒ **Aussage**

Um die einzelnen Schritte zur Bewältigung der Aufgabenstellung zu vollziehen, ist es sinnvoll dieses Schema zu verfolgen und dabei die einzelnen Punkte, ggf. mehrfach, nach und nach abzuarbeiten. Dieses wird in den nachfolgenden Abschnitten detailliert beschrieben, wobei vornehmlich die später verwendeten Verfahren des Anwendungsprogramms dargestellt und diskutiert werden.

Da die zur Verfügung stehenden, aufgenommenen Bilder bereits in Kapitel 2 sowie die Bildvorverarbeitung, welche sich auf die Kalibrierung von Farb- und Tiefenbild, d.h. dessen Entzerrung und Überlagerung, beschränkt, in Abschnitt 2.3 theoretisch behandelt wurden, sei an dieser Stelle mit der Segmentierung und Merkmalsextraktion als erstem Punkt begonnen.

3.2.1 Ansätze zur Segmentierung und Merkmalsextraktion

In diesem Abschnitt sollen die bildverarbeitungstechnischen Möglichkeiten und Ansätze für die Problemlösung theoretisch vorgestellt. Dabei werden später nicht alle Ideen in das Konzept übernommen, jedoch sollen hier einige andere Verfahren mit vorgestellt werden, welche sinnvoll oder ggf. ergänzend sein können.

3.2.1.1 Background subtraction

Einen ersten Ansatz zur Detektion von bewegten Objekten bzw. Personen in einer Folge von Bildern stellt die Differenzbildberechnung dar. Hierbei werden zwei aufeinander folgende Bilder Pixelweise voneinander subtrahiert. Auf diese Weise werden die Bereiche des Bildes, welche sich stark gegenüber dem vorangegangenen verändert haben, hervorgehoben. Anders ausgedrückt werden in Bezug auf die Veränderung homogene Bereiche ausgelöscht.

Eine Erweiterung dieses Prinzips stellt die „background subtraction“ [3, S.265ff] (Hintergrundsubtraktion) dar. Das Ziel dieses Verfahrens ist die Segmentierung von Vordergrundobjekten aus Bildern, wobei die Einteilung der Objekte in Vorder- und Hintergrundobjekte aufgrund ihrer Dynamik in der Szene erfolgt. Dafür wird ein Hintergrundmodell erzeugt, welches mit jedem neu eingehenden Bild verglichen wird. Auf diese Weise wird Pixel für Pixel entschieden, ob es zum Hintergrund oder zum Vordergrund gezählt wird. Das Ergebnis ist ein Binärbild als Maske hierfür. Außerdem werden regelmäßig die Informationen des Hintergrundmodels aktualisiert, sodass Änderungen in der Hintergrundszene beachtet werden können. Am Ende stehen dem Anwender ein binäres Vordergrundbild und ein immer aktuelles Hintergrundbild zur Verfügung.

Mit Hilfe dieses Verfahrens der Hintergrundsubtraktion ist es also möglich, Objekte, welche sich in einer Bildfolge bewegen, zu detektieren. Eine Differentiation zwischen Objekt und Mensch ist auf diese Weise allein jedoch in keinster Weise denkbar.

3.2.1.2 Weiterführende Verfahren zur Personenidentifikation

Ähnlich zur „background subtraction“ verhält es sich mit anderen, kanten- oder konturbasierten, Verfahren. Objekte lassen sich damit segmentieren, jedoch nicht ohne weiteres als Person identifizieren. Hierfür sind weiterführende Prozeduren notwendig, die Merkmale an Personen extrahieren und diese als „typisch menschlich“ Klassifizieren. Denkbar sind Extraktionsverfahren, welche z.B. die Form oder Struktur des Objektes weitergehend analysieren.

Ein sicheres Verfahren stellt dabei die Gesichtsdetektion dar. Sie wird nachfolgend vorgestellt.

3.2.1.3 Viola und Jones - Gesichtsdetektion

Der Algorithmus zur Detektion von Gesichtern in Bildern nach Paul Viola und Michael Jones [15] ist seit der Einführung im Jahr 2001 das zurzeit gängigste Mittel zum Zweck. Wie in der Motivation bereits beschrieben, haben aktuelle Digitalkameras und Smartphones dieses verbreitete Verfahren implementiert, um bei Aufnahmen z.B. die Scharfstellung oder Belichtung von Personen bzw. von Gesichtern zu erreichen. Dieser Unterabschnitt soll die Funktionsweise darstellen.

Ziel der Gesichtsdetektion ist es, ein Gesicht sicher zu finden und den irrelevanten Hintergrund zu verwerfen. Die Grundlage dafür bildet ein Fenster bestimmter Größe, welches über das gesamte zu untersuchende Bild geschoben wird. Jeder Bildausschnitt wird dann auf Merkmale hin untersucht. Eine Auswahl an Merkmalen stellen die nachfolgend dargestellten Haar-Features dar:

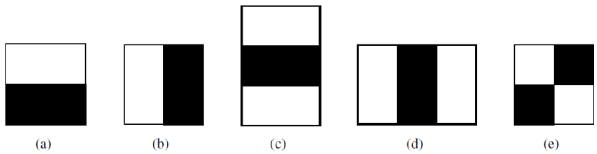

Bild 3.15: Haar-Features der Gesichtsdetektion, Quelle [15]

Diese Features (nach den Haar-Wavelets benannt) berechnen Helligkeitsdifferenzen innerhalb des jeweiligen Bildausschnittes, wobei die Lage und Skalierung völlig frei wählbar ist. Großflächige Features filtern dabei niederfrequente Merkmale (Helligkeitsunterschiede) heraus, schmale Features filtern hochfrequentes (z.B. Ecken oder Kanten). Die Berechnung erfolgt über die Summation von Pixeln in den jeweiligen Blockregionen. Bei einer Standardgröße des Fensters von 24x24 Pixel ergeben sich beim Viola und Jones Algorithmus insgesamt zwischen 160.000 und 180.000 solcher, unterschiedlich skalierter Merkmale. Eine so beschriebene Berechnung wäre zu keiner Laufzeit realistisch durchführbar.

Aus diesem Grund wird die Verwendung des so genannten Integralbildes eingeführt. Dabei ergibt sich auf jedes Pixel die Summe über alle Pixel links und oberhalb von ihm. Ein Beispiel ist in nachfolgendem Bild 3.16 dargestellt.

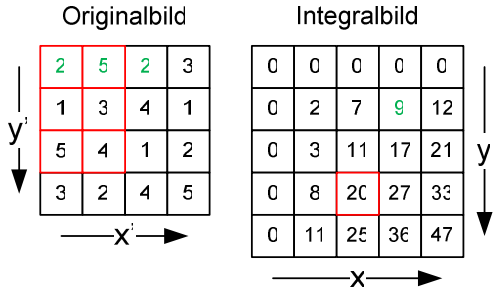

Bild 3.16: Wirkprinzip des Integralbildes

Der Pixelwert im Integralbild mit der roten Umrandung berechnet sich aus der Summe aller im originalen Bild rot markierten Pixel. Ebenso verhält es sich mit dem grünen Beispiel.

Gemäß [15, S.2] ergeben sich folgende mathematische Beziehungen für die einzelnen Pixel, wobei *ii(x,y)* für ein Pixel des Integralbildes, *i(x',y')* für ein Pixel des Originalbildes stehen:

$$ii(x,y) = \sum_{x'=0}^{x' \le x} \sum_{y'=0}^{y' \le y} i(x',y') \tag{11}$$

Ohne Summenzeichen lassen sich die Integralpixel in zwei Schritten anhand ausgewählter anderer Pixel bestimmen. Hier ein Beispiel in Anlehnung an das rote Integralpixel in Bild 3.16:

$$\begin{aligned} &s(x',y') = s(x',y'-1) + i(x',y') \text{ , hier: } s(1,2) = (5+3)+4 = 12 \\ &ii(x,y) = ii(x-1,y) + s(x',y') \text{ , hier: } \underline{\underline{ii(2,3)}} = 8+12\underline{\underline{=20}} \end{aligned} \tag{12}$$

Diese zwei Gleichungen zeigen, dass sich die Anzahl an Berechnungsschritten verringert, da die Werte der Integralpixel von den Summen anderer Pixel abhängig sind. Mit jeweils vier Zugriffen auf das Integralbild lassen sich nun neue Pixel(summen) berechnen.

Auf das Integralbild angewendet lassen sich schließlich die in Bild 3.15 vorgestellten Haar-Features so gebrauchen, dass die Pixel in den weißen Bereichen aufsummiert und von denen im schwarzen Bereich subtrahiert werden. Der so erzeugte Summenwert stellt eines der Merkmale für die spätere Klassifikation dar. Auf die unterschiedlich aufgebauten und skalierten Features angewendet ergibt sich eine Vielzahl von Merkmalen. Das nachfolgende Bild stellt zwei davon beispielhaft vor:

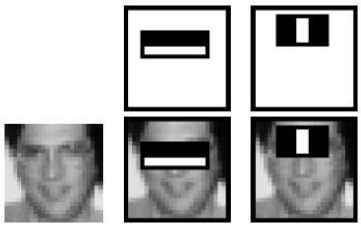

Bild 3.17: Anwendungsbeispiel zweier Haar-Features auf ein Gesicht, Quelle: [15]

Das erste Merkmal vergleicht die Helligkeit der Augenpartie (dunkler) mit dem oberen Wangenbereich (heller). Das zweite Feature bestimmt die Helligkeit der Augen (ebenfalls dunkler) mit denen des Nasenbeins (auch heller). Aber auch Merkmale wie Kanten lassen sich wie beschrieben mit Hilfe dieser Features extrahieren. Um die Erkennung dieser zu verbessern, ist es möglich auf das Bild ein Canny-Edge-Filter anzuwenden, welcher über lokale Gradienten die Kanten nachbearbeitet. Auf die genaue Funktionsweise wird an dieser Stelle jedoch nicht weiter eingegangen, da die Betrachtung der anderen Verfahren für diese Arbeit von größerer Bedeutung ist.

Die Funktionsweise der, auf die Merkmalsextraktion folgende, Klassifikation wird als Abschluss der erfolgreichen Gesichtsdetektion nachfolgend behandelt.

Die Merkmale, welche mit Hilfe der zuvor vorgestellten Verfahren gefunden werden können, sind alleine als Schwach anzusehen. Schließlich gibt es z.B. Helligkeitsunterschiede in bestimmten Bereichen von Unterbildern auch auf vielfältige Art und Weise in Bildern ohne ein Gesicht. Aus diesem Grund werden einzelne Haar-Features zu Klassifikatoren zusammengefasst, welche über einen Mehrheitsentscheid gemeinsam entscheiden, ob das betrachtete Objekt ein Gesicht und damit eine Person darstellt oder nicht. Dabei gibt es schwache Klassifikatoren, die lediglich ein Merkmal bewerten, aber auch Steigerungen zu komplexen Klassifikatoren, welche eine Vielzahl an Merkmalen auf sich vereinigen.

Trainiert werden diese Klassifikatoren mit einer Auswahl von Testbildern unterschiedlichster Personen. Dabei haben Fehldetektionen eine höhere Gewichtung im Entscheidungsprozess als positive, was dazu führt, dass später bereits kleinste Abweichungen zum Ausschluss führen. Demnach werden nur sicher klassifizierte Merkmale akzeptiert. Bekannt ist diese Art der Klassifikatoren unter dem Namen „AdaBoost“, was für Adaptive Boosting steht. Der Name

geht darauf zurück, dass die Konstellation der Merkmale sowie die Art der Entscheidungsfindung auf eine Beschleunigung in der Klassifikation führen.

Zu guter Letzt werden die Klassifikatoren in einer bestimmten Reihenfolge kaskadiert, sodass am Ende der Gesichtsdetektion nach Viola und Jones ein Kaskadendetektor steht. Bild 3.18 zeigt schematisch die Funktionsweise:

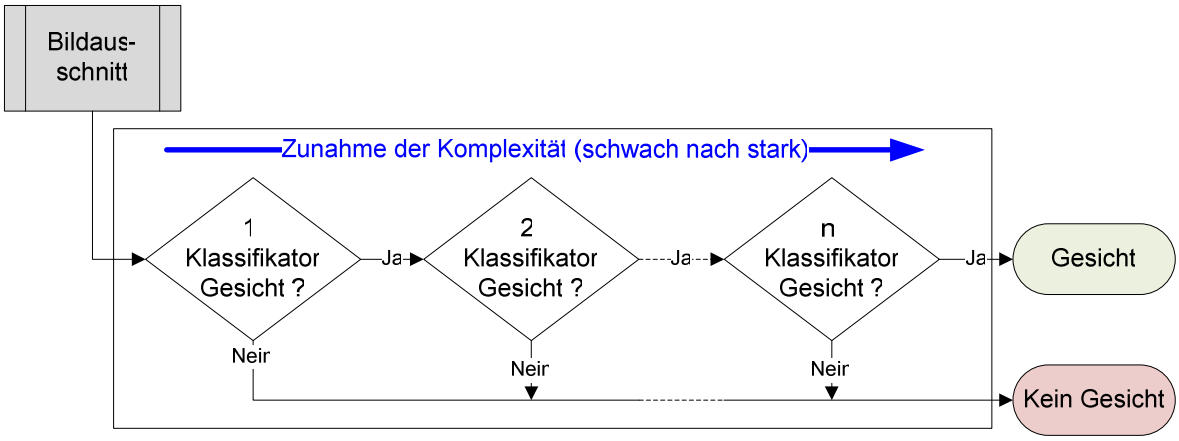

Bild 3.18: Schematische Darstellung eines Kaskadendetektors

Zu Beginn der Kaskade stehen einfache (schwache) Klassifikatoren, welche schnell arbeiten und damit bei kurzer Rechenzeit eine erste Entscheidung über eine positive oder negative Detektion machen können. In den weiteren Stufen werden die Klassifikatoren immer stärker und erhöhen dabei die Wahrscheinlichkeit auf eine positive Detektion. Doch sobald auch nur eine Stufe als negativ verlassen wird, wird der gesamte Kaskadendurchlauf beendet und so unnötige Berechnungen verhindert. Starke, aber auch langsame Klassifikatoren kommen ausschließlich in der höchsten Stufe zur Anwendung, um die Fehlerraten zu minimieren.

Bei Viola und Jones kommen schlussendlich 38 Kaskadenstufen mit insgesamt über 6000 Merkmalen (unterschiedlichen Haar-Features) zum Einsatz.

3.2.1.4 Segmentierung der Tiefeninformationen

Ähnlich dem in Abschnitt 3.2.1.1 vorgestellten Prinzip der „background subtraction“ lassen sich auch mit Hilfe der Tiefeninformationen der Kinect Objekte in einem definierten Arbeitsbereich direkt identifizieren. Segmentiert man das Tiefenbild mit einem festen Schwellwertverfahren (Thresholding) auf eine Entfernung von wie gewünscht 3,5m, so werden alle Objekte über der Schwelle als Hintergrund verworfen (die Pixel werden zu Null gesetzt). Die verbleibenden Objekte vor der Schwelle sind dann in weiß die Objekte. Auf diese Weise lässt sich das Tiefenbild als Maske für das Farbbild nutzen, sodass auch dieses auf die interessanten Bereiche beschränkt segmentiert werden kann. Näheres hierzu ist der entsprechenden Realisierung in 4.2.3 zu entnehmen.

Wie dort näher beschrieben, ist hierfür die morphologische Basisoperation des „Openings“ eine sinnvolle Ergänzung (vgl. [16]). Beim Opening werden überstehende Fransen an Objekten abgetragen. Das Resultat ist ein quasi rausch-gefiltertes, d.h. von weniger kleinen Störobjekten und Fehlerpixeln durchsetztes, Bild. Das Prinzip des Openings vereinnahmt die grundlegenden Operationen der Erosion und Dilatation in dieser Reihenfolge. Bild 3.19 veranschaulicht das Prinzip anhand eines symmetrischen 3x3-Pixel großen Strukturelements (*B*):

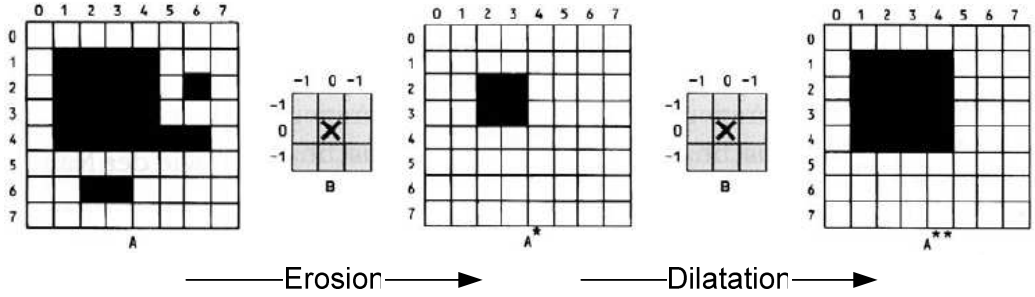

Bild 3.19: Opening – eine morphologische Basisoperation (schwarzes Objekt), Quelle: [16]

Werden für die Erosion und die Dilatation unterschiedlich große Strukturelemente (B_E und B_D) verwendet, so lassen sich Strukturen bis zur Elementgröße B_E (z.B. 10x10 Pixel) eliminieren (Entrauschung). Das gleichzeitig verkleinerte Objekt A* wird danach bei der Dilatation jedoch mit einem größeren Strukturelement B_D (z.B. 15x15 Pixel) um 10 Pixel im Durchmesser vergrößert. Der Vorteil dieser Möglichkeit wird ebenfalls später im oben benannten Abschnitt zur Realisierung begründet.

3.2.1.5 SURF/SIFT-Algorithmus und Histogrammvergleich

SURF (engl. Speeded Up Robust Features) ist ein beschleunigter Algorithmus für die robuste Erkennung von markanten Merkmalen in Bildern. Die Basis dabei bilden Mittelwertfilter und erneut Integralbilder. Vorangegangen ist die „Scale-invariant feature transform“ (SIFT), welche hingegen mit Gauß-Filtern arbeitet. Die nachfolgenden Ausführungen beziehen sich auf [3, S.321].

Nachdem ein Bild mit Hilfe des entsprechenden Filters geglättet wurde, werden die markanten Punkte ermittelt – sie sind dann markant, wenn sie stark von ihrem Hintergrund abweichen. Hierzu werden allen Punkten ihre Gradienten (z.B. an Ecken oder Kanten) zugeordnet und lokal als Histogramm gespeichert. Die herausragendsten Gradienten bleiben dann als Merkmale übrig. Der große Vorteil dieser Merkmale ist, dass sie unempfindlich gegenüber Beleuchtungsänderungen und perspektivischen Verzerrungen (Rotation, Translation, Skalierung) sind. Normalerweise werden für diese Verfahren, aufgrund der überwiegenden Verfügbarkeit, Grauwertbilder verwendet. Es lassen sich jedoch in Farbbildern mit ihrem dreidimensionalen Merkmalsraum aufgrund der um Zwei erweiterte Dimension mehr und bessere robuste Merkmale finden. Die Verwendung von Farbbildern für den SURF- oder SIFT-Algorithmus ist also unbedingt empfehlenswert.

Mit Hilfe des, hier ebenfalls nicht näher beschriebenen, RANSAC[10]-Algorithmus lassen sich dann in zwei Bildern mit markanten, homologen Punkten Übereinstimmungen suchen und diese somit vergleichen.

Aufgrund der absoluten Invarianz gegenüber Verzerrungen und verminderten Anfälligkeit auf Beleuchtungsänderungen lässt sich dieser Algorithmus besonders gut zur Unterscheidung von Gesichtern einsetzten. Jedes Gesicht besitzt einzigartige Merkmale, welche sich auf diese Weise gut extrahieren und mit Merkmalen anderer Gesichter vergleichen lassen. Bezogen auf dieses Projekt, wo Gesichter zu unterschiedlichen Zeitpunkten an unterschiedlich beleuchteten Orten und damit auch in unterschiedlicher Rotation und Skalierung detektiert und untereinander verglichen werden sollen, stellt das hier beschrieben Prinzip des SURF/SIFT das Mittel zur Wahl dar.

Dieser Entscheidung vorangegangen waren Ansätze über pixelbasierte Merkmalsextraktionen wie z.B. der Histogrammverlgleich. Hierbei wird, ebenfalls auf Grundlage der Gesichtsdetektion, ein Histogramm der Farbverteilung im Gesicht einer Person erzeugt. Auch diese Histogramme (von verschiedenen Personen) lassen sich untereinander vergleichen. Da aber unter anderem von einer möglichen Änderung der Beleuchtung auch die Farbe in einem Bild abhängig ist, kann es hier passieren, dass sogar das gleiche Gesicht beim Vergleich zweier Histogramme nicht als identisch identifiziert werden kann. Aus diesem Grund wird dieses Verfahren als Ansatz wieder verworfen und lediglich der SURF/SIFT-Algorithmus beibehalten.

3.2.1.6 Räumliche Koordinaten als Merkmale zu Personen

Die letzten, wichtigen Merkmale stellen die Positionen der Personen im Raum dar. Sie werden aus den Mittelpunkten der Gesichter bestimmt und bestehen schließlich aus einer x, einer y und einer z-Koordinate, wobei die ersten beiden aus der Pixelposition im Farbbild stammen und letzte die Information aus dem Tiefenbild repräsentiert.

[10] **ran**dom **sa**mple **c**onsensus: Übereinstimmung mit einer zufälligen Stichprobe

3.2.2 Klassifikation und Datenmanagement

Dieser Abschnitt soll die Ansätze zur Klassifikation der im vorangegangenen Abschnitt extrahierten Merkmale beschreiben.

Ein Objekt im Rahmen dieser Arbeit lässt sich als Person Klassifizieren bzw. Kategorisieren, wenn es die folgenden Merkmale besitzt:

- Das Objekt befindet sich im festgelegten Arbeitsbereich des Kinect-Sensors.
- Das Objekt besitzt ein Gesicht.
- Das Objekt besitzt eine Position im Raum, welche anhand der 3D-Koordinaten bestimmt ist.

Den Anfang macht dabei das segmentierte Tiefenbild. Durch die Maskierung des Farbbildes werden ausschließlich Objekte im Vordergrund des Farbbildes für die weitere Analyse zugelassen. Mit Hilfe der Gesichtsdetektion wird ferner bestimmt, ob das Objekt ein Mensch ist. Dadurch, dass der Hintergrund der relevanten Objekte vollkommen und homogen schwarz ausgeblendet ist, kann der Kaskadendetektor hier bereits in den ersten Stufen der Detektion abbrechen. Selbst mit den schwächsten Klassifikatoren können keinerlei sinnvolle Merkmale verifiziert werden. Dies führt zu einer erheblichen Performancesteigerung. Zuletzt werden noch die Koordinaten des detektierten Gesichts bestimmt.

Alle gesammelten Daten werden für jede Person separat in einer Struktur für das Datenmanagement abgelegt. Diese soll die folgenden Elemente enthalten:

- Gespeichertes Gesicht der Person als Bild
- Aktuelle Position im Arbeitsraum
- Hinweis, ob die Person aktuell (aktiv) detektiert wird oder über Vorhersagemechanismen getracked wird
- Einen Zähler für die Anzahl an erfolgreichen Detektionen der Person inkl. Zeitstempel der letzten Erfassung
- Hinweis, ob die Person bereits vom System als gezählt erfasst wurde

Für den Fall, dass zwei Filter für eine Person erstellen wurden, wobei die Gründe hierfür an dieser Stelle irrelevant sind, müssen anhand der abgelegten Bilder der Gesichter an sich verglichen werden. Hierfür werden die mittels SURF/SIFT-Algorithmus extrahierten Merkmale klassifiziert. Abhängig vom Ergebnis sind weiterführende Prozeduren durchzuführen.

3.3 Tracking mittels Partikelfilter

Die bis hierhin erfolgreich detektierten und klassifizierten Personen müssen nun in einer kleinen Datenbank verwaltet werden (vgl. Abschnitt 3.2.2). Da das Datenmanagement und das Tracking der Personen eng miteinander verknüpft sind, sollten sie auch zusammen betrachtet und vor allem später realisiert werden.

Das Hilfsmittel, welches im Rahmen dieser Arbeit für das Tracking gewählt wurde, ist das so genannte Partikelfilter. Dieses gehört zu der Gruppe der Schätzfilter, zu der u.a. auch das bekanntere Kalmanfilter zählt. In den nachfolgenden Ausführungen soll in Anlehnung an [3, S.364ff] dargelegt werden, was die Grundlage des Partikelfilters ist und warum es sich gut für die Realisierung des Trackingproblems in diesem Anwendungsbeispiel eignet.

Das Fundament eines Partikelfilters stellt der 1998 von Michael Isard und Andrew Blake veröffentlichte Condensation-Algorithmus dar. Das bis zu dem Zeitpunkt gängige Verfahren zur Objektverfolgung war das bereits erwähnte Kalman-Filter. Der große Vorteil eines Partikelfilters ist der, dass es im Gegensatz zum Kalmanfilter die gleichzeitige Verfolgung mehrerer Hypothesen für die Zustandsschätzung ermöglicht. Diese Eigenschaft macht das neue Filter sehr Robust gegenüber Störungen wie z.B. die vorübergehende Verdeckung des zu trackenden Objekts. Dabei bleibt der Algorithmus in seiner Rechenintensität so überschaubar, dass er nahezu in Echtzeit arbeiten kann.

Die vom Filter erzeugten Partikeln repräsentieren dabei die besagten Hypothesen über den Zustand des verfolgten Objektes. Dem entsprechend, wie gut diese Hypothese, wie hoch also die Wahrscheinlichkeit hierfür, ist, kann sie entweder verworfen oder weiterverfolgt werden.

Der Condensation-Algorithmus arbeitet mit Modellen in zwei Schritten. Mit Hilfe des aktuellen bzw. letzten Zustandes eines Objektes wird ein zu erwartender Folgezustand berechnet/geschätzt. Maßgebend für dieses Bewegungsmodell ist die Position des Objekts, welches für diese Arbeit als statisch angesehen und somit im Verlauf nicht optimiert wird. Mit dem zweiten Modell, dem Beobachtungsmodell, wird der geschätzte Zustand mit dem neu gemessenen verglichen und die Partikelverteilung neu arrangiert. Diese ist jedoch nicht über den gesamten Zustandsraum gleichverteilt, sondern auf den Bereich der Hypothesen mit den höchsten Wahrscheinlichkeiten konzentriert (vgl. Bild 4.28). Bild 3.20 veranschaulicht das Prinzip des Algorithmus noch einmal.

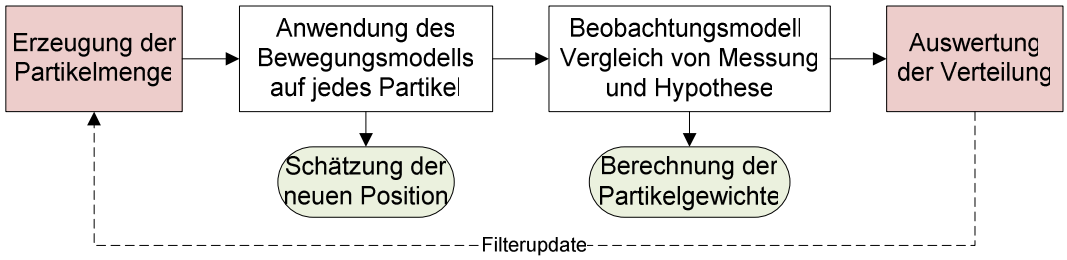

Bild 3.20: Funktionsweise des Partikelfilters (Condensation-Algorithmus)

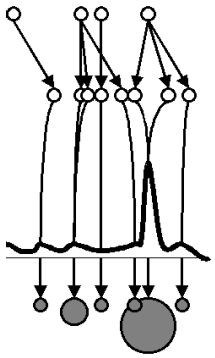

Bild 3.21: Partikel- / Verteilungsdarstellung

Die Auswertung erfolgt z.B. über den Mittelwert der Wahrscheinlichkeiten, sodass sich am Ende eine zu erwartende Objektposition ergibt. Allerdings liefert der Mittelwert nur dann sinnvolle Werte, wenn die Verteilung unimodal ist, d.h. wenn sie ausschließlich einen Spitzenwert hat. Das gleichzeitige Tracking von mehreren Personen über nur einen Partikelfilter ist demnach nicht sinnvoll. Jede Person sollte ein eigenes Filter bekommen.

In [3] wird noch darauf hingewiesen, dass der Condensation-Algorithmus von Bewegungen mit konstanten Geschwindigkeiten ausgeht und deshalb nur Objekte, die diese Bedingung erfüllen, zuverlässig verfolgen kann. Dies trifft auch auf die zu trackenden Personen im Rahmen dieser Arbeit zu – plötzliche Positionssprünge oder Änderungen in der Gehgeschwindigkeiten werden nicht angenommen.

3.4 Zusammenfassung – Das Konzept

Nachdem in diesem Kapitel grundlegend mögliche Verfahren zur Personenverfolgung und –zählung vorgestellt, im Detail betrachtet und analysiert wurden, gilt es abschließend ein Gesamtkonzept für die Realisierung zu formulieren. Dieses soll die essentiellen und für die Lösung der Problemstellung optimalen Methoden beinhalten. Das Konzept fasst dabei im Wesentlichen die Ergebnisse aus Bild 3.13 und dem Abschnitt 3.2.2 zusammen.

Am Anfang steht der Empfang der beiden Bilddaten. Sowohl das Farbbild, als auch das in der Kinect berechnete Tiefenbild müssen von der Software entgegengenommen werden. Für einen optimalen Überblick über den zu überwachenden Durchgangsbereich ist die Kamera dabei in einer Höhe von ca. 220cm positioniert. Dies entspricht der Oberkante einer üblichen Tür von 2m Durchgangshöhe.

Die Bilder werden nach dem Erhalt vorverarbeitet, d.h. sie werden entzerrt und kalibriert. Hierfür sind die intrinsischen und extrinsischen Parameter beider Bildsensoren zu ermitteln bzw. als Ergebnis einer vorherigen Kalibrierung aus einer Datei zu laden. Mit der abgeschlossenen Überführung von Farb- und Tiefenbild ineinander, gilt es das Arbeitsbild zu konstruieren.

Das Arbeitsbild betrachtet einen Bereich von ca. 45cm bis 3,5m vor der Kinect-Kamera. Hierzu wird mit Hilfe der Tiefeninformationen das Farbbild durch eine Binärmaske (im Rahmen dieser Arbeit als Schleier bezeichnet) in seinem Sichtbereich beschränkt. Gleichzeitig besitzen die noch sichtbaren Pixel nun jeweils Entfernungsinformationen aus dem Tiefenbild.

Aus dem so bearbeiteten RGB-D-Bild gilt es nun, mit Hilfe des als optimal befundenen Verfahrens zur Gesichtsdetektion von Viola und Jones, Personen anhand ihrer im Arbeitsbereich befindlichen Gesichter zu detektierten.

Über die gefundenen Gesichter erhalten die Personen gleichzeitig in Form von Koordinaten einen Bezug im 3D-Raum. Ihre Positionen sind somit determiniert und lassen sich mittels Tracking verfolgen bzw. vorhersagend schätzen. Dabei kann primär ausschließlich auf die Koordinaten zurückgegriffen werden, da diese Eindeutig genug sind um die Personen zu differenzieren.

Das Mittel der Wahl hierfür stand seit Beginn dieser Thesis fest. Es sollten Partikelfilter verwendet werden. Allerdings haben die Analysen gezeigt, dass für jede Person, die verfolgt wird, ein eigenes Filter initialisiert werden muss. Gleichzeitig ergibt sich, dass es sinnvoll ist die Daten zu den Personen ebenfalls in Zusammenhang mit den Filtern zwischen zu speichern.

Der eigentliche Vorgang der Zählung der Personen ergibt sich durch die, im Datenmanagementsystem abgelegten, Informationen. Ist eine Person, neben anderen Bedingungen, oft genug erfasst worden, so gilt sie als Gezählt und die Lösung der Problemstellung als erfüllt.

4 Realisierung

Kapitel vier stellt die Realisierung des im vorangegangenen Konzepts vor. Diese beinhaltet dabei größtenteils die Umsetzung in ein funktionsfähiges C++-Programm. Schritt für Schritt werden dabei die entwickelten Komponenten des Quellcodes vorgestellt und so nacheinander die einzelnen Funktionen beschrieben und zusammengefügt.

Am Ende dieses Kapitels steht dann das vollständige Programm, welches die im Rahmen dieser Arbeit entwickelten Verfahren zur Personenverfolgung und –zählung beinhaltet. Zu Beginn steht jedoch der Empfang der Bilddaten von der Kinect-Kamera, welche im nachfolgenden Abschnitt beschrieben wird.

An dieser Stelle sei noch darauf hingewiesen, dass auf die Darstellung von Quellcodes in diesem Kapitel größtenteils verzichtet wird. Lediglich in Ausschnitten werden zum besseren Verständnis einzelne Passagen bzw. die Funktions- oder Methodenaufrufe namentlich genannt. Die Hauptprogramme und Header-Files sind jedoch dem Anhang B zu entnehmen, tiefer gehende Funktionen und Klassendarstellungen hingegen der beiliegenden CD-ROM.

4.1 Empfang der Bilddaten

Um die Bilddaten, welche der Kinect-Sensor erzeugt und zur Verfügung stellt, zu nutzen, wird der in Abschnitt 2.3 beschriebene Treiber verwendet. Der Treiber bildet jedoch nur die Schnittstelle zwischen Hardware und Computer. Um auf das Farb- und das Tiefenbild, aber auch Steuerungselemente wie den Motor und die LED der Kinect zugreifen zu können, muss in dem zu entwickelnden Programm ein Programmteil eingefügt werden, welcher regelmäßig diese Schnittstelle anspricht und die Daten abruft.

Grundlage der dafür zuständigen und unter Anhang B.2 dargestellten Klasse `class MyFreenectDevice` stellt dabei das „C++OpenCvExample“[11] der OpenKinect-Kommunity [13] dar. Dieses musste jedoch zum größten Teil umgeschrieben und so modifiziert werden, dass es mit OpenCV und dem aktuellen Treiber in deren jeweils verwendeten Versionen kompatibel ist. Die Treiberseitig wichtigen Header-Files heißen `libfreenect.hpp` mit seinen Wrapper-Klassen für C++ und `libfreenect.h`. Sie liefern weitere Informationen zu einzelnen Funktionen und Definitionen und sind deshalb für das Verständnis so wichtig, da es zurzeit keine befriedigende Dokumentation für den freenect-Kinect-Treiber gibt.

Die oben beschriebene Klasse sorgt dafür, dass die im Hintergrund als MUTEX-pThread[12] ablaufenden Abrufe über die Methoden `VideoCallback()` und `DepthCallback()` das

[11] http://openkinect.org/wiki/C%2B%2BOpenCvExample
[12] Mutex-Verfahren (**mut**ual **ex**clusion) verhindern, dass nebenläufige Prozesse bzw. Threads gleichzeitig oder zeitlich verschränkt, gemeinsam genutzte Datenstrukturen unkoordiniert verändern.

jeweils aktuell zu Verfügung stehende Farb- bzw. Tiefenbild (noch in der Rohdatenform) als entsprechende `cv::Mat` Variable speichern. Diese stellen den aktuell standardmäßig verwendeten Datentyp für Bilder in OpenCV dar. Da diese Art von Threads auch in der OpenCV-Bibliothek definiert sind, dürfen sie an dieser stelle nicht noch einmal deklariert werden. Dies stellt die erste Veränderung zu dem Beispielcode dar.

Die weitaus aufwendigere Veränderung ist in der Verwendung der Treiberversion 0.2 zu begründen. Hier wurden grundlegende Funktions- und Methodenaufrufe geändert, welche in dem für Version 0.1 geschriebenen Beispiel demnach nicht enthalten sind.

Mit Hilfe der Methoden `getVideo()` und `getDepth()` erhalten andere Programme oder Programmteile Zugriff auf die bisher nur innerhalb dieser Klasse verarbeiteten Bilddaten. Das Farbbild wird dabei in ein `CV_8UC3-Mat` (8Bit-3Layer-Matrix) konvertiert, das Infrarotbild in ein `CV_8UC1-Mat` (8Bit-1Layer-Matrix) und das Tiefenbild steht hier noch als `CV_16UC1-Mat` (16Bit-1Layer-Matrix) zur Verfügung. Grundsätzlich werden für diese Anwendung von Anfang an alle Bildgrößen auf 640x480 Pixel beschränkt bzw. definiert.

Die Klasse enthält mit der Variablen `lut_depth` eine Loop-Up-Table (LUT), welche die in Abschnitt 2.1.2.2 dargestellten Entfernungswerte der Gleichung (1) liefert. Aufgerufen in der Methode `calcDepth()` kann so später bei Bedarf einem quantisierten Tiefenwert eine reale Entfernung in Millimetern zugeordnet werden.

Die letzten beiden Methoden dieser Klasse (`setVideoMode()` und `getVideoMode()`) machen Änderungen am aktuellen Video-Modus der Kinect möglich bzw. können diesen auslesen. Hier ist es möglich die standardmäßige Ausgabe von Farb- und Tiefenbild auf das Infrarotbild umzustellen und wieder zurück. Als Format kann also `FREENECT_VIDEO_RGB` oder `FREENECT_VIDEO_IR_8BIT` gewählt werden.
Eine weitere Besonderheit hier stellt eine Veränderung der Bildauflösung in der Größe dar, welche seit der Treiberversion 0.2 implementiert ist. Es ist nämlich möglich, der Kinect größere oder kleinere Bilder zu entlocken.

Parameter	Auflösung [Pixel]	Frame-Rate [fps]
`FREENECT_RESOLUTION_MEDIUM` (Standard)	VGA 640x480	30
`FREENECT_RESOLUTION_HIGH`	SXGA 1280x1024	10
`FREENECT_RESOLUTION_LOW`	QVGA 320x240	unbekannt

Tabelle 4.2: Mögliche Bildauflösungen und Frame-Rates seit Treiberversion 0.2

Nichts desto trotz wird im Rahmen dieser Arbeit ausschließlich die Standardauflösung von 640x480 Pixel verwendet.

Der Depth-Modus kann standardmäßig von vorn herein zwischen 11 Bit oder 10 Bit Auflösung umgeschaltet werden. Standardmäßig ist eine Tiefenauflösung von 11 Bit (2048 Quantisierungsstufen) eingestellt.

In der Main-Funktion muss nun ein Objekt der Klasse `MyFreenectDevice` erzeugt werden, welches den Zugriff auf die Kinect dann ermöglicht. Dieses ist in den Zeilen 73-79 des Hauptprogramms (Anhang B.1) wieder zu finden. Auch der Start von Video- und Depth-Stream ist dort implementiert. Die Zeilen 255-258 zeigen hingegen die Beendigung des Kinect-Zugriffs. Dazwischen werden über die Funktionen `f_getDepth()` und `f_getVideo()` der `functions.h` (vgl. Anahng B.5) die Bilder endgültig empfangen und Vorverarbeitet. Diese Vorverarbeitung beschränkt sich beim Farbbild jedoch auf eine vertikale Spiegelung des Bildes, beim Tiefenbild kommt zusätzlich noch die Umrechnung vom 16 Bit auf ein sehbares 8 Bit Grauwertbild hinzu. Mit diesen Schritten ist sowohl die Sichtweise für die spätere Anwendung besser, aber auch die weiterführende Verarbeitung des Tiefenbildes ist so einfacher.

4.2 Kamerakalibrierung und Arbeitsbildgenerierung

Nachdem die Bilder von dem Kinect-Sensor nun empfangen werden können, gilt es sich mit dem Problem der Verzerrungen und Verschiebung der beiden Bilder zueinander aus Abschnitt 2.2 zu befassen und dieses zu beseitigen. Wie bereits beschrieben ist dies über die Kamera-Kalibrierung (siehe Abschnitt 2.2.3) möglich. Dabei werden die Bilder sowohl entzerrt als auch so ineinander überführt, dass ein akzeptables Arbeitsbild erzeugt wird.

Um verwendbare Farb- und Infrarot-Bilder mit Schachbrettmustern zu erhalten und zu speichern, ist die Verwendung des zweiten Hauptprogramms (Main2.cpp, siehe CD-ROM) möglich. Dieses Prüft bereits bei der Aufnahme automatisch, ob gleichzeitig in beiden Kameras akzeptable Schachbrettmuster zu erkennen sind und speichert diese erst dann automatisch als PNG-Bilder – je Kamera 20 Stück. Ist einmal eine Sequenz aufgenommen kann diese immer wieder verwendet werden. Dabei ist es jedoch wichtig, dass das Schachbrettmuster im Ersten Bild relativ bildfüllend zu sehen ist und sich in den 19 anderen Fällen über den gesamten Fokusbereich der Sensoren verteilt.

Im Rahmen der Realisierung sind im Hauptprogramm für die Kalibrierung die Zeilen 46-58, welche die Parameter bestimmen oder laden, und die Zeilen 143-167 als essentiell zu nennen, welche dann die Entzerrung sowie das Übereinanderlegen der Bilder zum Arbeitsbild beinhalten. Die hierfür benötigte und für jeden Bildsensor als Objekt initialisierte Klasse `class CameraCalibrator` ist in Anhang B.3 dargestellt.

Die Klasse für de Kalibrierung beinhaltet durch ihre Methoden alle Schritte zur Kalibrierung der Kameras. Sie beruht auf den Quellcode-Stücken, welche in [3, S.221ff] beschrieben sind. Angefangen beim Erkennen der Punkte im Schachbrettmuster und Speichern dieser im jeweiligen Objekt der Klasse folgt die eigentliche Kalibrierung mit Hilfe der OpenCV-Funktion `calibrateCamera()`. Das Herzstück stellt jedoch die `remap()`-Methode dar, welche die ermittelten Koeffizienten auf die Bilder anwendet und diese schließlich entzerrt. Zusätz-

lich zu dem vorgegebenen Quellcode wurden hier Methoden zum Setzen und Lesen der Klassenvariablen implementiert, welche das Laden und Speichern dieser in einer externen Datei (`Calibration.yml`, vgl. Abschnitt 4.2.1) ermöglichen.

Das Laden aus dieser Datei erfolgt über die Funktion `f_loadCameraMatrix()` der `functions.h`. Das Ausführen der Methoden der oben beschriebenen Klasse zur Kalibrierung erfolgt hingegen Schritt für Schritt in der Funktion `f_calibration()`. Hierin werden Dateilisten der Kalibrierbilder (jeweils 20 Bilder) erstellt und darin die Schachbrettmuster gesucht sowie deren Eckpunkte bestimmt. Mit diesen Punkten werden dann die Parameter sowohl für das Farbbild, als auch für das Infrarotbild ermittelt. Zum Schluss ist hier außerdem die Bestimmung der Homogenitäts-Matrix für die Projektion des Infrarotbildes in das Farbbild enthalten. Hierzu wird die OpenCV-Funktion `findHomography()` verwendet. Das Resultat ist die Matrix *H*.

Aus unerfindlichen Gründen ist jedoch das Vorzeichen des dritten Matrixelements in der ersten Zeile nicht korrekt und muss durch eine Multiplikation mit -1 in einer zusätzlichen Code-Zeile berichtigt werden.

Abschließend erfolgt in dieser Funktion noch das Speichern der Parameter in der beschriebenen externen Datei, deren Inhalt im nächsten Abschnitt dargestellt ist.

4.2.1 Inhalt der „Calibration.yml“

Die Datei mit dem Titel `Calibration.yml` beinhaltet die aus der Kalibrierung resultierenden Kamera-Matrizen und Verzerrungskoeffizienten, aber auch die Homogenitätsmatrix. Sie dient dazu, dass die Kalibrierung nicht bei jedem Programmstart vollständig durchlaufen werden muss. Auf diesem Weg werden die abgelegten Parameter lediglich geladen und direkt angewendet. Der für diese Arbeit verwendete Inhalt der Datei ist im nachfolgenden Listing aufgeführt:

```
%YAML:1.0
RGB_Cam: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 5.3486939740955017e+02, 0., 3.2170597720408279e+02, 0.,
       5.3434153749315442e+02, 2.5306412094076654e+02, 0., 0., 1. ]
RGB_Dist: !!opencv-matrix
   rows: 1
   cols: 5
   dt: d
   data: [ 2.2783570344427734e-01, -6.2921529134395182e-01,
       1.1852154844134018e-03, 1.7098233251039588e-03,
       4.8250264378767604e-01 ]
IR_Cam: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 6.0357644566746194e+02, 0., 3.2030855627675305e+02, 0.,
       6.0256244643687796e+02, 2.5157668139947108e+02, 0., 0., 1. ]
IR_Dist: !!opencv-matrix
   rows: 1
   cols: 5
```

```
   dt: d
   data: [ -1.2885351809038265e-01, 6.8295062499755854e-01,
       -1.1532738269126241e-03, 5.0901315922130129e-04,
       -1.2244629864382692e+00 ]
H: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 1.0815642556794793e+00, 1.7432169313843893e-02,
       -3.6268408463416378e+01, -2.7359748125407842e-03,
       1.0898334920900015e+00, -2.6566239717448944e+01,
       -6.4390013391368038e-06, 1.7576725548595185e-05, 1. ]
```

Gerundet und übersichtlicher Formatiert ergibt sich daraus:

$$RGB\_CAM = \begin{bmatrix} 534{,}869 & 0 & 321{,}706 \\ 0 & 534{,}342 & 253{,}064 \\ 0 & 0 & 1 \end{bmatrix}$$

$$RGB\_DIST = \begin{bmatrix} 0{,}228 & -0{,}629 & 1{,}185 \cdot 10^{-3} & 1{,}710 \cdot 10^{-3} & 0{,}483 \end{bmatrix}$$

$$IR\_CAM = \begin{bmatrix} 603{,}576 & 0 & 320{,}309 \\ 0 & 602{,}562 & 251{,}577 \\ 0 & 0 & 1 \end{bmatrix}$$

$$IR\_DIST = \begin{bmatrix} -0{,}129 & 0{,}683 & -1{,}153 \cdot 10^{-3} & 5{,}090 \cdot 10^{-4} & -1{,}224 \end{bmatrix}$$

$$H = \begin{bmatrix} 1{,}082 & 0{,}017 & -36{,}268 \\ -2{,}735 \cdot 10^{-3} & 1{,}090 & -26{,}566 \\ -6.439 \cdot 10^{-6} & 17{,}577 \cdot 10^{-6} & 1 \end{bmatrix}$$

Die Datei beinhaltet also die Parameter, wie sie in den Grundlagen (Abschnitt 2.2) verlangt werden.

4.2.2 Kalibrierung mit Hilfe der MATLAB-Toolbox

Die Kalibrierung kann aber auch mit Hilfe der MATLAB-Toolbox [14] für die Kamera Kalibrierung durchgeführt bzw. mit aussagekräftigeren Grafiken untermalt werden. Verwendet man dieselben 20 Bilder wie zuvor beschrieben im Programm, so ergeben sich zusätzlich Vergleichswerte für die im Anwendungsprogramm ermittelten Verzerrungsparameter.

Nach dem Download der Toolbox und Einbindung in MATLAB wird diese mit dem Befehl `calib_gui(0)` gestartet. Es folgen Angaben zum Dateinamen und das Importieren der Bilder. Über den Button „extract grid corners“ müssen im Anschluss auf allen Bildern (hier 20 Stück) die äußeren vier Ecken des Schachbrettmusters händisch definiert werden, sodass am Ende die Kalibrierung erfolgen kann.

Die Ergebnisse für die Kamera-Matrizen und Verzerrungskoeffizienten bewegen sich jeweils in der Größenordnung, wie sie auch zuvor mit Hilfe von OpenCV und dem entwickelten Programm bestimmt wurden. Aus diesem Grund wird eine erneute Aufführung hier nicht vorgenommen.

Über weitere Button der GUI sowie den Befehl `visualize_distorsions` können jedoch die nachfolgen Grafiken erzeugt werden. Sowohl für die Farb- als auch die Infrarot-Kamera durchgeführt ergeben sich somit:

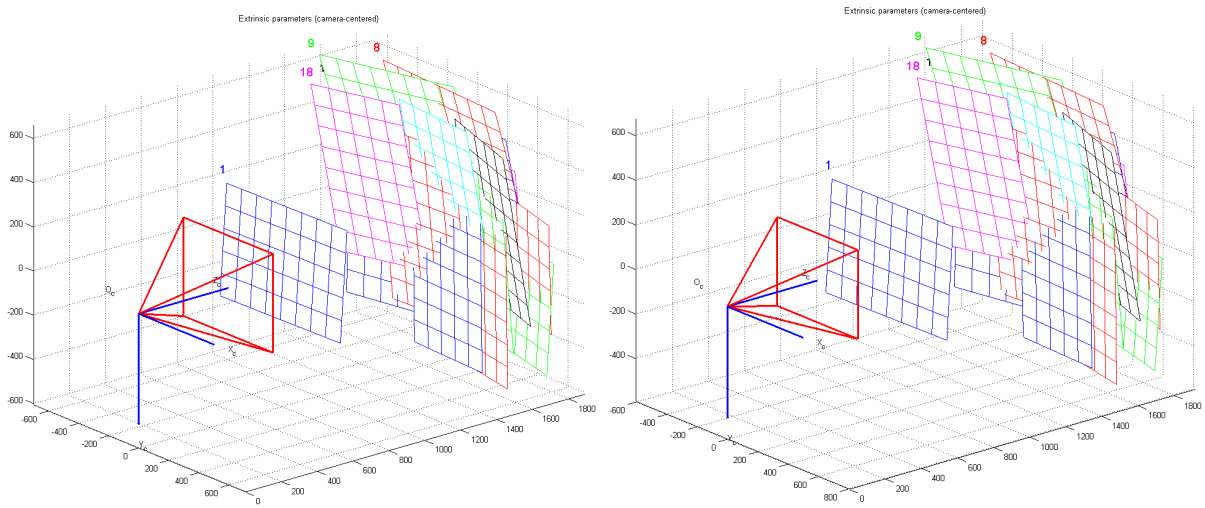

Bild 4.22: Relative Position der Schachbrettmuster in Bezug auf die Kamerazentren (Farbbild links, Infrarotbild rechts)

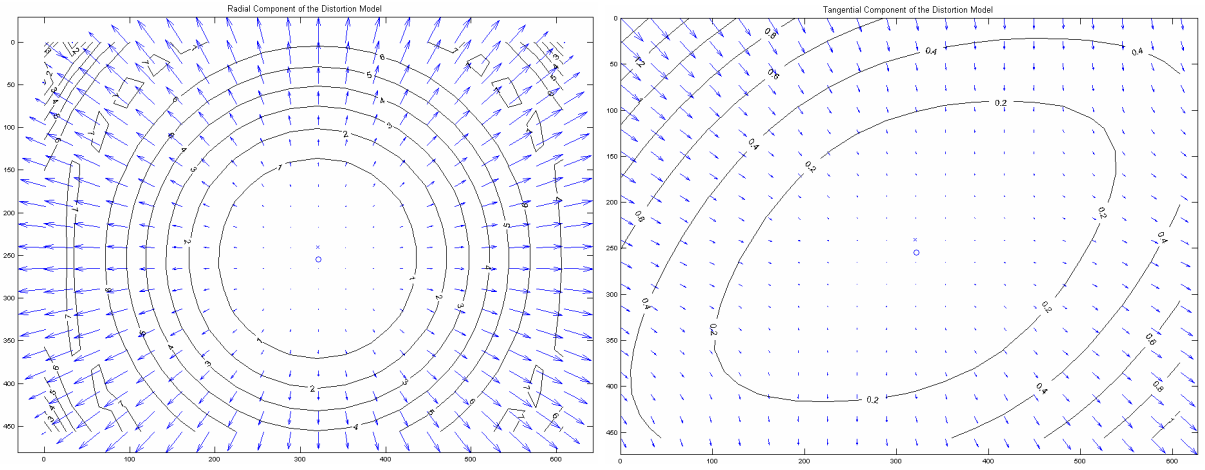

Bild 4.23: Farbsensor-Verzerrungen – radial (links) und tangential (rechts)

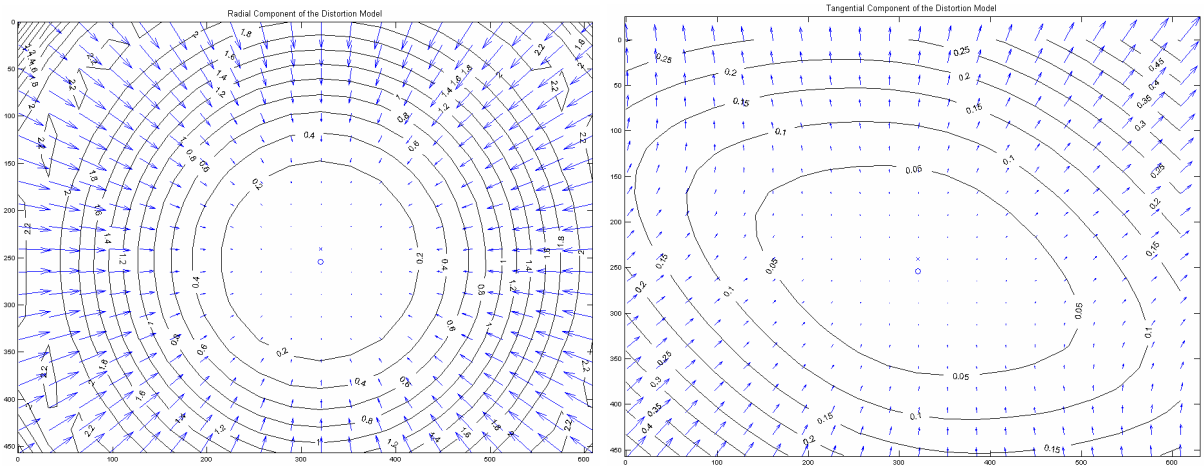

Bild 4.24: Infrarotsensor-Verzerrungen – radial (links) und tangential (rechts)

Bild 4.22 zeigt noch einmal gut die relativen Positionen der einzelnen Schachbrettmuster vor den beiden Bildsensoren der Kinect-Kamera. Bis auf eine bereits erkennbare Verschiebung auf der x-Achse sind die Bilder in soweit korrekter Weise identisch.

Des Weiteren zeigen sowohl Bild 4.23 als auch Bild 4.24, dass beide Kameras Verzeichnungen aufweisen bzw. Verzerrungen in erfassten Sequenzen erzeugen. Dabei sind die radialen, also durch die Linsen hervorgerufenen, Fehler stärker als die tangentialen. Alle Fehler stellen sich jedoch symmetrisch und zirkular zum Bildmittelpunkt dar, was auf Linsen mittlerer Qualität schließen lässt. Interessant dabei ist, dass die Farbkamera radiale Verzeichnungen von bis zu 7 Pixeln vom Mittelpunkt ausgehend nach außen, die Infrarot-Kamera hingegen nur Verzeichnungen von maximal 2,2 Pixel zum Mittelpunkt hin aufweist. Die tangentialen Verzeichnungen sind bei beiden Sensoren in der Größenordnung relativ identisch gering, jedoch in ihrer Richtung um 90° zueinander verdreht.

Nichts desto trotz können diese Fehler durch eine erfolgreiche Entzerrung größtenteils behoben werden Damit ist mit einer weiteren Bildvorverarbeitung die Grundlage für die nachfolgende Erzeugung eines brauchbaren Arbeitsbildes geschaffen.

4.2.3 Erzeugung des Arbeitsbildes

Die Erzeugung des eigentlichen Arbeitsbildes für die nachfolgenden Schritte der Detektion und des Trackings bzw. Zählens erfolgt im Hauptprogramm in den Zeilen 149-167. Bis hierhin sind das Farb- und das Tiefenbild lediglich entsprechend der zuvor beschriebenen Vorgänge entzerrt, jedoch nicht perspektivisch übereinander gelegt.

Diese Projektion erfolgt nun mit der OpenCV-Funktion `warpPerspective()`, welche zur Berechnung lediglich auf die Homogenitätsmatrix *H* zugreift. Dies hat mit einer durchschnittlichen Dauer von 25ms für Entzerrung und Projektion Vorteile in der Performance des Programms.

Ein Vergleich der entzerrten Tiefenbilder vor und nach der Projektion zeigt deutlich den Unterschied:

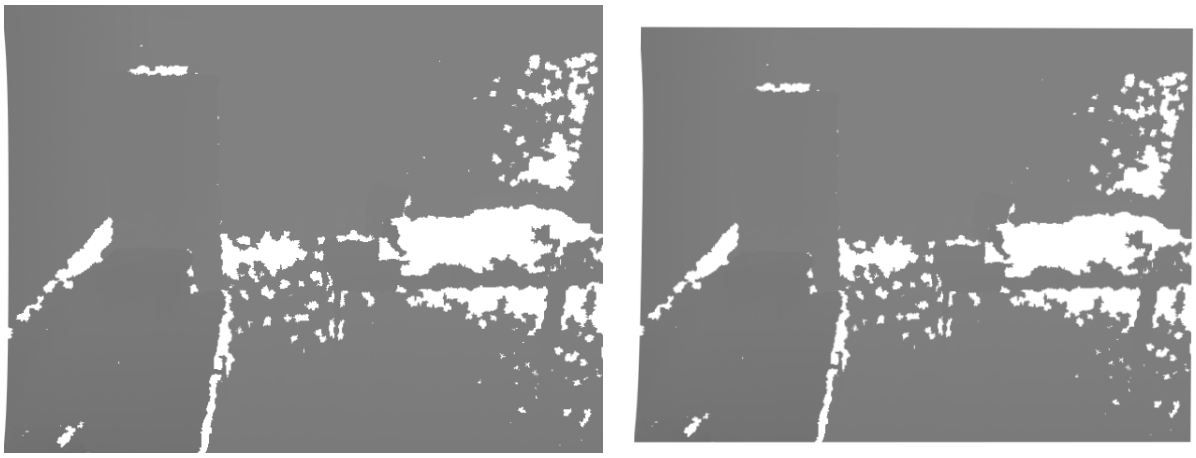

Bild 4.25: Vergleich der Tiefenbilder vor und nach der Projektion

Das projizierte Bild wird zusammengestaucht und um ca. 35 Pixel nach rechts verschoben. Dieses ist mit der größeren Brennweite des Infrarot-Sensors sowie die örtliche Distanz beider Sensoren zueinander begründet.

Für die Erzeugung des Arbeitsbildes wird das so erzeugte Tiefenbild nun binarisiert. Es ergibt sich damit eine Art Schablone für das Farbbild (vgl. Bild 4.26), welche mit ihren weißen Bereichen (d.h. den im Arbeitsbereich befindlichen Objekten) bestimmt, welche Gebiete des Farbbildes weiterhin zur Verfügung stehen sollen. Erreicht wird dies durch das Aufsplitten des Farbbildes in seine einzelnen Farbkanäle (`split()`-Funktion) und anschließender pixelweiser Verknüpfung der Schablone mit den Kanälen. Ist die Schablone Null (schwarz), so bleiben keine Farbinformationen erhalten. Nach Abschluss dieser Manipulation werden die einzelnen Kanäle über die `merge()`-Funktion wieder zu einem Farbbild zusammengesetzt.

Um zu kleine Elemente, d.h. Störungen im Binärbild zu eliminieren, wird dieses mittels morphologischen Operationen noch geöffnet (opening, vgl. Abschnitt 3.2.1.4). Dieses ist hier jedoch nicht symmetrisch, d.h. die Strukturelemente der Erosion und Dilatation sind nicht gleich groß. Das Erodieren erfolgt mit einem rechteckigen 10x10 Element, die darauf folgende Dilatation hingegen mit einem 15x15 Pixel großen Element. Dadurch wird neben der gewünschten Entfernung der Störungen der weiße Objektbereich minimal größer. Dies hat den Vorteil, dass der Schleier um die später detektierten Personen nicht direkt am Körperumriss der Person anhaftet, sondern einen minimallen Spielraum als Toleranz übrig behält.

Die weißen Bereiche sind aber auch durch den Effekt des Infrarotschattens, welcher in Abschnitt 2.1.2.1 beschrieben wurde, bereits einseitig vergrößert. Dies hat jedoch keinen negativen Einfluss.

Bild 4.26: Aus dem Tiefenbild generierte Binärschablone

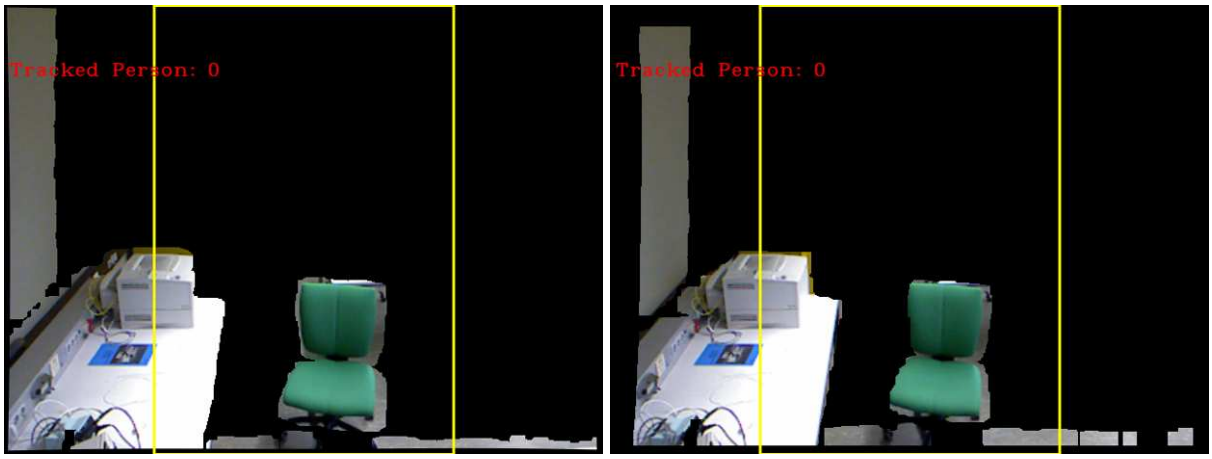

Bild 4.27: Arbeitsbild (vor der Projektion: links, nachher: rechts)

Das nun fertige Arbeitsbild ist in Bild 4.27 (rechts) dargestellt. Als Vergleich dazu ist links noch einmal ein unvorteilhaftes Arbeitsbild ohne die Projektion abgebildet. Hier ist die Überlagerung z.B. an der nicht optimalen Umrandung von Stuhl und Drucker, aber auch an der braunen Steckdosenleiste an der Wand erkennbar. Das rechte Bild weist hier deutlich bessere Ergebnisse auf.

4.3 Detektion von Personen bzw. Gesichtern

Mit dem konstruierten Arbeitsbild kann nun die Detektion der Personen vollzogen werden. Hierfür wird zunächst noch ein Arbeitsbereich in horizontaler Richtung, d.h. der x-Achse, definiert, da normalerweise nicht der gesamte Bildausschnitt benötigt wird. Dieser ist bereits in Bild 4.27 als gelber Rahmen dargestellt. Das Arbeitsbild ist somit nun in der Breite und Tiefe beschränkt. Beide Werte lassen sich durch die Präprozessor-Definitionen `FRAME_ROI` (standardmäßig auf 160 gesetzt) und `DEPTH_THRESHOLD` (bei normalerweise 122) einstellen oder verändern.

Mit Hilfe der Funktion `f_detection()` wird nun in diesem Bereich nach Gesichtern und damit Personen gesucht.

Dies geschieht mit Hilfe des Kaskaden-Klassifikators, welcher in den Zeilen 61-70 des Hauptprogramms erstellt und initialisiert wird. An dieser Stelle werden bereits bestehende Daten, mit denen der Klassifikator trainiert wurde, aus einer für frontal betrachtete Gesichter Datei geladen (`haarcascade_frontalface_default.xml`[13]).

Angewendet wird der Klassifikator über die Methode `detectMultiScale()` jedoch innerhalb der Funktion `f_detection()`, welche in Zeile 178 aufgerufen wird. Die minimale Bildgröße beträgt dabei in Anlehnung an die zur Verfügung stehenden Daten im Klassifikator 24x24 Pixel. Aus Gründen der optimalen Performance wird der Scaling-Faktor auf 1,3 gesetzt und eine Vorfilterung des Arbeitsbildes mittels Canny-Algorithmus zur Kantendetektion durchgeführt. Mit dem überwiegend schwarz verschleierten Hintergrund ist hier eine deutliche Leistungsverbesserung des Programmablaufs erreicht worden. Abschließend sei noch erwähnt, dass ein Gesicht nur als solches detektiert wird, wenn mindestens drei Merkmale des Klassifikators in der Nachbarschaft um einen Punkt erfüllt werden. Dies beugt Fehldetektionen vor, wobei der Wert auch nicht viel weiter erhöht werden sollte, da sonst gar keine Gesichter mehr detektiert werden.

Vor dem Verlassen der Funktion wird jedes so erkannte Gesicht nun noch mit einem roten Kasten markiert und in einem Fenster mit dem Ausschnitt des Arbeitsbereiches ausgegeben.

Diese Art der Detektion ist jedoch nicht auf das Erkennen von Gesichtern beschränkt. Wird der Kaskaden-Klassifikator mit einer anderen Datei und damit anderen Merkmalen geladen, so kann z.B. auch der Oberkörper detektiert werden. Auch hierfür ist zu Testzwecken ein Quellcode-Teil mit entsprechenden Konfigurationen implementiert. Diese Möglichkeit hat sich jedoch aus zweierlei Gründen nicht als optimal erwiesen: Zwar ist die Detektion prinzipiell einfacher und nicht so fehleranfällig, weil die Haltung und Drehung von Kopf und/oder Körper kaum negative Einflüsse haben, dafür ist er aber aufgrund der größeren und komplexeren Bildelemente um ein Vielfaches langsamer in der Berechnung. Außerdem ist die Wahrscheinlichkeit, dass sich zwei Personen gegenseitig mit ihren Oberkörpern verdecken, um

[13] Die Datei ist in den Verzeichnissen von OpenCV unter `/data/haarcascades` zu finden.

einiges Höher als bei der Verwendung von Köpfen. Dennoch bleibt diese Möglichkeit eine sinnvolle Alternative, welche im Rahmen dieser Arbeit jedoch nicht weiter verfolgt wird.

Die auf diese Weise erfassten Personen bzw. Gesichter werden aus der Funktion als `vector` zurückgegeben, wobei hier die 2D-Koordinaten des Gesichtsmittelpunktes gespeichert sind. Auch die Anzahl an insgesamt detektierten Gesichtern ist nun als `num_faces` bekannt.

4.4 Tracking und Zählen von Personen

Das Tracking von Personen wird, wie in Abschnitt 3.3 beschrieben, in dieser Thesis mit Hilfe von Partikelfiltern realisiert. Da das Verfolgen und Zählen der detektierten Personen eng miteinander verbunden sind, wird dieses sowohl in diesem Abschnitt zusammen betrachtet, also auch gemeinsam ich der Funktion `f_tracking()` im Quellcode umgesetzt. Diese wird in Zeile 191 aufgerufen.

Doch zunächst sei an dieser Stelle das komplexe Partikelfilter als eigene Klasse vorgestellt. Weiterhin sei darauf hingewiesen, dass die für das Partikelfilter benötigten Funktionen der OpenCV-Bibliothek leider nicht mehr in den standardmäßig eingebundenen Header-Files enthalten sind, sondern in die Kategorie „legacy“[14] verschoben wurden. Warum das durchaus weit verbreitere und effektive Partikelfilter dorthin ausgelagert wurde, ist jedoch nicht bekannt.

4.4.1 Partikelfilter

Das Partikelfilter ist als `class Partikelfilter` in der `partikelfilter.h` (Anhang B.4) definiert. Diese Klasse besteht im Grunde aus einer Reihe von internen, privaten Variablen, einem Konstruktor/Destruktor-Methodenpaar, weiteren Methoden für den Zugriff auf die Klassenvariablen sowie zwei essentiellen Methoden. Hier ist zum einen die Methode `clear()` zu nennen, welche ein nicht mehr benötigtes Filter korrekt leert, zum anderen die Hauptmethode `filter()`, welche die eigentliche Arbeit verrichtet.

Aufgrund der in Abschnitt 3.3 beschriebenen Tatsache, dass jede Person ein eigenes Filter besitzen soll, welches diese dann tracked und zählt, muss jedes neu erstellte Filter auch individuell initialisiert werden. Daher ist bereits der Konstruktor für Objekte dieser Klasse umfangreich und initialisiert jedes Klassenobjekt umfassend. Prinzipiell geschieht die nachfolgend beschriebene Vorgehensweise in Anlehnung an [3, S.364ff] – alle essentiellen Funktionen für das Filter werden von OpenCV unter dem Stichwort „condensation“ oder „condensation algorithm“ zur Verfügung gestellt.

4.4.1.1 Initialisierung

Der erste Schritt zur Initialisierung des Partikelfilters ist der Funktionsaufruf von `cvCreateConDensation()`. Die Dimension `dim` wird dabei aufgrund des dreidimensionalen Raumes, welcher überwacht werden soll, auf 3 gesetzt. Die Anzahl der Partikel ist mit 500 auf einen ausreichend großen und für die Rechendauer gleichzeitig minimalen Wert gewählt. Gespeichert wird das Filter in einer Variablen vom Datentyp `CvConDensation*`.

[14] „legacy“ (deutsch: veraltet) beschreibt einen Bereich der OpenCV-Bibliothek, in dem nicht mehr empfohlene oder überholte Algorithmen, Funktionen oder Klassen abgelegt sind.

Diese Struktur hat folgende Elemente, welche in weiteren Schritten weiter initialisiert werden müssen.

```
typedef struct CvConDensation
    {
        int MP;     // Messvektor-Dimension
        int DP;     // Zustandsvektor-Dimension
        float* DynamMatr;       // Dynamik-Matrix
        float* State;           // Statusvektor (Zu Beginn: Anfangswert)
        int SamplesNum;
        float** flSamples;      // Positionsvektor der einzelnen Partikel
        float** flNewSamples;
        float* flConfidence;    // Wahrscheinlichkeitsvektor der einzelnen Partikel
        float* flCumulative;
        float* Temp;
        float* RandomSample;
        CvRandState* RandS;     // Strukturarray zur Erzeugung von Zufallsvektoren
    } CvConDensation;
```

Die Dynamik-Matrix ergibt sich als Einheitsmatrix zu:

$$\text{DynamMatr}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{13}$$

Hiermit wird von der ersten Hypothese ausschließlich die erste Dimension, von der zweiten die zweite und von der dritten Zustandsvorhersage ausschließlich die dritte Dimension beeinflusst.

Weiterhin muss dem Filter ein Startwert vorgegeben werden. Dieser ist mit der ersten Position im 3D-Raum, an welcher sich die dazugehörige Person befindet, identisch. Der Vector `state[3]` wird also mit den x, y, und z-Koordinaten initialisiert. An dieser Stelle wird darauf folgend die Partikelwolke des Filters über die Funktion `cvConDensInitSampleSet()` initialisiert. Dies geschieht zirkular um die aktuelle Position im 3D-Raum, jeweils auf 1/5 des möglichen Bereichs begrenzt. Somit wird nicht der gesamte Bildbereich mit Partikeln zufallsbedingt geflutet, sondern nur ein sinnvoller und möglicher Aufenthaltsbereich um den letzten Aufenthaltsort der Person (vgl. Bild 4.28).

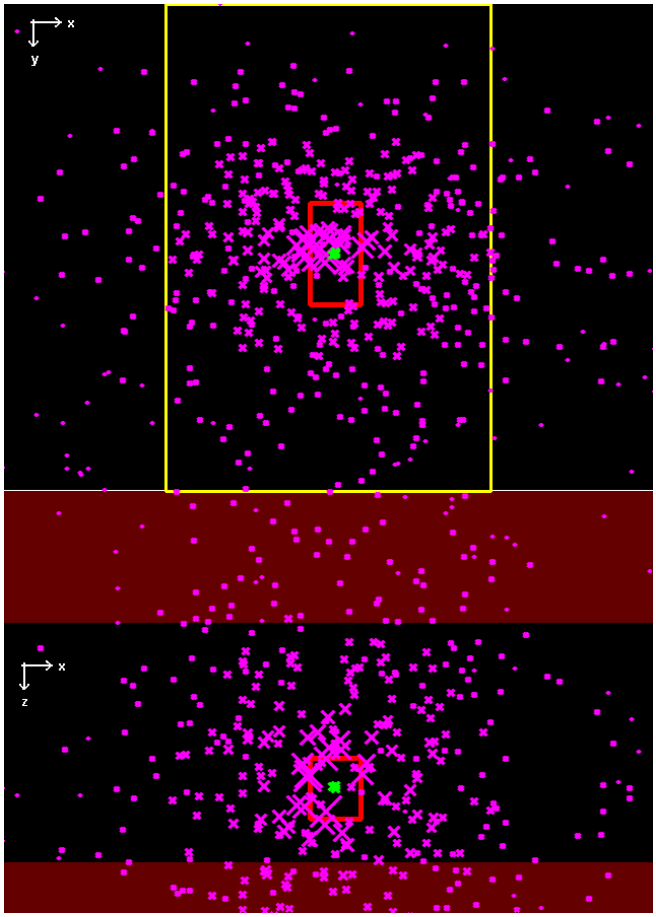

Bild 4.28: Arbeitsbild mit dargestellten Partikeln eines Partikelfilters im 3D-Plot

4.4.1.2 Filterung

Die speziell für diese Anwendung konzipierte Filterung mittels Partikelfilter erfolgt in der Methode `filter()` der übergeordneten Klasse. Diese unterscheidet grundsätzlich zwei Arbeitszustände – aktiv oder inaktiv, d.h. dass Messwerte zur Verfügung stehen oder nicht. Existieren aktuelle Messwerte, so werden diese direkt als erwartete Position (`est_x`, `est_y`, `est_z`) übernommen. Im anderen Fall wird mit einer einfachen Bewegungsvorhersage eine zu erwartende Position berechnet.

Hierzu werden die Koordinaten der letzten drei Standorte im 3D-Raum analysiert und durch die Abstände und Richtungen zueinander die neue Position interpoliert. Es ergibt sich folgende Berechnung, wobei P_1, P_2 und P_3 die drei bekannten Positionen sind:

$$P_{est} = P_3 - d = P_3 - \frac{(P_3 - P_2) + (P_2 - P_1)}{2} \text{ mit } P_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \tag{14}$$

P_{est} repräsentiert den erwarteten neuen Standort, welcher jedoch nur dann berechnet wird, wenn die Änderung von P_3 in einem Bereich von $2 < d < 50$ Pixel liegt. Somit können Fehlberechnungen oder irreguläre Vorhersagen abgefangen werden. Der Interpolationsradius ist hier also auf 50 Pixel begrenzt. Außerdem wird noch verhindert, dass der sichtbare Arbeitsbereich durch die Berechnung verlassen werden kann. Die minimal mögliche Koordinate ist mit P_{MIN}, die maximale mit P_{MAX} festgelegt:

$$P_{MIN} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, P_{MAX} = \begin{pmatrix} 640 \\ 480 \\ 255 \end{pmatrix} \tag{15}$$

Die Messpunkte im Filter werden in der Variablen `measV` vom Datentyp `Point3f` gespeichert, die Kombination aus diesen mit den interpolierten Punkten später unter `particleV`. Doch um diese Vorhersagepunkte zu bestimmen, muss das Filter noch aktualisiert werden. Dies geschieht wie folgt:

Für jedes Partikel wird die Entfernung zur erwarteten Koordinate in eine Pseudo-Wahrscheinlichkeit umgerechnet und auf das entsprechende Partikel gespeichert. Folgende Gleichung ist für die Wahrscheinlichkeitsberechnung zuständig. Sie nutzt dabei einfache, orthogonal geometrische Zusammenhänge:

$$\Omega = \frac{1}{\sqrt{d_x{}^2 + d_y{}^2 + d_z{}^2}} \tag{16}$$

Dort, wo alle Distanzen d_i am kleinsten sind, ist die Wahrscheinlichkeit Ω somit am größten.

Zu guter Letzt muss das Filter wie bereits erwähnt nur noch aktualisiert werden. Dies geschieht über die OpenCV-Funktion `cvConDensUpdateByTime()`. Hierbei wird intern die prognostizierte neue Position als `State` in der Struktur zur Verfügung gestellt. Diese gilt es abschließend noch abzurufen sowie gespeichert werden. In diesem Moment gilt der Filterprozess als beendet. Direkt in der Filtermethode ist außerdem noch die grafische Ausgabe der Mess- und auch der Filterpunkte in dem 3D-Plot implementiert.

4.4.2 Personenverfolgung (Tracking)

Das Tracking der einzelnen Personen erfolgt mit Hilfe der eben vorgestellten Klasse des Partikelfilters in der Funktion `f_tracking()` (Siehe Zeile 191 des Hauptprogramms). Der gesamte Programmablauf hierfür ist in dem nachfolgenden Flussdiagramm dargestellt:

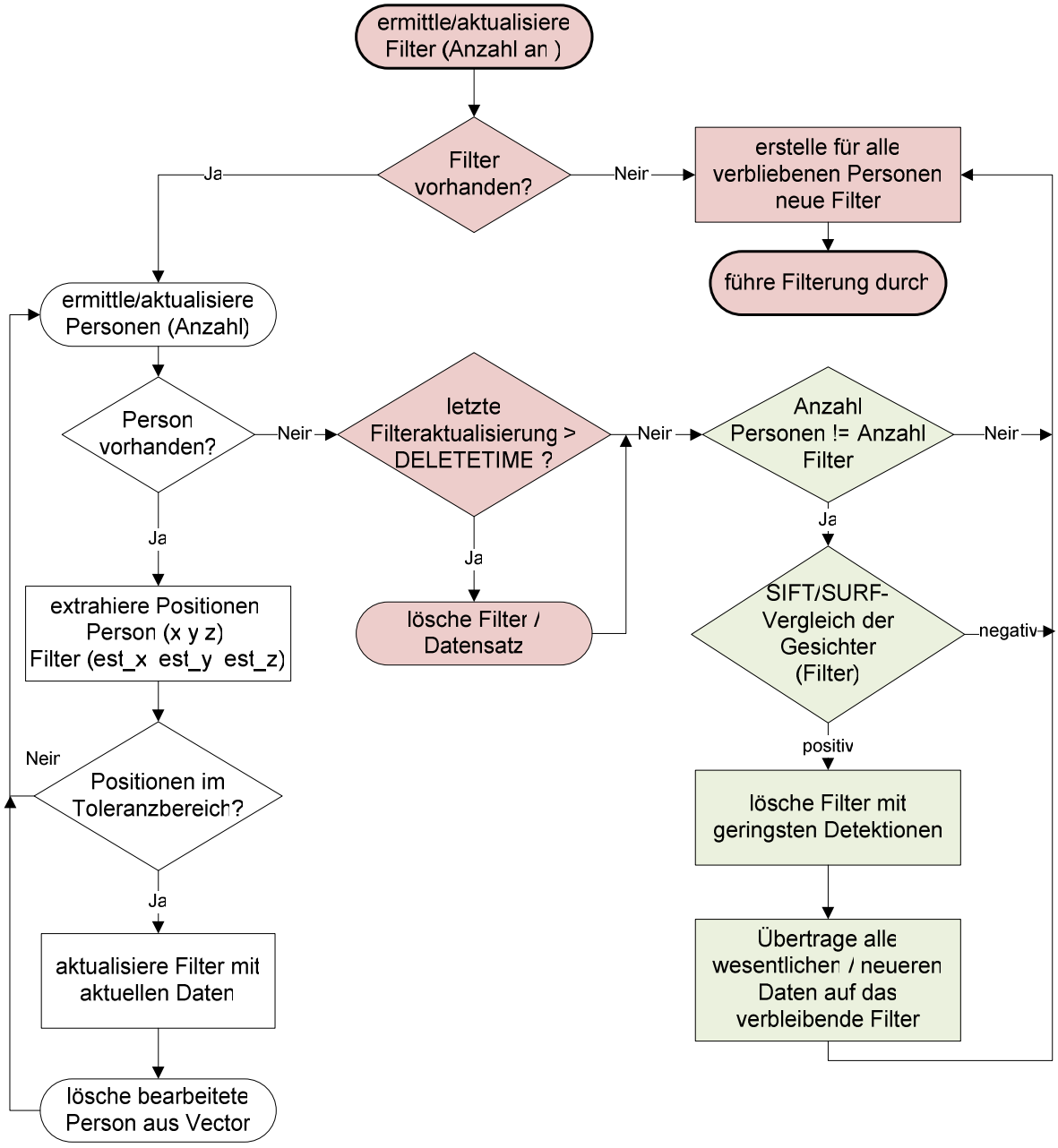

Bild 4.29: Programmablauf der Tracking-Funktion

Den Beginn stellt in jedem Schleifendurchlauf die Ermittlung der Anzahl aller vorhandenen Filter dar. Existieren bereits Partikelfilter, so wird die linke Spalte durchlaufen. Hier werden die personenbezogenen Informationen mit denen der Filter verglichen und bei Übereinstimmung im entsprechenden Filter gesichert. Den Toleranzbereich stellt dabei der Bereich um die erwartete Position dar, welcher für eine positive Übereinstimmung definiert ist. Auf diese Weise erfolgt eine sehr effiziente Zuordnung von Personen zu Filtern allein über die Koordinaten – aufwendige Vergleiche, wie später beschrieben, sind an dieser Stelle nicht notwendig.

Sind alle detektierten Personen verarbeitet worden (erfolgreich oder auch nicht erfolgreich), so wird geprüft, ob im System veraltete oder überflüssige Datensätze vorhanden sind. Mit einer Löschung von nicht mehr benötigten oder fälschlicher Weise erzeugten Filtern, z.B. alle 1000ms (`DELETE_TIME`), wird die Effizienz des Systems aufrechterhalten.

Stimmt die Anzahl an, in diesem Durchlauf detektieren, Personen nicht mit der Anzahl an vorhandenen Filtern überein, so gibt es entweder Personen, für die noch kein Filter existiert oder zu viele Filter für zu wenig Personen. In letzterem Fall kann es auch sein, dass für eine Person zwei Filter existieren, wobei einer aufgrund von Koordinatendifferenzen kurzfristig, aber überflüssiger Weise, erzeugt wurde. Um „verwandte“ Filter, d.h. Filter die zur Identifikation Bilder der gleichen Person gespeichert haben, zu finden und zusammen zu führen, existiert der in Bild 4.29 in grün dargestellte Programmabschnitt. Hier wird mit Hilfe des SURF/SIFT-Algorithmus das eine Filterbild mit dem anderen verglichen. Die genaue Funktionsweise ist in nachfolgendem Unterabschnitt 4.4.2.1 beschrieben. Wird eine Übereinstimmung gefunden, so werden die beiden Datensätze miteinander kombiniert.

Zurück auf der Filterebene werden nun für alle verbleibenden, noch nicht in Filter umgesetzte, Personen eben solche Partikelfilter erzeugt.

Zum Abschluss des Trackings erfolgt für alle vorhandenen Filter die eigentliche Filterung, d.h. der Update der Filterobjekte an sich.

4.4.2.1 SURF/SIFT-Vergleich

Der zuvor erwähnte Vergleich von Gesichtern erfolgt in dieser Arbeit mit Hilfe des SURF/SIFT-Algorithmus. Die theoretischen Grundlagen sind bereits aus 3.2.1.5 bekannt. Wie sich die Realisierung im Detail in der Funktion `f_compareFaces()` gestaltet, ist in den nachfolgenden Absätzen beschrieben.

Grundsätzlich beruht der implementierte Programmteil auf dem Beispiel für „Feature Matching with FLANN“ in [5][15]. Leichte Modifizierungen, aber auch die Erweiterung auf die mögliche Verwendung von SURF anstelle von SIFT, waren für diese Anwendung notwendig.

Mit Hilfe eines `SurfFeatureDetector` werden in den zu vergleichenden Bildern markante Punkte (Referenzpunkte) detektiert. Der weiterhin angewendete `SurfDescriptorExtractor` sowie der `FlannBasedMatcher` sorgen dafür, dass eine bestimmte Anzahl an Übereinstimmungen in beiden Bildern entdeckt wird. Schließlich werden die gefundenen, markanten Punktpaare mit Hilfe von Eingrenzung der maximalen Distanzgröße noch minimiert, sodass für die weitere Betrachtung und Analyse lediglich korrekt zugeordnete Punkte zur Verfügung stehen (good mathces).

Sofern die Anzahl hierfür größer als 4 Referenzpunkte beträgt, erfolgt der weiterführende analytische Vergleich. Über die bereits aus der Kalibrierung bekannten Funktionen

[15] http://docs.opencv.org/doc/tutorials/features2d/feature_flann_matcher/feature_flann_matcher.html

`findHomography()` bzw. `perspectiveTransform()` lassen sich die Eckpunkte des einen auf die des anderen Bildes projizieren. Grundlage hierfür sind die bestimmten Referenzpunkte in beiden Bildern. Bild 4.30 veranschaulicht dies mit einer Grafik:

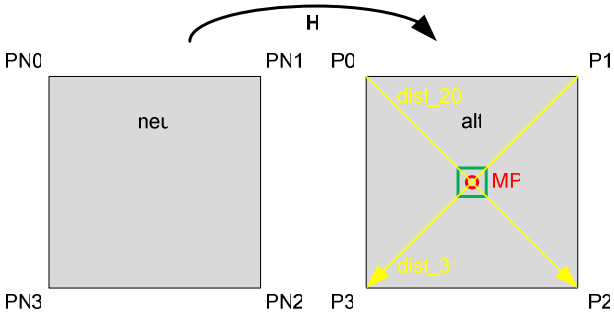

Bild 4.30: Transformation der Eckpunkte

Für die Transformation mit *H* gilt folgende Gleichung. Sie projiziert die Punkte **PN** des neuen Bildes in das alte Bild (Punkte **P**).

$$P = H \cdot PN \tag{17}$$

Die Punkte besitzen jeweils x aber auch y-Koordinaten. Es gilt:

$$P0 = \begin{pmatrix} x0 \\ y0 \end{pmatrix}, P1 = \begin{pmatrix} x1 \\ y1 \end{pmatrix}, P2 = \begin{pmatrix} x2 \\ y2 \end{pmatrix}, P3 = \begin{pmatrix} x3 \\ y3 \end{pmatrix} \tag{18}$$

Der Mittelpunkt **MP** lässt sich nun wie folgt berechnen:

$$MP = P0 + \alpha \cdot (dist\_20), \text{ mit } dist\_20 = P2 - P0 \tag{19}$$

Aus dem folgenden Gleichungssystem lässt sich mit wenigen elementaren Umformungen α bestimmen.

$$\begin{matrix} x: \\ y: \end{matrix} \left\{ \begin{matrix} x0 + \alpha(x2 - x0) = x1 + \beta(x3 - x1) \\ y0 + \alpha(y2 - y0) = y1 + \beta(y3 - y1) \end{matrix} \right\} \tag{20}$$

$$\Rightarrow \alpha = \frac{(x1 - x0)(y3 - y1) + (x3 - x1)(y0 - y1)}{(x2 - x0)(y3 - y1) - (x3 - x1)(y2 - y0)} \tag{21}$$

Was bringt diese Berechnung des Mittelpunktes nun? Die Antwort ist so erstaunlich wie simpel: Sind die beiden Bilder (neu und alt) identisch bzw. weisen sie ähnliche markante Punkte auf, die einander entsprechen, so erzeugt die Perspektivenänderung über *H* kaum bis keine Verzerrungen. Somit werden auch die Eckpunkte des neuen Bildes auf die eigentlichen Eckpunkte des alten Bildes übertragen. Geschieht dies wirklich korrekt, so ist der Bildmittelpunkt identisch mit dem Punkt, wo sich die beiden Diagonalen kreuzen (Punkt **MP**). Wird weiter ein kleiner Toleranzbereich definiert (grünes Quadrat in Bild 4.30), so muss nur noch überprüft werden, ob der Punkt **MP** innerhalb dieses Bereichs liegt. Ist dieses der Fall, so zeigen

die beiden Bilder eine identische oder sehr ähnliche Struktur aufgrund ihrer ermittelten Referenzpunkte.

Zur Sicherheit wird an dieser Stelle noch eine zweite Bedingung eingeführt, welche die korrekte Identifikation verbessern soll. Und zwar müssen die genannten Diagonalen mindestens doppelt so groß sein wie die halbe Länge der Diagonalen des originalen Bildes (alt). So wird sichergestellt, dass die Projektion nicht nur in der Mitte zentriert, sondern auch genügend weit aufgespannt wird.

Befindet sich der Mittelpunkt nun also im Toleranzbereich auf dem alten Bild und weißt genügend große Diagonalen auf, so wird der Vergleich der beiden Bilder als positiv gewertet. Fällt eine der Bedingungen hingegen heraus, so ist der SURF/SIFT-Gesichtsvergleich als negativ zu bewerten. Zwei Beispiele für einen Vergleich ist in Bild 4.31 zu sehen.

Bild 4.31: Ausgabe eines SURF/SIFT-Gesichtsvergleiches (positiv/negativ)

Das linke, grün umrandete, Bild zeigt einen positiven Vergleich, das rechte, rot umrandete, Bild einen negativen, obwohl eine große Menge an markanten Punkten gefunden wurde. Jedoch passen diese nicht zueinander, sodass die Projektion verzerrend gestört wird. Der rote Mittelpunkt befindet sich nicht in der Bildmitte und die Aufspannung erfolgte ebenfalls nicht korrekt.

Mit dieser Art von Visualisierung ist auch der Programmablauf in der `f_compareFaces()` abgeschlossen.

4.4.3 Personenzählung

Die schlussendlich noch offen gebliebene Zählung der Personen, welche den Arbeitsbereich erfolgreich durchquert haben, ist in diesem Abschnitt dargestellt. Auch hier ergibt sich zunächst der im nachfolgenden Bild 4.32 dargestellte Programmablauf als Flussdiagramm.

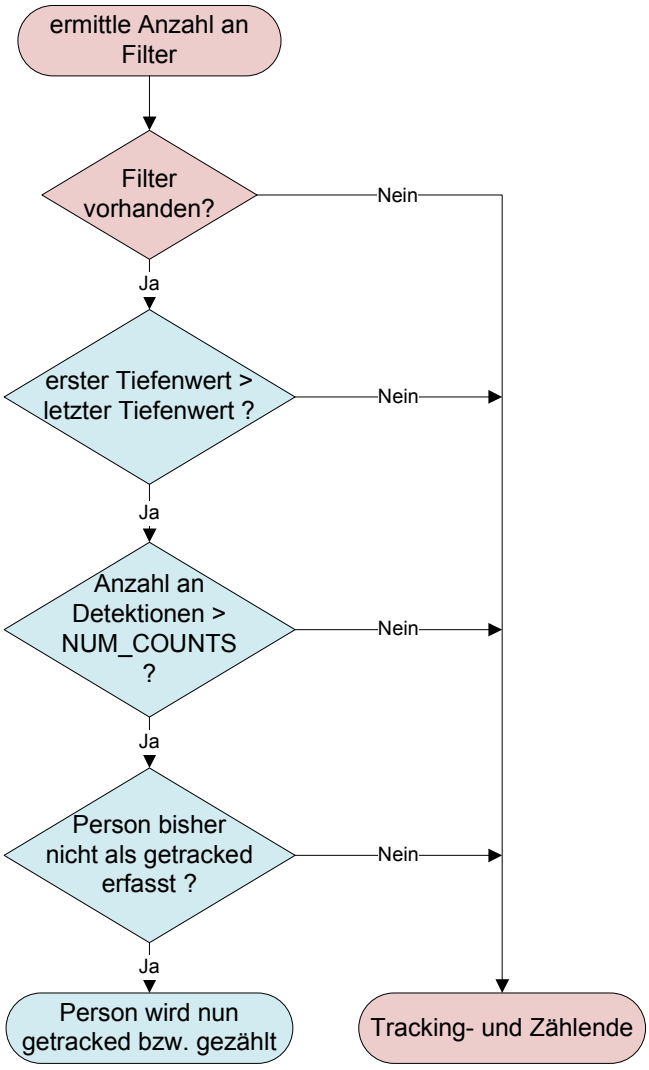

Bild 4.32: Programmablauf der Zähl-Funktion

Die im Bild dargestellten Bedingungen für eine korrekte Zählung einer Person lassen sich wie folgt in Worte fassen: Für jeden vorhandenen Filter und damit gesammelten Datensatz gilt es drei Übereinkünfte zu prüfen.

1) Ist die Person im Verlaufe der Detektionen bzw. während des Trackings auf die Kamera zu gegangen oder hat sie sich rückwärts gehend von ihr entfernt? D.h. ist der erste Tiefenwert größer als der letzte gemessene?

2) Wurde die Person mehrmals detektiert – hier öfter als `NUM_COUNT`?

3) Wurde die Person bisher noch nicht erfolgreich gezählt?

Sind alle diese Bedingungen für einen Filter erfüllt, so wird der programminterne Zähler um Eins erhöht. Für alle weiteren vorhandenen Filter wird diese Abfrage ebenfalls in jedem Schleifendurchlauf gemacht, sodass auf diese Weise die Personenzählung nach der erfolgreichen Detektion und durchgeführtem Tracking ebenfalls als erfolgreich umgesetzt angesehen werden kann.

5 Evaluation und Tests

Kapitel fünf befasst sich abschließend zur Konzeptionierung und Realisierung mit der Evaluation der Entwicklung. Diese beinhaltet verschiedene Tests mit einer einzelnen Person, aber auch das Verhalten mit zwei Personen wird untersucht. Zum Ende dieses Kapitels werden außerdem Tests mit einer größeren Personengruppe (17 bzw. 21 Personen) durchgeführt und analysiert, welche unter realen Bedingungen einen Durchgang entlanggehen.

Ziel ist es, die prinzipielle Funktionsweise des Programms auf Korrektheit zu untersuchen sowie die Zuverlässigkeit zu prüfen. Darauf aufbauend lassen sich Aussagen über den Erfolg der Entwicklung und Realisierung tätigen.

5.1 Vorbereitung

Vorbereitend zur Evaluation ist es notwendig, einen eindeutigen Testaufbau festzulegen. Außerdem wird ein Testkonzept erstellt, welches die einzelnen Tests beschreibt und deren jeweilige Ziele darstellt.

5.1.1 Testaufbau

Der Aufbau für die nachfolgend beschriebenen Tests ist in Bild 5.33 dargestellt. Der Kinect-Sensor ist dabei horizontal, jedoch in einem Neigungswinkel von 20° nach unten geneigt, an einem Stativ in einer Objektiv-Höhe von 220cm angebracht. Für die bessere Portabilität ist der benötigte Computer zusammen mit der Kamerakonstruktion auf einem Rolltisch positioniert.

Als Testumgebung ist für die in Abschnitt 5.1.2 definierten Einzel- und Zwei-Personen-Tests der Raum 06.80 der HAW vorgesehen, für die Mehrpersonen-Durchführungen ein Flur der HAW. Der Flur soll dabei den stark frequentierten Durchgangsbereich repräsentieren, wobei unterhalb der Kamera eine Tür oder ein ähnlicher Durchlass angenommen wird.

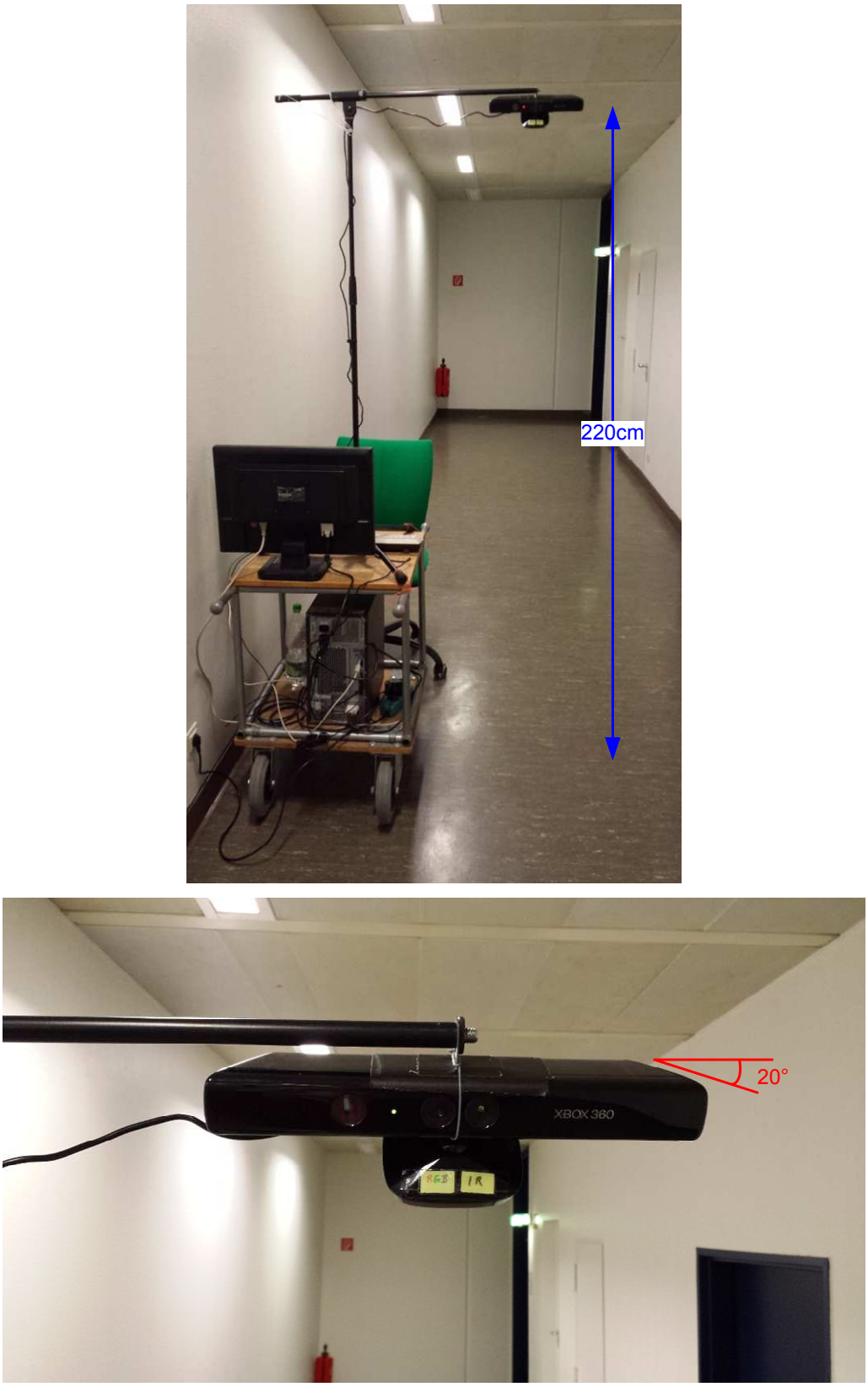

Bild 5.33: Testaufbau

5.1.2 Testkonzept

Die Ergebnisse der Tests sollen vor allem die korrekte Funktionsweise validieren. Hierzu gehören das korrekte Tracking von Personen sowie das erfolgreiche Zählen dieser. Aber auch die Eindeutigkeit in der Erfassung der Personen sowie ihre Unterscheidbarkeit sind von entscheidender Bedeutung.

Neben den bisher genannten Testpunkten gilt es weiter, den Funktionsumfang und Einsatzbereich zu prüfen und somit in Gegenüberstellung mit dem entwickelten Konzept eine optimale Arbeitsumgebung zu definieren.

Nachfolgend sind drei Testkonzepte ausgearbeitet, welche die wesentlichen Prüfungen beinhalten – aufgeteilt nach der Anzahl an betrachteten Personen.

5.1.2.1 Ein-Personen-Test

Mit einer Person vor der Kamera wird die grundsätzliche Funktion des Systems getestet. Hierbei wird vor allem untersucht, ob und wie eindeutig ein Objekt vor dem Sensor als Person detektiert wird. Dabei wird außerdem bestimmt unter welchen Bedingungen diese erkannt wird, wobei hier besonders auf die Haltung des Kopfes eingegangen wird. Des Weiteren soll überprüft werden, welche Einflüsse Mützen, Brillen, Schals oder andere typische Mittel, welche das Gesicht verdecken können, auf die Ergebnisse haben und diese ggf. verfälschen.

Testpunkte sind hierbei:

- Wird die Person eindeutig erkannt / gibt es Fehldetektionen?
- In welchen Bereich darf der Kopf sich drehen / sich neigen / die Person nicken?
- Haben typische Gegenstände oder Objekte im Gesicht Einfluss auf die Detektion?
- Wird die Person korrekt getracked?
- Wie Groß ist die Berechnungszeit im Leerlauf und mit einer detektierten und verfolgten Person?

5.1.2.2 Zwei-Personen-Test

Der Test mit zwei Personen im Arbeitsbereich soll zeigen, ob und in wie Weit gleichzeitig verfolgte Personen vom System separiert werden können. Außerdem kann mit diesem Test die durchschnittliche Rechendauer und damit Latenzzeit bestimmt werden, denn in einem 3m-Bereich vor der Kamera ist in einem Flur von durchschnittlich zwei Personen auszugehen.

Bei diesem Test ergeben sich nachfolgende Punkte, die es zu testen gilt:

- Werden zwei Personen korrekt und eindeutig detektiert?
- Können die Personen von einander unterschieden und separat getracked werden?

5.1.2.3 Mehr-Personen-Test (Test unter Realbedingungen)

Der Mehr-Personen-Test soll den Einsatz des Systems unter Realbedingungen untersuchen und dabei die korrekte Funktionsweise in einem stark frequentieren Durchgangsbereich untersuchen. Hier kommt es vor allem darauf an, dass möglichst viele – idealer Weise alle – der vorübergehenden Personen gezählt werden.

Die für diesen Test wichtigsten Punkte sind:

- Werden alle Personen korrekt und eindeutig detektiert?
- Wie Verhält sich die Leistungsfähigkeit des Systems?

Während des Mehr-Personen-Test werden insgesamt sechs Durchgange durchgeführt, welche die nachfolgenden Parameter beinhalten:

1) Die Personen gehen einzeln, nacheinander im Abstand von ca. 3-4m durch den Arbeitsbereich, wobei der Kopf fest in Laufrichtung fixiert wird. Speicherzeit für die Daten beträgt 3000ms.

2) Die Personen gehen einzeln oder zu zweit nebeneinander, im Abstand von ca. 2-3m durch den Arbeitsbereich, ebenfalls mit fixiertem Kopf und 3000ms Speicherzeit.

3) Wie 2), jedoch mit 1500ms Speicherzeit im Datenmanagement.

4) Die Personen gehen ohne feste Vorgaben durch den Arbeitsbereich. Dabei steht ihnen frei den Kopf ein wenig zu bewegen. Speicherzeit nach wie vor 1500ms.

5) Wie 4), jedoch mit nochmals Reduzierter Speicherzeit auf 1000ms.

---

6) Eine freie Anzahl von Personen bewegt sich völlig ohne Instruktionen durch den Arbeitsbereich bei ebenfalls 1000ms Speicherzeit.

Bei den Tests 1-5 stand eine Testgruppe aus 17 Personen unterschiedlicher Größe (16 Männern, einer Frau) zur Verfügung, Test 6 hingegen beinhaltet eine zufällige Abfolge von 21 Personen (19 Männer, 2 Frauen). Mit den bis hier nun definierten Testkonzepten können die nachfolgenden Testsequenzen weiter analysiert werden.

5.2 Testsequenzen

Die im vorangegangenen Abschnitt definierten Testkonzepte wurden anhand einer Handvoll Sequenzen und Tests schließlich angewendet und geprüft. Entsprechend der Aufteilung in Abschnitt 5.1.2 erfolgt auch die Darstellung und Auswertung dieser in diesem Abschnitt in jeweiligen Unterabschnitten. Bei dem Ein- und Zwei-Personen-Test ist der Arbeitsbereich in der Breite auf 320 Pixel eingestellt. Beim Mehr-Personen-Test auf dem Flur hingegen wird manuell die Breite, welche den Schleier vollständig einschließt bzw. die beiden Wände gerade berührt, verwendet.

5.2.1 Ergebnisse des Ein-Personen-Test

Der Test mit einer Person im Arbeitsbereich soll wie beschrieben zunächst zeigen, ob diese erfolgreich und eindeutig detektiert werden kann. Das nachfolgende Bild 5.34 zeigt das Ergebnis:

Bild 5.34: Gesichtsdetektion einer Person

Die sich im Arbeitsbereich aufhaltende Person tritt gut sichtbar aus dem schwarzen Schleier, welcher durch die Beschränkung des Farbbildes in Tiefe (vgl. 4.2.3) erzeugt wird, heraus und

ist eng von ihm begrenzt. Das Gesicht der Testperson wird korrekt erkannt und markiert. Auch andere Personen (hier nicht dargestellt) konnten erfolgreich detektiert werden. Fehldetektionen kommen hin und wieder vor. Zum Beispiel ist es auch möglich abgedruckte Bilder auf Bekleidung oder handgemalte Smileys als Gesicht zu detektieren. Aber auch bestimmte Strukturen von Falten in Bekleidung können unter Umständen fälschlicher Weise von dem Algorithmus als Gesicht identifiziert werden. Diese Detektionen sind jedoch meist nur von kurzer und damit einmaliger Dauer und werden beim später getesteten Tracking herausgefiltert bzw. ignoriert und nicht gezählt.

Das nachfolgende Bild stellt einen Ausschnitt von Bild 5.34 dar, welcher nun zur näheren Beschreibung und Analyse beitragen soll.

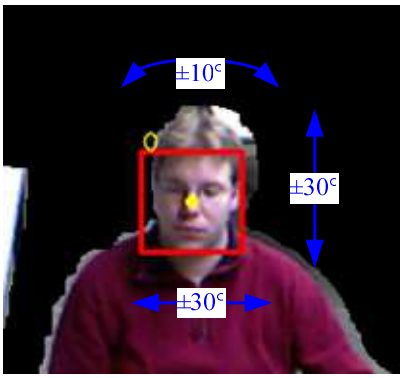

Bild 5.35: Möglicher Bewebungsbereich des Kopfes

Wie in Bild 5.35 dargestellt, ist eine erfolgreiche Detektion trotz einer Drehung des Kopfes um ±30° möglich. Wird der Kopf und damit das Gesicht weiter gedreht, so haben der zu Grunde liegende „Viola and Jones“-Algorithmus und der, auf frontale Gesichter trainierte, Klassifikator nicht mehr genügend Merkmale, welche extrahiert und klassifiziert werden können. Ebenso verhält es sich bei einer Nickbewegung, welcher über ±30° hinausgeht. Kritischer als eine Nick oder Drehbewegung ist die Neigung des Kopfes zur Seite zu bewerten. Wird der Kopf mehr als ±10° geneigt, so ist ebenfalls keine Detektion mehr möglich.

Eine weitere Frage, die es im Verlaufe dieses Tests zu klären gilt, ist, ob Gegenstände oder Objekte im Gesicht einer Person zu Problemen bei der Detektion führen können. Hier ist besonders an Brillen (Sehhilfe oder Sonnenbrille), Bärte und lange Haare zu denken. Wie in Bild 5.35 schwach zu erkennen ist, trägt die Person eine Sehbrille. Somit ist hier kein Problem bei anderen Personen mit Brillen zu erwarten. Auch ein Test mit einer Sonnenbrille offenbarte keine Einschränkungen. Problematischer zeigten sich hingegen Bärte und tief in die Stirn hängende Haare. Bei Personen mit Oberlippenbart scheint der Algorithmus Probleme in der Merkmalsextraktion zu haben, da dieser bekanntlich nach bestimmten Strukturen oder Helligkeitsunterschieden sucht und ein Bart in der Hinsicht nicht verarbeitet werden kann. Vermutlich ist die Ähnlichkeit zu einem (zweiten) Mund zu groß und im Klassifikator nicht vorgesehen. Aber auch eine verdeckte Stirn führt aufgrund damit fehlender Merkmale zu Fehldetektionen. Ein vergleichbarer Effekt wird durch tief sitzende Mützen und Schirmmützen oder bis zur Nase gewickelte Schals hervorgerufen.

Mit der nunmehr erfolgreichen Detektion von Personen gilt es den nächsten Punkt des Testkonzeptes zu überprüfen. Dieser soll klären, ob eine Person vor dem Kinect-Sensor erfolgreich getracked und gezählt werden kann. Hierzu sind nachfolgend Bild 5.36 bis Bild 5.38 dargestellt:

Bild 5.36: Getrackte Person (ohne Bewegung)

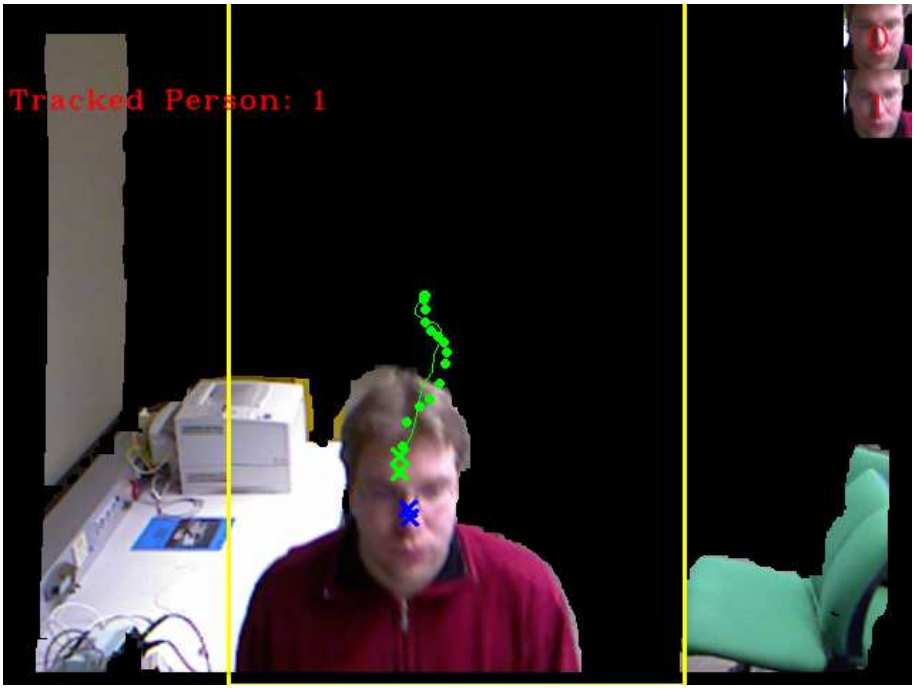

Bild 5.37: Getrackte Person (in Bewegung)

Bild 5.38: Tracking einer Person – SURF-Gesichterabgleich (links positiv, rechts negativ)

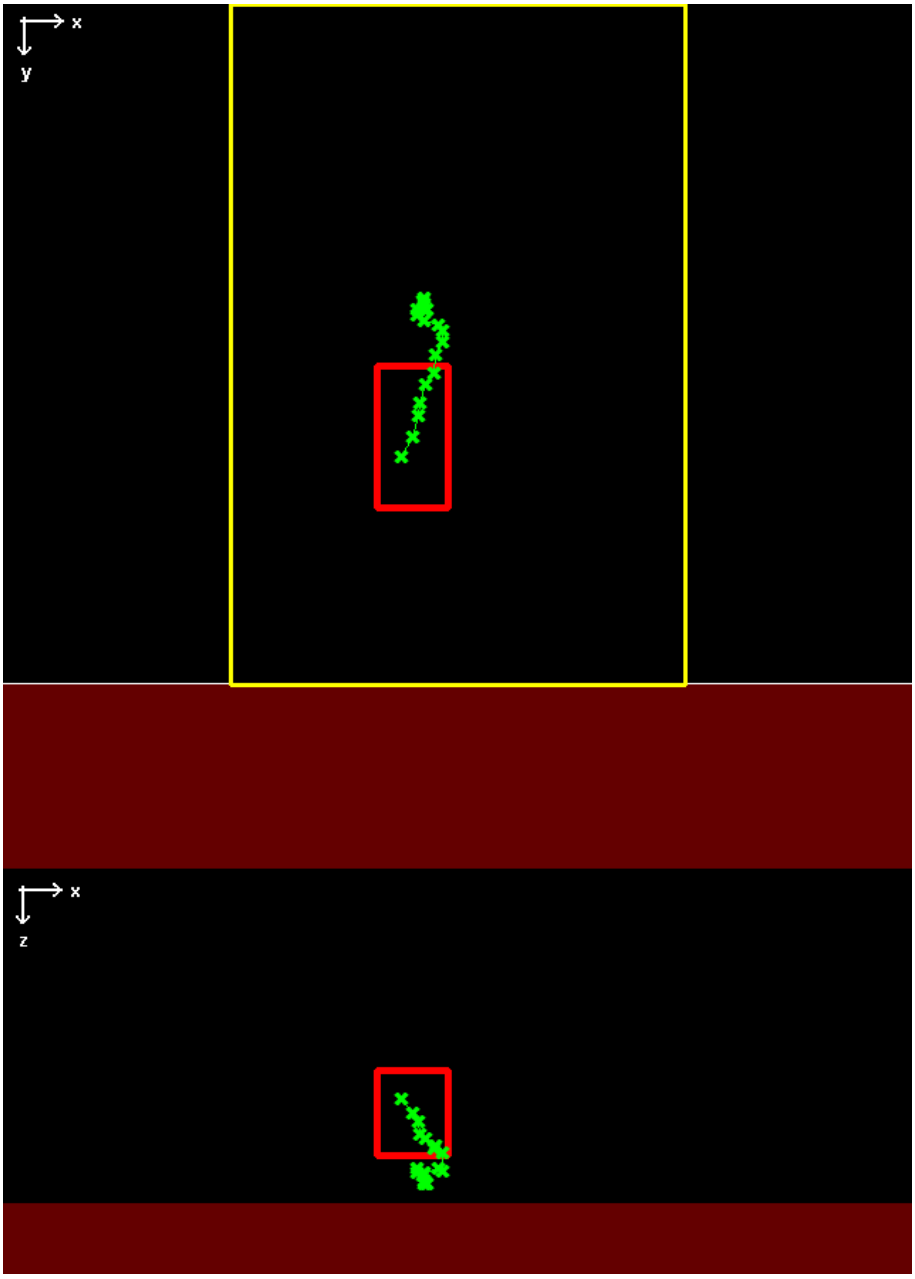

Bild 5.39: Tracking einer Person (3D-Raum)

Zunächst ist in Bild 5.36 das erfolgreiche Tracking und Zählen einer ruhenden Person im Arbeitsbereich dargestellt.

Aber auch eine, sich auf die Kamera zu bewegende, Person wird erfolgreich getracked (siehe Bild 5.37). Aufgrund der zunehmenden Abstände der Messpunkte in y-Richtung entsteht hier am Schluss ein Fehlabgleich der Koordinaten (der blau markierte Messpunkt liegt außerhalb des Toleranzbereichs), sodass eine theoretische zweite Person erkannt wird. Die Person bewegt sich zwar mit angenommener konstanter Geschwindigkeit, in Bezug auf die y-Achse liegt jedoch eine Beschleunigung vor. Nichts desto trotz wird ausschließlich die eigentliche Person mit den grünen Messpunkten und der grünen Tracking-Linie positiv gezählt.
Gleichzeitig erfolgt der Abgleich der beiden zwischengespeicherten Bilder von Filter 0 und 1 mit Hilfe des in Bild 5.38 dargestellten SIFT-Algorithmus. Hier ist zu erkennen, dass dieser im linken Fall ebenfalls positiv ist und somit die Person Nr.1 mit der Person Nr. 0 vereinigt werden wird sodass wieder nurnoch die Eine wirklich vorhandene Person auch im Programm vorliegt. Das rechte Bild zeigt noch mal Beispielhaft einmal einen negativen Abgleich eines Gesichtes. Obwohl die gleiche Person, wenn auch leicht unscharf, abgeglichen wird, weisen beide Teilbilder einen zu großen Haltungsunterschied des Kopfes auf. Dies kann passieren, wenn, wie hier, eine Person zu rasch den Kopf dreht, sodass binnen der später beschriebene Latenzzeit z.B. eine zu große Drehbewegung ausgeführt wird. Wie gut zu sehen ist, kann es passieren, dass dann die linke mit der rechten Gesichtshälfte verglichen wird und es somit zu keiner sinnvollen Übereinstimmung kommen kann.

Die verbleibende Darstellung des durchgeführten Trackings ist in dem letzten Bild der Bildergruppe abgebildet (vgl. Bild 5.39). Der obere Teil zeigt wie gewünscht die Koordinaten im x-y-Bereich des 3D-Raumes, der untere Bereich den x-z-Bereich. Auch hieraus ist also erkennbar, dass eine Person aus dem Mittelpunkt des Bildes nach unten und gleichzeitig auf die Kamera zu gegangen ist.

Die Berechnungszeit lässt sich mit Zeitvariablen im Programmablauf bestimmen. Im Leerlauf, d.h. ohne ein Objekt vor dem Schleier und mit vollständiger Verschleierung des Hintergrundes im Arbeitsbereich, liegt im Mittel eine Berechnungsdauer von **65,45ms** vor. Diese ist jedoch auch von der Breite des Arbeitsrahmens abhängig – je schmaler dieser ist, desto kleiner wird die Zeit und vice versa. Mit einer Person im Testbereich ergibt sich hingegen eine durchschnittliche Latenzzeit von **91,8ms**. Liegt zusätzlich ein SURF-Vergleich vor, kommen für den Zeitraum noch ca. 20ms hinzu.

5.2.2 Ergebnisse des Zwei-Personen-Test

Auch das Testkonzept mit zwei Personen vor dem Kamerasystem gilt es zu testen. Gemäß der Festlegungen in Abschnitt 5.1.2.2 soll hierbei die korrekte Detektion beider Personen sowie das jeweilige Tracking überprüft werden. Bild 5.40 bis Bild 5.43 visualisieren die Ergebnisse dieses Tests:

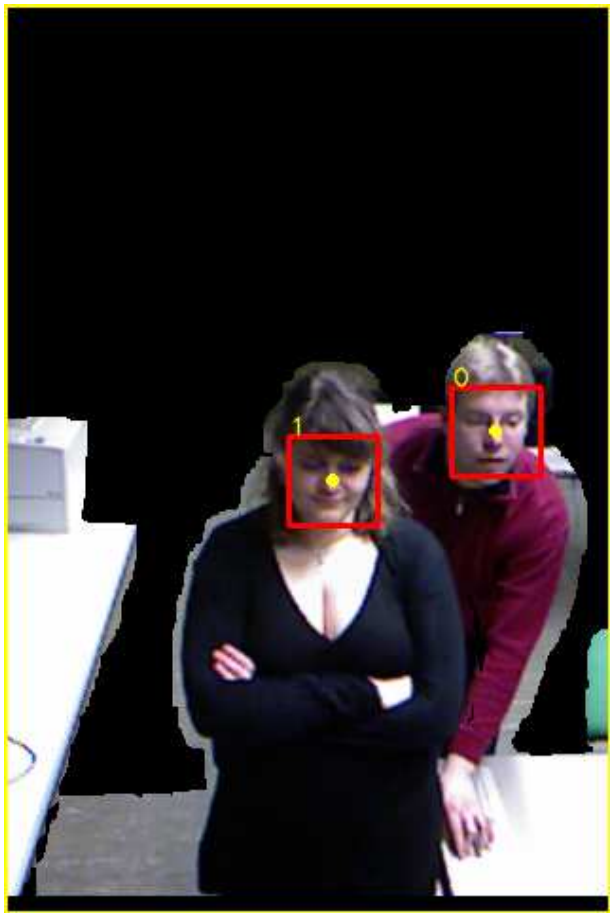

Bild 5.40: Gesichtsdetektion von zwei Personen

Bild 5.41: Zwei getrackte Personen (Person 1 im Hintergrund in Bewegung)

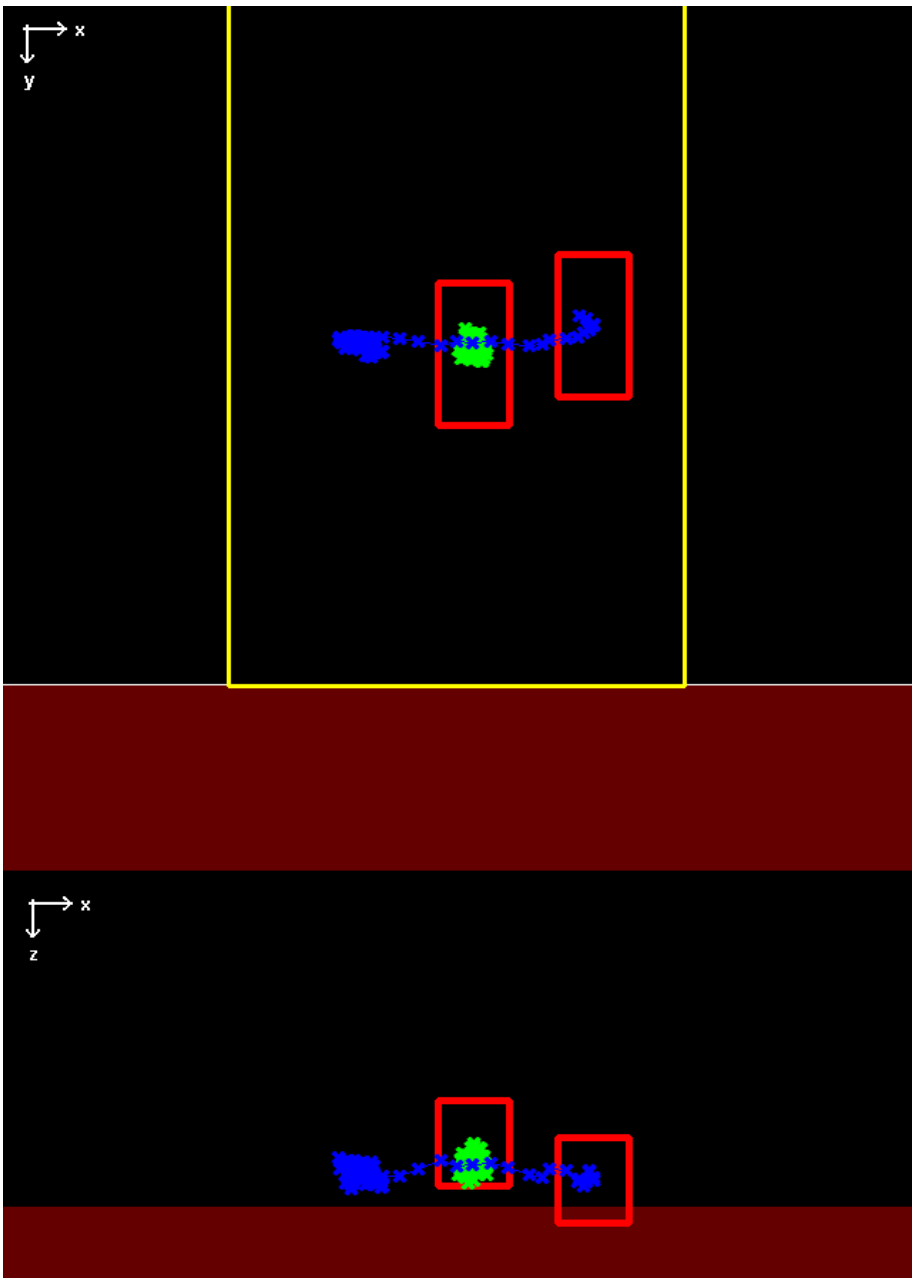

Bild 5.42: Tracking von zwei Personen (3D-Raum)

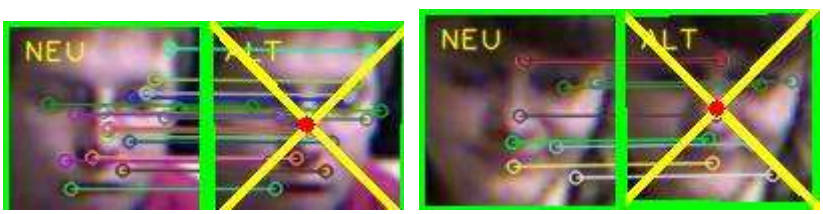

Bild 5.43: Möglicher SURF-Gesichterabgleich beider Personen

In Bild 5.40 ist gut zu erkennen, dass auch zwei Gesichter problemlos von dem Programm detektiert werden. Sie werden getrennt voneinander als Person Nr. 0 und Person Nr.1 markiert.

Das darauf folgende Bild 5.41 zeigt das Ende eines Tracking-Tests mit den Personen, wobei beide korrekt gezählt werden. Allerdings sind die Nummerierungen hier getauscht. Person Nr.0 (grün) steht während des Testdurchlaufs in der Bildmitte, während Person Nr.1 (blau) von links nach rechts hinter Person Nr.0 entlanggeht. Dabei ist es zu dem Zeitpunkt, wo der Kopf hinter dem von Person Nr.0 verschwindet, nicht mehr möglich diesen messtechnisch zu erfassen. Ab diesem Moment greift die Bewegungsvorhersage und zwar so lange, bis der Kopf von Person Nr.1 rechts wieder erscheint und detektiert werden kann. Bis dahin arbeit das Filter von Person Nr.1 im Schätzmodus – im Bild daran zu erkennen, dass nur noch die blaue Linie, d.h. die prognostizierten Positionen von Person Nr.1, im Bereich des Gesichts von Person Nr.0 eingezeichnet wird. Auf der rechten Seite vereinigen sich die Vorhersage und die reell, durch die erneute Detektion gemessenen, Punkte wieder.

Das erfolgreiche und separate Tracking beider Personen ist auch in Bild 5.42 noch mal im 3D-Raum dargestellt. Die zuvor beschriebenen Vorgänge lassen sich ebenso hier wieder finden, auch wenn dieses Mal ausschließlich die prognostizierten Punkte der Filter abgebildet werden. Besonders gut ist hier zu erkennen, dass die Position von Person blau in den Toleranzbereich für die Zuordnung zu Person grün eindringt und diesen durchlaufend auf der anderen Seite ungestört wieder verlässt. Somit ist bewiesen, dass auch sich kreuzende Personen voneinander separiert werden können.
Aus Gründen zusätzlicher Sicherheit wäre es zwar möglich auch hier einen automatisch bei jedem Gesicht durchgeführten Abgleich mit einem anderen Gesicht durch den SURF-Vergleich aktiviert werden kann (vgl. Bild 5.43), jedoch ist dies aufgrund der zuvor getätigten Feststellung nicht zwingend notwendig. Lediglich wenn ein Filter seinen eigenen Toleranzbereich verlässt wird ein zusätzlicher Gesichtsabgleich mittels SURF-Algorithmus aktiviert, um eine wieder korrekte Zuordnung der Koordinaten zu erreichen.

Bei dieser Testsequenz beläuft sich die durchschnittliche Rechendauer auf **103,7ms**, mit dem SURF-Vergleich auf **147,9ms**.

5.2.3 Ergebnisse des Mehr-Personen-Test

Der abschließende Mehr-Personen-Test, welcher Realbedingungen simulieren soll, soll die eigentliche und grundsätzliche, korrekte sowie zeitkonsistente Funktionsweise des Programms durchleuchten. Die wichtigste Prüfung hierbei stellt die Frage dar, ob alle den Arbeitsbereich durchlaufenden Personen auch wirklich gezählt werden. Dabei soll aber die Leistungsfähigkeit des Systems nicht allzu stark leiden und somit eine weiterhin zuverlässige Funktionsweise garantiert werden.

Das nachfolgende Bild zeigt aus Sicht des Kinect-Sensors den Arbeitsbereich des Durchgangsbereiches. Das angezeigte Zählergebnis entspricht dem von Test Nr.5 (siehe Abschnitt 5.1.2.3), als dieser abgeschlossen ist. Die Breite des Arbeitsbereichs beträgt hier 360 Pixel.

Bild 5.44: Arbeitsbereich beim Mehr-Personen-Test

Aus Rücksicht auf die Persönlichkeitsrechte der Testpersonen sind an dieser Stelle keine weiteren Abbildungen eingefügt.

Nachdem die Testpersonen instruiert wurden, haben sie entsprechend der Testvorgaben Nr.1 bis Nr.6 die Versuchsumgebung durchlaufen. Die jeweils genannten Parameter wurden eingestellt. Die Ergebnisse des Zählvorganges sind in nachfolgender Tabelle 5.3 zusammengefasst.

Test Nr.	**1)**	**2)**	**3)**	**4)**	**5)**	**6)**
Anzahl Testpersonen	17	17	17	17	17	21
Anzahl gezählter Personen	17 (100%)	10 (58,9%)	15 (88,2%)	13 (76,4%)	16 (94,1%)	19 (90,5%)
Datenmanagement-Speicherzeit [ms]	3000	3000	1500	1500	1000	1000

Tabelle 5.3: Ergebniszusammenstellung des Mehr-Personen-Test (Realbedingungen)

Der erste Test soll zunächst sicherstellen, dass alle zur Verfügung stehenden Personen auch vom System erkannt werden. Dies konnte als 100% Erfolgreich bestätigt werden. Bei Test Nr.2 hingegen kam es aufgrund der hohen Speicherzeit im Datenmanagement von 3s zum Übersehen einiger vorübergehender Testpersonen. Aus diesem Grund wurde für Test Nr.3 die Speicherzeit halbiert, sodass immerhin 88,2% der Personen wieder erkannt wurden. Steht es den Personen im vierten Test nun frei, den Kopf leicht zu bewegen, so werden wiederum zwei Personen weniger erfasst. Mit einer erneut reduzierten Speicherzeit auf 1s werden jedoch 16 von 17 frei laufenden Personen in Test Nr. 5 erfolgreich gezählt.

Ein ähnlich gutes Ergebnis von über 90% erbrachte auch Test Nr. 6. Das besondere an diesem Test ist, dass die Personen nicht instruiert waren und absolut unbefangen den Testbereich durchlaufen konnten.

Als zeitkritisch stellten sich wie gesagt lediglich die Tests zwei bis vier heraus. Hier waren Latenzzeiten von bis zu 500ms Schuld daran, dass Personen im nun zu langsam laufenden Programm übersehen wurden. Begründen lässt sich dies damit, dass bei hohen Speicherdauern im Quadrat mehr SURF-Vergleiche durchgeführt werden müssen, wenn sich die Koordinaten der Filter zwischenzeitlich den Toleranzbereich verlassen. Dies bremst natürlich den Programmablauf ungemein. Mit den schließlich verwendeten Speicherzeiten von 1000ms relativiert sich dieses Problem größtenteils, sodass bei Test Nr. 5 und 6 Rechendauern von durchschnittlich wieder **100-120ms** erreicht werden.

5.3 Auswertung und Fazit

Nach dem erfolgreichen Abschluss der Tests gilt es nun die Ergebnisse genauer zu betrachten und zu bewerten. Dem endgültigen Fazit vorgreifend lässt sich bereits an dieser Stelle sagen, dass das entwickelte System in den Testsequenzen gezeigt hat, dass es funktioniert und die Aufgabenstellung korrekt erfüllt. Einschränkungen zu dieser Aussage sind unumgänglich, da die Erfolgssicherheit von einigen Parametern abhängt. Doch dies soll Teil der nachfolgenden Analyse sein.

Der Ein-Personen-Test sollte zeigen, dass die grundlegenden Funktionen wie die Detektion, das Tracking sowie das Zählen einer Person zum Erfolg führt. Aber auch Grenzen bzw. Bedingungen, welche zu Fehlern im Programmablauf führen, galt es hierbei herauszufinden. Sofern es dem Kaskadendetektor möglich ist eine Person bzw. deren Gesicht relativ frontal zu erfassen, ergibt sich die Detektion als positiv. Leichte Veränderungen der Kopfposition sind dabei zu verkraften, jedoch nur bis zu einem gewissen Maß. Die Grenze liegt hier zwischen ±10° bis ±30° Neigung, Drehung oder Nicken gegenüber der Normalposition. Bewegt sich eine Person in einem Flur auf eine Tür zu, welche sie durchqueren möchte, ist die Wahrscheinlichkeit hoch, dass der Durchgang direkt visuell fixiert wird. Damit richtet die Person ihr Gesicht automatisch auch in Richtung der Kinect aus, sodass die Detektion zu einem hohen prozentualen Anteil erfolgreich sein wird.

Aber auch Fehldetektionen sind möglich. Wie beschrieben klassifizieren die Algorithmen zum Teil auftretende Merkmale, die Gesichtern offenbar ähnlich sind, kurzzeitig als solches. Am Häufigsten wurden Falten in Bekleidungsstücken aufgrund ihrer Struktur als falsch positiv erkannt. Da dieses aber nicht verhindert werden kann, sind Mechanismen in das Anwendungsprogramm implementiert, welche diese Entgleisung herausfiltern und verwerfen. Mit einem Schwellwert der, für die erfolgreiche Zählung, notwendigen Anzahl an Detektionen wird dieses Problem zuverlässig beseitigt. Sofern eine Fehldetektion nicht mehrfach hintereinander, sondern eher einzeln und über größere Zeiträume hinweg, auftritt stellt sie keine Gefahr dar. Während den Tests gesammelte Erfahrungswerte haben dies als vernachlässigbare Unsicherheit bestätigt.

Problematischer stellt sich die nicht-Detektion von Gesichtern dar. Können Gesichter von Personen aufgrund verschiedener möglicher Ursachen, nicht oft genug oder gar nicht erfolgreich detektiert werden, ist logischer Weise auch keine Zählung dieser möglich. Wissen Personen von der Funktionsweise des in dieser Arbeit entwickelten Tracking-Verfahrens, so ist es ihnen auch immer möglich eine erfolgreiche Zählung z.B. durch ununterbrochenes nach Unten schauen zu verhindern. Unter Normalbedingungen wird jedoch angenommen, dass dies nicht der Fall ist.

Auch das Tracking konnte in seiner prinzipiellen Funktion verifiziert werden. Probleme traten in den betrachteten Testsequenzen lediglich dann auf, wenn die Personen nach unten aus dem Arbeitsbereich heraustreten. Die im Abschnitt 3.3 prognostizierte Problematik der quasi beschleunigten Bewegung in y-Richtung lässt sich also bestätigen. Dramatische Auswirkungen hat dieses Phänomen jedoch ebenfalls nicht auf die Entwicklung. Ähnlich wie zuvor kann die dabei entstehende Fehldetektion und überflüssige Erzeugung eines zweiten Partikelfilters über die wieder nicht erreichte Schwelle für die erfolgreiche Zählung verhindert werden. Dies bestätigen auch die Tests. Sofern die Datensätze zusätzlich nach einer möglichst geringen, aber ausreichend großen Zeit gelöscht werden, können auch mögliche Folgedetektionen verhindert werden.

Dass die Dauer der Aufrechterhaltung von Datensätzen einen erheblichen Einfluss auf die reibungslose Funktion des Systems hat, ist besonders aus den Testergebnissen des Mehr-Personen-Tests ersichtlich. Durch die Reduzierung der Speicherdauer von 3 Sekunden auf eine Sekunde hat sich die Effizienz des Systems fast verdoppelt, sodass am Ende Erfolgsquoten von 90 bis 95% dauerhaft möglich sind. Obwohl eine Person mit normaler Gehgeschwindigkeit circa 3 Sekunden zum vollständigen Durchqueren des Arbeitsbereiches benötigt, dabei aber im Schnitt mindestens zu 75% der Zeit regelmäßig detektiert wird, ist eine Löschung der Datensätze bei mehr als einer Sekunde Inaktivität unbedenklich und mit Blick auf die Leistungsfähigkeit mehr als sinnvoll. Eine Analyse der Belastungsgrenzen erfolgt im späteren Verlauf dieser Auswertung.

Die zweite Testsequenz hat schließlich gezeigt, dass sich auch gleichzeitig getrackte Personen nicht gegenseitig beeinflussen. Auch der unumgängliche, kurzzeitige Verlust an Messwerten durch eine Überschneidung von Gesichtern wird durch die Fähigkeit des Schätzens im Partikelfilter kompensiert. Sofern sich die geschätzten und real wieder gemessenen Positionen nach dem Verlust wieder vereinigen gibt es keinerlei Probleme. Sollte eine Schätzung einmal außerhalb des definierten Toleranzbereiches liegen, so wird einfach ein neues Filter angelegt. Im nächsten Durchlauf erfolgt dann ein Vergleich der Gesichter untereinander mittels der beschriebenen SURF/SIFT-Algorithmen und die Person wird erfahrungsgemäß mit einer nahezu 100&igen Wahrscheinlichkeit wieder gefunden.

Berücksichtigt man schließlich, dass die Kinect mit einer Bildwiederholzahl von 30fps durchschnittlich alle 33ms ein Bild zur Verfügung stellt, so wird mit den schlussendlich verwendeten Parametern ein Verhältnis von 1:2 an genutzten Bildern errecht. Dies bedeutet, dass bei normalem Betrieb von rund 100ms Rechendauer eines von drei möglichen Bildern verwendet und die anderen beiden verworfen werden. Von Echtzeitfähigkeit im engeren Sinne kann hier zwar nicht gesprochen werden, jedoch arbeitet das System für die gewünschte Funktion effizient und sicher genug. Theoretisch wäre es also möglich einen der anderen in Tabelle 4.2

vorgestellten Auflösungsmodi zu verwenden. Mit 10fps würden sich die Zeiten ausgleichen und der Anwender kann auf ein fast doppelt so großes Farbbild zurückgreifen. Jedoch ändert sich die Tiefenauflösung nicht, sodass sich diese Möglichkeit wieder relativiert.

Ein zwischenzeitlicher Test eines anderen Training-Files für den Kaskaden-Klassifikator, welcher hier nicht weiter beschrieben wird, erbrachte noch mal eine kleine Verbesserung der Performance und Detektionssicherheit, ergab jedoch auch einen erheblichen Nachteil. Mit Hilfe des Prinzips des „local binary pattern“[16] (lbp) verringert sich noch mal die Rechenzeit für die Gesichtsdetektion und erbringt zuverlässigere Resultate. Allerdings zeigte sich, dass hier die Haltung des Kopfes einen noch erheblicheren Einfluss auf den Detektionserfolg hat.

Ebenfalls wurde die Detektion von Oberkörpern in Auszügen getestet, jedoch zeigten sich hier zwei wesentliche, negative Effekte. Zum einen hat sich die Vermutung, dass die Detektion durch höhere Überlagerungswahrscheinlichkeit von Personen untereinander verschlechtert hat, bestätigt, andererseits hat sich die Berechnungsdauer aufgrund der größeren Bildausschnitte erheblich vergrößert. Prinzipiell stellt dieses Verfahren auch eine Möglichkeit zur Identifikation und Zählung von Personen dar, wird jedoch im Rahmen dieser Thesis aufgrund der negativen Eigenschaften nicht weiter verfolgt.

Abschließend noch einmal auf den Punkt gebracht lässt sich sagen, dass die Ergebnisse der Tests die Funktionsweise wirklich als korrekt bestätigt und gleichzeitig akzeptable Randbedingungen zur Erfüllung der Aufgabenstellung definiert haben.

[16] Das local binary pattern stellt eine weitere Möglichkeit der Merkmalsextraktion zur Klassifikation dar. Es hat seine Stärken in der Strukturerkennung und nutzt zirkulare Nachbarschaftsbedingungen von Pixeln um Merkmale zu bestimmen.

6 Zusammenfassung und Ausblick

Am Ende dieser Masterarbeit werden Ziele reflektiert und Errungenschaften oder wesentliche Erkenntnisse zusammenfassend dargestellt. Auch ein Ausblick auf mögliche Erweiterungen und Optimierungen ist Teil dieses abschließenden Kapitels.

Grundsätzlich lässt sich sagen, dass mit dieser Entwicklung ein mögliches und auch Erfolg versprechendes Verfahren zur Personenverfolgung und -zählung geschaffen wurde, sofern das Gesicht einer Person identifiziert bzw. detektiert werden kann. Ist dies nicht der Fall, so gibt es keine Möglichkeit einer erfolgreichen Zählung. Doch dies war ein Stück weit das Ziel der Aufgabenstellung zu Beginn dieser Thesis. Schließlich galt es sicher zu stellen, dass ausschließlich Personen und keine anderen Objekte erfasst, verfolgt und gezählt werden.

Mit Blick auf die Problemstellung bestätigt sich somit, dass die Ziele dieser Arbeit durchaus als erfüllt angesehen werden können. Die Anforderungen an die Umsetzung sind dabei zunächst theoretisch abgeschlossen und schließlich in einer Software realisiert worden. Über die Festlegung der Aufgabenstellung hinausgehend ist der Einsatz des Systems in einem stark frequentierten Durchgangsbereich unter realen Bedingungen ebenfalls erfolgreich bestätigt worden. Die Zählung von mindestens zwei Personen wurde also überboten.

Ohne weiteres wäre dies nicht möglich gewesen, wenn nicht die Treiber zur Ansteuerung der Kinect frei zur Verfügung gestanden hätten. Ebenso verhält es sich mit der überaus mächtigen Bibliothek für die Bildverarbeitung, die OpenCV- Bibliothek. Ohne den Kinect-Sensor, welcher den Ausgangspunkt für diese Arbeit und Anwendung von Verfahren bildet, wäre vor allem keine kostengünstige Methode zur Personenverfolgung und -zählung in dieser Art und Weise möglich. Abstriche ergeben sich hieraus in der Dokumentation dieser Mittel, weshalb die Analyse und korrekte Anwendung der Funktionen zu Beginn eine aufwendige und akribische Recherche zur Folge hatte.

Durch den Verlauf der Arbeit hat sich vor allem die Tatsache, dass die Kalibrierung der beiden Kinect-Bildsensoren für das Farb- und das Infrarot- bzw. Tiefenbild, als unabdingbar gezeigt. Ohne die, auf Pixelebene korrekte, Projektion des Bildes mit den Entfernungsinformationen auf das Farbbild wären die Erfolge nicht möglich gewesen. Sowohl für die einwandfreie Entfernungsbestimmung als auch für die Erzeugung eines günstigen Arbeitsbildes ist diese als essentiell anzusehen.

Mit den somit zur Verfügung stehenden Bilddaten war es mittels einfacher, aber wirkungsvoller Algorithmen und Verfahren möglich Gesichter als das Merkmal für Personen zu detektieren. Das Verfolgen von Personen mittels gängiger Tracking-Verfahren stellte die zweite Hauptaufgabe dar, welche mit Hilfe von Partikelfiltern ebenfalls erfolgreich implementiert wurde. Die obligatorische Vorstellung dieser Verfahren auf theoretischer Basis war dabei für das Verständnis unentbehrlich.

Trotz aller Erfolge bei der Umsetzung der Aufgabenstellung wurden, besonders über die durchgeführten Tests, Probleme und Grenzen aufgedeckt bzw. Bedingungen für den möglichst einwand- und fehlerfreien Betrieb formuliert. Erste Grenzen ergeben sich bereits in der, durch die Kinect bedingte, Einschränkung der eindeutigen Ermittlung von Tiefeninformationen. Eine sinnvolle Detektion von Personen, welche weiter als 3,5 Meter vom Sensor entfernt sind, zeigt sich als schwierig. Gleichzeitig dürfen die zu detektierenden Gesichter nicht kleiner als 24x24 Pixel sein, was wiederum eine Grenze in der Auflösung der Kinect-Bildsensoren repräsentiert. Die größte Unsicherheit stellt jedoch das Problem der inkonsistenten Rechenzeit dar. Dem Entsprechend wie viele Personen gleichzeitig verarbeitet werden sollen, ändert sich die Dauer in den einzelnen Berechnungsschritten. Für die Stabilität eines Systems ist diese Tatsache im Prinzip ein k.o.-Kriterium, jedoch ist die Problematik im Rahmen dieser Entwicklung durch zusätzliche Vorgänge in den Griff zu bekommen.

Von den schließlich sehr guten Erfolgsquoten von weit über 90% und der viel versprechenden ersten Lösung der Problemstellung der Personenverfolgung und –zählung mit Hilfe des Kinect 3D-Bildsensors konnte zu Beginn der Arbeit nicht ausgegangen werden. Vor allem die normalerweise starke Beeinflussbarkeit von bildverarbeitenden Lösungen durch äußere Einflüsse stellt in Anwendungen meist das größte Problem dar. Hier konnte die Abhängigkeit von der Umgebung durch die analysierten Verfahren minimiert werden, sofern die Gesichter der Personen denn in irgendeiner Weise belichtet werden. Nur die Dunkelheit stellt neben extremen Haltungen oder Verdeckungen des Kopfes ein allgemeines Ausschlusskriterium dar.

Zum Schluss dieser Masterthesis folgt nun noch eine kurze Evaluation für Weiterentwicklungsmöglichkeiten des insgesamt jedoch durchaus ausgereiften Konzepts.

6.1 Ausblick

Das größte Potential für zukünftige Weiterentwicklungen stellt die Extraktion weiterer Merkmale zur Detektion von Personen dar. Neben erfolgreich erkannten Gesichtern sowie der Position im Raum sind mit Sicherheit eine Vielzahl weiterer Merkmale denkbar. Da solche Überlegungen den Rahmen dieser Arbeit gesprengt hätten sind sie möglichen Folgearbeiten überlassen.

Weiterhin stellt die Optimierung des Partikelfilters eine Verbesserung der Tracking-Eigenschaften dar, welche nicht ignoriert werden sollte. Zusammen mit dem Ausbau der Leistungsfähigkeit, z.B. durch die Verwendung von Threads, lassen sich hier Reserven schaffen.

Neben Möglichkeiten zur Optimierung sind auch Erweiterungen im Bereich der Aufgabenstellung denkbar. Zum Beispiel könnten zukünftig Untersuchungen zu Verfahren erfolgen, welche auch das Verlassen von Räumen bzw. ein umgekehrtes entlang gehen des Durchgangsbereiches, mit derselben Kinect-Kamera gestatten. Auch die Verwendung von Kinect-typischen Skelettierungs- oder Gestenerkennungsverfahren sind als Ausbaustufen vorstellbar, sofern sie vom Kameratreiber unterstützt oder deren Prinzip zur Verfügung gestellt werden. Vorstellbar wäre etwa der Einsatz an einer Flugzeugtür, durch die die Passagiere ein und auch wieder aussteigen. Neben nicht weiter betrachteten Identifikationsmöglichkeiten von Perso-

nen über Bildausschnitte könnte das hier entwickelte System die Sicherheit an Bord dadurch erhöhen, dass der Crew bekannt ist wie viele Personen das Flugzeug betreten und wieder verlassen haben. Ein Abgleich mit der Passagierliste würde sich um ein Vielfaches minimieren und nicht mehr notwendiges, manuelles Zählen erleichtert die Arbeit.

Es ergeben sich, auf der Basis der mit dieser Arbeit gesammelten Erfahrungen und Erfolge, mit Sicherheit eine Vielzahl weiterer Anwendungsbereiche. Abgesehen von überwachenden Aufgaben lässt sich die Kinect z.B. auch zur Steuerung von Maschinen oder Robotern einsetzen. Hierfür ist sie schließlich an der Xbox entwickelt worden.

An diese Stelle soll diese Masterarbeit jedoch mit den oben getätigten Zusammenfassungen und Denkanstößen sowie der funktionierenden Entwicklung abgeschlossen werden.

Literaturverzeichnis

[1] St. Jean, Jared: *Kinect Hacks*: O'Reilly Media Inc., 1. Auflage, November 2012, Sebastopol, CA (USA) – ISBN 978-1-449-31520-7

[2] Borenstein, Greg: *Making Things See – 3D Vision with Kinect, processing, Arduino and MakerBot*: O'Reilly Media Inc., 1. Auflage, Januar 2012, Sebastopol, CA (USA) – ISBN 978-1-449-30707-3

[3] Bradski, Gary & Kaehler, Adrian: *Learning OpenCV – Computer Vision with the OpenCV Library*: O'Reilly Media Inc., 1. Auflage, September 2008, Sebastopol, CA (USA)
– ISBN 978-0-596-51613-0

[4] Langanière, Robert: *OpenCV 2 – Computer Vision Application Programming Cookbook*: Packt Publishing, 1. Auflage, Mai 2011, Birmingham (UK)
– ISBN 978-1-849513-24-1

[5] OpenCV development team (Hrsg.): *OpenCV online documentation (Version 2.4.4.0. - 2.4.8.0),* Zeitraum 04/2013 – 01/2014,
`http://docs.opencv.org/index.html`

[6] Jähne, Bernd: *Digitale Bildverarbeitung und Bildgewinnung*: Springer Verlag, 7. Auflage, 2012, Berlin/Heidelberg (D) – ISBN 978-3-642-04951-4

[7] Russ, John C.: *The image processing handbook*: CRC/Taylor & Francis, 5. Auflage, 2007, Boca Raton, Fla. (USA) – ISBN 978-0-8493-7254-4

[8] Microsoft Corporation: *Komponenten des Kinect-Sensors,* Aufgerufen am 02.01.2014,
`https://support.xbox.com/de-DE/xbox-360/kinect/kinect-sensor-components`

[9] Microsoft Corporation: *PrimeSense Supplies 3-D-Sensing Technology to "Project Natal" for Xbox 360*, 31. Mai 2010, Aufgerufen am 02.01.2014,
`http://www.microsoft.com/en-us/news/press/2010/mar10/03-31primesensepr.aspx`

[10] ROS wiki (Hrsg.): *Technical description of Kinect calibration,* Aufgerufen am 02.01.2014, `http://wiki.ros.org/kinect_calibration/technical`

[11] Matthias Kofler (Hrsg.): *Intebriebnahme und Untersuchung des Kinect Sensors – Masterprojekt I*, FH Oberösterreich, September 2011, Wels (A)

[12] Eckhardt Seyfert, DGPF (Hrsg.): *DGPF Tagungsband Nr. 21*: Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation (DGPF) e.V., 1. Auflage, 2012, Potsdam (D) – ISSN 0942-2870

[13] OpenKinect (Hrsg.): *OpenKinect Wiki*, Zeitraum 04/2013 – 01/2014, `http://openkinect.org/wiki`

[14] Bouguet, Jean-Yves (Hrsg.): *Camera Calibration Toolbox for Matlab*, California Institute of Technology, Zeitraum 04/2013 – 01/2014, `http://www.vision.caltech.edu/bouguetj/calib_doc/`

[15] Viola, Paul & Jones, Michael (Hrsg.): *Rapid Object Detection using a Boosted Cascade of Simple Features*: IEEE Paper , 1. Auflage, 2001, Cambridge, MA (USA) – ISBN 0-7695-1272-0

[16] Kölzer, Hans-Peter: *Bildverarbeitung und Mustererkennung - Vorlesungsskript*: Hochschule für angewandte Wissenschaften (HAW), März 2012, Hamburg (D)

Glossar

AdaBoost ..... Adaptiv Boosting Klassifikator

CMOS ..... Complementary Metal Oxide Semiconductor (Metalloxid-Halbleiter)

Detection ..... Registrierung von Objekten in einem Bild

Features ..... Merkmale

IR ..... Infrarotbild

OpenCV ..... Open Computer Vision – Bildverarbeitungs-Bibliothek

Opening ..... Morphologische Basisoperation (Erosion und darauf folgende Dilatation)

Partikelfilter ..... Schätzfilter zur Objektverfolgung

RGB/BGR ..... Additiver Farbraum mit den Farben Blau, Gelb, Rot bzw. Rot, Gelb, Blau

RGBD-Bild ..... Kombiniertes RGB-Farb- und Tiefenbild (Depth)

SIFT ..... Scale-invariant feature transform

SURF ..... Speeded Up Robust Features

Tracking ..... Verfolgung von bewegten Objekten

Xbox (360) ..... Microsoft® Spielekonsole

x,y,z-Richtung ..... Beschreibung der Richtung in Bildern sowie im 3D-Raum – x: horizontal, y: vertikal, z: Tiefe

A Installationsanleitung

Anhang A dieser Arbeit fasst die wesentlichen Schritte und Befehle für die reibungslose Installation der Treiber und Programme dar, welche für diese Thesis benötigt werden. Als Betriebsystem ist wie unter Abschnitt 2.3.2 beschrieben Linux/Ubuntu 12.04 LTS zu wählen. Hierauf aufbauend ergeben sich die nachfolgenden Schritte:

A.1 Kinect-Treiber

Die Installation des OpenKinect freenect-Treibers für die Anbindung des Kinect-Sensors ergibt sich wie folgt, wobei zunächst notwendige, abhängige Komponenten installiert werden müssen:

```
sudo apt-get install git-core cmake freeglut3-dev pkg-config build-essential
libxmu-dev libxi-dev libusb-1.0-0-dev

git clone git://github.com/OpenKinect/libfreenect.git
cd libfreenect
mkdir build
cd build
cmake ..
make
sudo make install

sudo ldconfig /usr/local/lib64/

sudo gedit /etc/modprobe.d/blacklist.conf
-> blacklist gspca_kinect
```

A.2 openFrameworks bzw. Code::Blocks

Die Installation von openFrameworks bzw. der Programmierumgebung Code::Blocks ergibt sich nach [1, S.28f].

Zunächst ist der Download der aktuellen openFrameworks-Version und damit inbegriffenen Version von Code::Blocks von http://www.openframeworks.cc/download durchzuführen. Es folgt nach dem Entpacken der Dateien und Ordner:

```
cd openFrameworks/scripts/Linux/Ubuntu

sudo ./install_codeblocks.sh
sudo ./install_dependencies.sh

sudo ./install_codecs.sh
```

Die Installation der Video- und Audio-Codecs ist dabei optional.

A.3 OpenCV

Vor der eigentlichen Installation der OpenCV-Bibliothek sind wieder eine Reihe von Abhängigen Komponenten zu installieren bzw. ggf. nachzuinstallieren. Hierzu gehören:

```
sudo apt-get install build-essential cmake git libgtk2.0-dev pkg-config python-dev
python-numpy libqt4-dev libqt4-opengl-dev ffmpeg libjpeg-dev libpng-dev
```

Ist dieses durchgeführt, wird eine Version (hier 2.4.4a) der OpenCV-Bibliothek benötigt. Diese kann unter http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.4/ bezogen werden. Nach dem Entpacken des Inhalts folgt dann:

```
cd ~/opencv-2.4.4
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D
WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON
-D BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON ..
make
sudo make install
```

Nach der länger andauernden Installation sind noch weitere Einstellungen notwendig, welche wie folgt aussehen:

```
sudo gedit /etc/ld.so.conf.d/opencv.conf
-> /usr/local/lib

sudo ldconfig /etc/ld.so.conf

sudo gedit /etc/bash.bashrc
-> PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
-> export PKG_CONFIG_PATH
```

A.4 Treibertausch für Kinect (Web-Cam vs. libfreenect)

Wie in Abschnitt 2.3.2.2 erwähnt, ist es möglich die Kinect auch als Web-Cam zu betreiben und mit Hilfe von gStreamer die Farb- und Infrarotbilder auszulesen. Hierzu muss der gspca-Treiber für die Kinect aktiviert oder deaktiviert werden.

Der aktuelle Status kann über folgenden Befehl abgefragt werden:

```
lsmod | grep gspca
```

Soll der gspca-Treiber aktiviert werden, d.h. die Kinect als Web-Cam eingebunden werden, so ist nachfolgender Befehl auszuführen:

```
sudo modprobe gspca_kinect
```

Soll hingegen der freenect-Treiber aktiv sein, so sind die folgenden beiden Befehle notwendig.

```
sudo modprobe -r gspca_kinect
sudo modprobe -r gspca_main
```

B Quellcode

Als Anhang B an diese Masterthesis sind die Main-Funktionen sowie die Header-Files der entwickelten Softwarequellcodes mit deren Strukturen, Klassen, Funktionsprototypen angefügt. Der vollständige Quellcode kann hingegen der beigelegten CD entnommen werden.

B.1 Hauptprogramm (main.cpp)

```
/** MASTERTHESIS
 * Titel: Verfahren zur Personenverfolgung und -zählung in stark frequentierten
 *        Durchgangsbereichen mit Hilfe des Kinect 3D-Bidsensors
 * Autor: Gregor Falk (1931144)
 * Dateiname: main.cpp
 * Erstellt am: 01.09.2013
 * Version: 1.0 vom 20.12.2013
 */

/// Includes
#include <opencv2/opencv.hpp>
#include <opencv2/legacy/legacy.hpp>
#include <opencv2/nonfree/nonfree.hpp>
#include <opencv2/nonfree/features2d.hpp>
#include "functions.h"

/// Namespace
using namespace cv;
using namespace std;

/// Präprozessor-Definitionen
#define ESCAPE 27
#define MINIPIC_SIZE 480/10
#define DEGREE 25 // Winkelangabe für Kinect-Sensor
#define CASCADE 0 // 0: Gesicht, 1: Oberkörper
#define DEPTH_THRESHOLD 122 // Schwellwert für die Tiefendarstellung (Schleier)

/// Globale Variablen
CascadeClassifier cascade; // Kaskadenklassifikator für Gesichtsdetektion
int counter = -1; // Zählvariable für gezählte Personen
timeval begin, finish, begin_getVideos, finish_getVideos,
        begin_calib, finish_calib; // Variablen für Zeitmessung

/// MAIN-Funktion
int main(int argc, char **argv)
{
   cout << "BEGIN" << endl;

   /// Initialisierung
   bool status(false);
   Mat videoMat;  // Variable für das Farbbild
   Mat depthMat(Size(640,480),CV_16UC1);  // Variable für das Tiefenbild (16Bit)
   Mat depthf(Size(640,480),CV_8UC1);  // Variable für das Tiefenbild (8Bit)

   /// Initialisierung der Kalibrierung
   bool calibrated (false);
   CameraCalibrator calibrateCam_RGB;
   CameraCalibrator calibrateCam_IR;
   Mat H;   // Homogenitäts-Matrix
   calibrated = f_loadCameraMatrix(calibrateCam_RGB,calibrateCam_IR,H);
   if(calibrated==false)
   {
      f_calibration(calibrateCam_RGB,calibrateCam_IR,H);
```

```
54       calibrated = true;
55
56       cvWaitKey(0);
57       cvDestroyAllWindows();
58    }
59
60    /// Initialisierung des Kaskaden-Klassifikators
61    #if (CASCADE==0)
62    if(cascade.load("haarcascade_frontalface_default.xml") == false)
63    // Mögliche Angaben: haarcascade_frontalface_default.xml ; haarcascade_mcs_upperbody.xml ;
64    //                   lbpcascade_frontalface.xml ; haarcascade_frontalface_default.xml
65    #elif (CASCADE==1)
66    if(cascade.load("haarcascade_mcs_upperbody.xml") == false)
67    #endif
68       printf("cascade.load() failed...\n");
69    else
70       printf("cascade.load() completed...\n");
71
72    /// Initialisierung des Kinect-Sensors
73    Freenect::Freenect freenect;
74    MyFreenectDevice& kinect_device = freenect.createDevice<MyFreenectDevice>(0);
75    /*double deg = DEGREE;
76    kinect_device.setTiltDegrees(deg);*/
77    kinect_device.startVideo();
78    kinect_device.startDepth();
79    kinect_device.setLed(LED_RED);
80
81    /// Initialisierungen für den Programmablauf
82    char code = (char)-1;
83    Mat frame;  // Variable für das Arbeitsbild
84    vector<Rect> faces;  // Variable für detektierte Gesichter
85    int num_faces(0), num_person(0), num_filter(0);
86    int temp_person_counter(0);
87
88    /// Initialisierung der Partikelfilter
89    vector<Partikelfilter> partikelfilter;
90    Mat plot_xyz(Size(640,480+DEPTH_THRESHOLD*FAKTOR_XYZ_PLOT+50),CV_8UC3,Scalar(0));
91
92    /// Initialisierungen für Fenster
93    namedWindow("Detektionsbereich",CV_WINDOW_AUTOSIZE);
94    moveWindow("Detektionsbereich", 690, 0);
95    namedWindow("XYZ_3D_Plot",CV_WINDOW_AUTOSIZE);
96    moveWindow("XYZ_3D_Plot", 1038, 0);
97    namedWindow("Arbeitsbereich",CV_WINDOW_AUTOSIZE);
98    moveWindow("Arbeitsbereich", 0, 369);
99    //namedWindow("depthf",CV_WINDOW_AUTOSIZE);
100   #if SHOW_RAW
101   namedWindow("video",CV_WINDOW_AUTOSIZE);
102   namedWindow("depth",CV_WINDOW_AUTOSIZE);
103   #endif
104
105   /// Main-Loop
106   while(1)
107   {
108      gettimeofday(&begin, 0);
109
110      ///XYZ-3D-Plot
111      rectangle(plot_xyz,Point(0,0),Point(640,480+DEPTH_THRESHOLD*FAKTOR_XYZ_PLOT+50),
112         CV_RGB(0,0,0),CV_FILLED);
113      line(plot_xyz, Point(0,480), Point(640,480), CV_RGB(255,255,255), 2);
114      rectangle(plot_xyz,Point(0,480),Point(640,480+43*FAKTOR_XYZ_PLOT),CV_RGB(100,0,0),
115         CV_FILLED);
116      rectangle(plot_xyz,Point(0,480+DEPTH_THRESHOLD*FAKTOR_XYZ_PLOT),
117         Point(640,480+DEPTH_THRESHOLD*FAKTOR_XYZ_PLOT+50),CV_RGB(100,0,0),CV_FILLED);
118      rectangle(plot_xyz,Point(160,0),Point(640-160,480),CV_RGB(255,255,0),2);
119
120      /// Bilddaten der Kinect empfangen
121      gettimeofday(&begin_getVideos, 0);
122
123      status = f_getVideo(kinect_device,videoMat);
124      if(!status)
```

```
125       {
126          cerr << "Error reading videoMat" << endl;
127          continue;
128       }
129       status = f_getDepth(kinect_device,depthMat,depthf);
130       if(!status)
131       {
132          cerr << "Error reading depthMat" << endl;
133          continue;
134       }
135
136       gettimeofday(&finish_getVideos, 0);
137       cout << "Dauer_Bilddaten: " <<
138          (double)(finish_getVideos.tv_sec+((double)finish_getVideos.tv_usec)/1000000)*1000-
139          (double)(begin_getVideos.tv_sec+((double)begin_getVideos.tv_usec)/1000000)*1000
140          << "ms" << endl;
141
142       /// Kalibrierung
143       gettimeofday(&begin_calib, 0);
144
145       f_getStream(kinect_device,videoMat,depthf,calibrated,false,0,
146                   calibrateCam_RGB,calibrateCam_IR);
147       videoMat.copyTo(frame);
148
149       warpPerspective(depthf,depthf,H,videoMat.size(),INTER_LINEAR|WARP_INVERSE_MAP,
150                       BORDER_CONSTANT,Scalar(255,255,255));
151       //imshow("depthf",depthf);
152
153       Mat temp_depth;
154       depthf.copyTo(temp_depth);
155       threshold(temp_depth,temp_depth,DEPTH_THRESHOLD,255,THRESH_BINARY_INV);
156       Mat element1(10,10,CV_8U,Scalar(1));
157       morphologyEx(temp_depth,temp_depth,MORPH_ERODE,element1);
158       Mat element2(15,15,CV_8U,Scalar(1));
159       morphologyEx(temp_depth,temp_depth,MORPH_DILATE,element2);
160       //imshow("temp_depth",temp_depth);
161
162       vector<Mat> frame_channels;
163       split(frame,frame_channels);
164       bitwise_and(frame_channels[0],temp_depth,frame_channels[0]);
165       bitwise_and(frame_channels[1],temp_depth,frame_channels[1]);
166       bitwise_and(frame_channels[2],temp_depth,frame_channels[2]);
167       merge(frame_channels,frame);
168
169       gettimeofday(&finish_calib, 0);
170       cout << "Dauer_Kalibrierung: " <<
171          (double)(finish_calib.tv_sec+((double)finish_calib.tv_usec)/1000000)*1000-
172          (double)(begin_calib.tv_sec+((double)begin_calib.tv_usec)/1000000)*1000
173          << "ms" << endl;
174
175       /// Gesichtsdetektion
176       rectangle(frame,Point(FRAME_ROI,0),Point(640-FRAME_ROI,480),CV_RGB(255,255,0),2);
177       Mat frame_roi(frame,Rect(FRAME_ROI, 0, 640-2*FRAME_ROI, 480));
178       faces = f_detection(frame_roi,cascade);
179       num_faces = faces.size();
180
181       /// Partikelfilter und Zählung
182       vector<Rect> person(faces);
183       num_person = person.size();
184       cout << "Personen: " << num_person << endl;
185       num_filter = partikelfilter.size();
186       #if SHOW_SURF_SIFT
187       int temp_num_filter = num_filter;
188       #endif
189       cout << "Filter: " << num_filter << endl;
190
191       f_tracking(num_filter,partikelfilter,num_person,person,depthf,plot_xyz,frame,num_faces,
192                  temp_person_counter);
193
194       #if SHOW_SURF_SIFT
195       for(int f=num_filter; f<temp_num_filter; f++)
```

```
196         {
197             char str[35];
198             sprintf(str,"SURF/SIFT-Vergleich: %d",f);
199             cvDestroyWindow(str);
200         }
201         #endif
202 
203         /// Ausgabe der Mini-Bilder
204         for(int i=0; ((i < num_filter) && (num_filter > 0) && (num_filter <= 20)); i++)
205         {
206             char str[3];
207             sprintf(str,"%d",i);
208             Mat roi;
209             resize(partikelfilter[i].get_picture(),roi,Size(MINIPIC_SIZE,MINIPIC_SIZE));
210 
211             putText(roi,str,cvPoint(0.3*MINIPIC_SIZE,0.75*MINIPIC_SIZE),FONT_HERSHEY_PLAIN,
212                     0.04*MINIPIC_SIZE,cvScalar(0,0,255),1.5,CV_AA);
213             if(i<10)
214             {
215                 Mat target_display_area(frame,Rect(frame.size().width-MINIPIC_SIZE,i*MINIPIC_SIZE,
216                                         MINIPIC_SIZE,MINIPIC_SIZE));
217                 roi.copyTo(target_display_area);
218             }
219             else
220             {
221                 Mat target_display_area(frame,Rect(frame.size().width-(MINIPIC_SIZE*2),
222                                         (i-10)*MINIPIC_SIZE,MINIPIC_SIZE,MINIPIC_SIZE));
223                 roi.copyTo(target_display_area);
224             }
225         }
226 
227         gettimeofday(&finish, 0);
228         cout << "Gesamtdauer: " <<
229             (double)(finish.tv_sec+((double)finish.tv_usec)/1000000)*1000-
230             (double)(begin.tv_sec+((double)begin.tv_usec)/1000000)*1000
231             << "ms" << endl << endl;
232 
233         /// Ausgabe
234         imshow("Arbeitsbereich",frame);
235         imshow("XYZ_3D_Plot",plot_xyz);
236 
237         /// Key-Handler
238         code = (char)waitKey(1);
239         if(code==ESCAPE || code=='q' || code=='Q')
240             break;
241         else if(code=='c')
242             {
243                 temp_person_counter = 0;
244                 for(int i=0; i<(int)partikelfilter.size();i++)
245                 {
246                     partikelfilter[i].clear();
247                 }
248                 partikelfilter.clear();
249                 plot_xyz.release();
250                 plot_xyz.create(480+DEPTH_THRESHOLD*FAKTOR_XYZ_PLOT+50,640,CV_8UC3);
251             }
252     }
253 
254     /// Kinect beenden
255     kinect_device.stopVideo();
256     kinect_device.stopDepth();
257     kinect_device.setLed(LED_BLINK_GREEN);
258     /*freenect.deleteDevice(0);*/
259     cvDestroyAllWindows();
260 
261     return 0;
262 }
```

B.2 Kinect-Treiber (kinect_driver.h)

```
#ifndef KINECT_DRIVER_H
#define KINECT_DRIVER_H
/** MASTERTHESIS
 * Titel: Verfahren zur Personenverfolgung und -zählung in stark frequentierten
 *        Durchgangsbereichen mit Hilfe des Kinect 3D-Bidsensors
 * Autor: Gregor Falk (1931144)
 * Dateiname: kinect_driver.h
 * Erstellt am: 01.09.2013
 * Version: 1.0 vom 20.12.2013
 */

/// Includes
#include <libfreenect.hpp>            // Kinect-Lib.
#include <opencv2/opencv.hpp>         // OpenCV2-Lib.
#include <math.h>

#define RGB_IR 0     // 0: RGB, 1: IR
#define DEPTH_BIT 11 // 11 Bit, 10 Bit
#define DEPTH_COUNT pow(2,DEPTH_BIT)
#define RANGE_8BIT 255

/// Namespace
using namespace cv;
using namespace std;

/// Klassenprototyp für den Kinect-Treiber
class MyFreenectDevice : public Freenect::FreenectDevice
{
   public:
      /// Konstruktor
      MyFreenectDevice(freenect_context *_ctx, int _index);

      /// Methode zum Empfangen der Video-Daten
      void VideoCallback(void* _video, uint32_t timestamp); // Do not call directly
      /// Methode zum Empfangen der Tiefen-Daten
      void DepthCallback(void* _depth, uint32_t timestamp); // Do not call directly

      /// Methode zum Empfangen des Farbbildes
      bool getVideo(Mat& output);
      /// Methode zum Empfangen des Tiefenbildes
      bool getDepth(Mat& output);

      /// Methode zur Tiefenkalkulation
      uint16_t calcDepth(uint16_t depth);

      /// Methode zum Setzten eines neuen Farbbildmodus ('VideoMode')
      void setVideoMode(freenect_resolution resolution, freenect_video_format format);
      /// Methode zum Lesen des aktuellen Farbbildmodus ('VideoMode')
      freenect_frame_mode getVideoMode();

   private:
      /// Klassenvariablen
      vector<uint16_t> lut_depth;
      bool new_video_frame;
      bool new_depth_frame;
      freenect_device *device;
      freenect_frame_mode video_mode;
      Mat depthMat;
      Mat videoMat_RGB;
      Mat videoMat_IR;
      Mutex video_mutex;
      Mutex depth_mutex;
};
#endif
```

B.3 Kalibrierung-Klasse (calibration.h)

```
 1 #ifndef CALIBRATION_H
 2 #define CALIBRATION_H
 3 /** MASTERTHESIS
 4  * Titel: Verfahren zur Personenverfolgung und -zählung in stark frequentierten
 5  *        Durchgangsbereichen mit Hilfe des Kinect 3D-Bidsensors
 6  * Autor: Gregor Falk (1931144)
 7  * Dateiname: calibration.h
 8  * Erstellt am: 01.09.2013
 9  * Version: 1.0 vom 20.12.2013
10  */
11
12 /// Includes
13 #include <opencv2/opencv.hpp> // OpenCV2-Lib.
14
15 /// Namespace
16 using namespace cv;
17 using namespace std;
18
19 /// Klassenprototyp für die Kalibrierung
20 class CameraCalibrator
21 {
22    public:
23       /// Konstruktor/Destruktor
24       CameraCalibrator();
25       ~CameraCalibrator();
26
27       /// Methode zum Erkennen der Schachbrettpunkte
28       int addChessboardPoints(const vector<string> &filelist, Size &boardSize);
29
30       /// Methode zum Zusammenführen aller 2D- und 3D-Punkte
31       void addPoints(const vector<Point2f> &imageCorners,
32                      const vector<Point3f> &objectCorners);
33
34       /// Methode zum Kalibrieren über 'calibrateCamera()'
35       double calibrate(Size &imageSize);
36
37       /// Methode zum Neu-Ausrichten der Kamerabilder
38       Mat remap(const Mat &image);
39
40       /// Methode zum Auslesen der 3D-Objekt-Punkte
41       vector<vector<Point3f> > get_objectPoints();
42       /// Methode zum Auslesen der 2D-Objekt-Punkte
43       vector<vector<Point2f> > get_imagePoints();
44
45       /// Methode zum Auslesen der Kamera-Matrix
46       Mat get_CamMatrix();
47       /// Methode zum Auslesen der Verzerrungskoeffizienten
48       Mat get_distCoeffs();
49
50       /// Methode zum Setzen der Kamera-Matrix
51       void set_cameraMatrix(Mat matrix);
52       /// Methode zum Setzen der Verzerrungskoeffizienten
53       void set_distCoeffs(Mat coeffs);
54
55    private:
56       /// Klassenvariablen
57       vector<vector<Point3f> > objectPoints;
58       vector<vector<Point2f> > imagePoints;
59       Mat cameraMatrix;
60       Mat distCoeffs;
61       int flag;
62       Mat map1, map2;
63       bool mustInitUndistort;
64       Mat R_Mat, T_Vec;
65 };
66 #endif
```

B.4 Partikelfilter-Klasse (partikelfilter.h)

```
#ifndef PARTIKELFILTER_H
#define PARTIKELFILTER_H
/** MASTERTHESIS
 * Titel: Verfahren zur Personenverfolgung und -zählung in stark frequentierten
 *        Durchgangsbereichen mit Hilfe des Kinect 3D-Bidsensors
 * Autor: Gregor Falk (1931144)
 * Dateiname: partikelfilter.h
 * Erstellt am: 01.09.2013
 * Version: 1.0 vom 20.12.2013
 */

/// Includes
#include <opencv2/opencv.hpp>          // OpenCV2-Lib.
#include <opencv2/legacy/legacy.hpp>
#include "opencv2/legacy/compat.hpp"
#include <stdio.h>

/// Namespace
using namespace cv;
using namespace std;

/// Präprozessor-Definitionen
#define drawCross( frame, center, color, d )                  \
  line( frame, cv::Point( center.x - d, center.y - d ),           \
    cv::Point( center.x + d, center.y + d ), color, 2, CV_AA, 0);   \
  line( frame, cv::Point( center.x + d, center.y - d ),           \
    cv::Point( center.x - d, center.y + d ), color, 2, CV_AA, 0 )

#define FAKTOR_XYZ_PLOT 3

#define PLOT_PARTICLES false
#define SHOW_TEXT false

/// Globale Definitionen
const CvScalar CYAN = CV_RGB(0,255,255);
const CvScalar GREEN = CV_RGB(0,255,0);
const CvScalar BLUE = CV_RGB(0,0,255);
const CvScalar YELLOW = CV_RGB(255,255,0);
const CvScalar MAGENTA = CV_RGB(255,0,255);

/// Klassenprototyp für das Partikelfilter
class Partikelfilter
{
   public:
      /// Konstruktor/Destruktor
      Partikelfilter(int _nParticles, int _dim, float _xRange, float _yRange, float _zRange,
         Point3f _point);
      ~Partikelfilter();

      /// Methode zur Berechnung und Aktualisierung des Partikelfilters
      void filter(Mat& frame, int index_person, Mat& plot_xyz);

      /// Methode zum Leeren des Partikelfilters
      void clear();

      /// Methode zum Lesen der Partikelfilter-Aktivität
      bool get_active();
      /// Methode zum Setzen der Partikelfilter-Aktivität
      void set_active(bool _active);

      /// Methode zum Lesen des Zählers
      int get_counter();
      /// Methode zum Addieren des Zählers
      void add_counter(int count);
      /// Methode zum Lesen des Tracking-Status
      bool get_tracked();
```

```
 67       /// Methode zum Setzen des Tracking-Status
 68       void set_tracked(bool status);
 69
 70       /// Methode zum Lesen des Personen-Bildes
 71       Mat& get_picture();
 72       /// Methode zum Setzen des Personen-Bildes
 73       void set_picture(Mat& roi);
 74
 75       /// Methode zum Lesen der prognostizierten x-Koordinate
 76       float get_est_x();
 77       /// Methode zum Lesen der prognostizierten y-Koordinate
 78       float get_est_y();
 79       /// Methode zum Lesen der prognostizierten z-Koordinate
 80       float get_est_z();
 81       /// Methode zum Lesen der gesammten Vorhersagtepunkte
 82       vector<Point3f> get_particleV();
 83
 84       /// Methode zum Lesen des aktuellen Messpunktes
 85       Point3f get_measurement();
 86       /// Methode zum Setzen eines neuen Messpunktes
 87       void set_measurement(Point3f measurement_point);
 88
 89       /// Methode zum aktualisieren der Filterzeit
 90       double filter_time();
 91
 92    private:
 93       /// Klassenvariablen
 94       bool active;
 95       int counter;
 96       bool tracked;
 97       timeval time;
 98
 99       Mat picture;
100       int nParticles;
101       int dim;
102       float xRange;
103       float yRange;
104       float zRange;
105       float* minRange;
106       float* maxRange;
107
108       CvMat LB, UB;
109       CvConDensation* condens;
110       float* trans_matrix;
111
112       vector<Point3f> measV, particleV;
113       float est_x, est_y, est_z;
114       float dist_x, dist_y, dist_z;
115       bool measuring;
116
117       Point3f measurement;
118 };
119 #endif
```

B.5 Funktionen (functions.h)

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H
/** MASTERTHESIS
 * Titel: Verfahren zur Personenverfolgung und -zählung in stark frequentierten
 *        Durchgangsbereichen mit Hilfe des Kinect 3D-Bidsensors
 * Autor: Gregor Falk (1931144)
 * Dateiname: functions.h
 * Erstellt am: 01.09.2013
 * Version: 1.0 vom 20.12.2013
 */

/// Includes
#include <opencv2/opencv.hpp>          // OpenCV2-Lib.
#include <opencv2/nonfree/features2d.hpp>
#include <stdio.h>
#include "kinect_driver.h"
#include "calibration.h"
#include "partikelfilter.h"

/// Namespace
using namespace cv;
using namespace std;

/// Präprozessor-Definitionen
#define FILENAME "snapshot/"
#define SUFFIX ".png"
#define ESCAPE 27
#define BACKSPACE 8
#define UP 'u'
#define DOWN 'd'
#define NEUTRAL 'n'
#define MAX_CAL_IMG 20
#define MUL_DIST 3

#define POSITION_DIFF_3D_X 25
#define POSITION_DIFF_3D_Y 50
#define POSITION_DIFF_3D_Z 10

#define DELETE_TIME 1000
#define NUM_COUNTS 5
#define FRAME_ROI 160
#define SHOW_RAW false
#define SHOW_SURF_SIFT true

/// Funktionsprototypen
/// Funktion zum Empfangen des Tiefenbildes der Kinect
bool f_getDepth(MyFreenectDevice&,Mat&,Mat&);
/// Funktion zum Empfangen des Farbbildes der Kinect
bool f_getVideo(MyFreenectDevice&,Mat&);
/// Funktion zum Empfangen der entzerrten Bilder (Farbbild und Tiefenbild)
void f_getStream(MyFreenectDevice&,Mat&,Mat&,bool,bool,int,
                 CameraCalibrator&,CameraCalibrator&);
/// Funktion zum Detektieren von Schachbrettmustern in einem Bild
bool f_chessboard(MyFreenectDevice&,Mat&,bool&,int&,bool&,bool&);
/// Funktion zum Auslesen und Verwalten gedrückter Tasten
bool f_keyHandler(char,MyFreenectDevice&,Mat&,Mat&,bool,double&,int&,int&,bool&,bool&);
/// Funktion zum Speichern von Schachbrettmustern-Bildern
void f_snapshootCalibration(MyFreenectDevice&,int&,Mat&);
/// Funktion zum Speichern von Bildern (Aufnahme einer Bildsequenz)
void f_record(MyFreenectDevice&,int&,Mat&,Mat&);
/// Funktion zum Umstellen des Farbbild-Formats (RGB <-> IR)
void f_switchFormat(MyFreenectDevice&);
/// Funktion zur Kalibrierung
void f_calibration(CameraCalibrator&,CameraCalibrator&,Mat&);
/// Funktion zum Laden der Kamera-Matrizen
bool f_loadCameraMatrix(CameraCalibrator&,CameraCalibrator&,Mat&);
```

```
67 /// Funktion zum Detektieren von Gesichtern
68 vector<Rect> f_detection(Mat&,CascadeClassifier&);
69 /// Funktion zum SIFT/SURF-Gesichtsvergleich
70 bool f_compareFaces(Mat,Mat,int);
71 /// Funktion zum Tracking von Personen mittels Partikelfilter
72 void f_tracking(int&,vector<Partikelfilter>&,int&,vector<Rect>&,Mat&,Mat&,Mat&,int&,int&);
73
74 #endif
```

C Inhalt des Datenträgers

Als Anhang C dieser Masterthesis ist die Struktur des beigefügten Datenträgers in Form einer CD-ROM aufgeführt:

- **/01_Masterarbeit_GregorFalk.pdf**
  Es handelt sich dabei um diese Version der Masterthesis im pdf-Format.

- **/02_Realisierung_Quellcodes/**
  Dieser Ordner enthält die für die Realisierung verwendeten Code::Blocks-Projekte und Quellcodes.

- **/02_Realisierung_Quellcodes/…/snapshot/calibration/**
  Dieser Ordner enthält die jeweils 20 Farb- und Infrarotbilder, welche für die Kalibrierung verwendet werden.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, den 17.01.2014

Ort, Datum Gregor Falk