TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Elektrotechnik und Informationstechnik
Professur Schaltkreis- und Systementwurf

MASTERARBEIT

Optische Methoden zur Positionsbestimmung auf Basis von Landmarken

zur
Erlangung des akademischen Grades
M.Sc.

Eingereicht von: Sebastian Bilda
Matrikel Nr.: 286046
Einreichungsdatum: 24. April 2017

Prüfer: Prof. Dr.-Ing. Göran Herrmann
Betreuer: Dipl.-Ing. Thomas Graichen

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Aufgabenstellung

zur

Abschlussarbeit
im Studiengang Master Information and Communication Systems

für

Herrn Sebastian Bilda
geb. am 28. August 1991 in Altenburg

zum Thema

Optische Methoden zur Positionsbestimmung auf Basis von Landmarken

Betreuer/ Prüfer: Prof. Dr. Göran Herrmann
Prüfer: Dipl.-Ing. Thomas Graichen
Ausgabedatum: 04.10.2016
Abgabedatum: 13.03.2017
Tag der Abgabe:

Unterschrift: ....
Prof. Dr. J. T. Horstmann
Vorsitzender des Prüfungsausschusses

Aufgabenbeschreibung:

Zur Bestimmung der Position finden derzeitig in Abhängigkeit des Anwendungsfalls verschiedenste Verfahren Anwendung. In Außenraumumgebungen werden satellitengestützte Verfahren, wie GPS oder GLONASS, in Verbindung mit einem Kompass eingesetzt, um neben den Ort auch die Orientierung zu ermitteln.

In Innenraumumgebungen ist jedoch die Positionierung über solche Verfahren nicht möglich, da sowohl die Signalstärke von GPS/GLONASS nicht genügend ist als auch das für den Kompass notwendige Magnetfeld der Erde durch metallische Körper und Konstruktionen gestört wird. Zur Lösung dieser Probleme werden daher lokale, funkbasierte Verfahren, Beschleunigungssensoren oder auch optische Systeme eingesetzt (QR-Codes, Wiedererkennung von Merkmalen).

Insbesondere die optischen Verfahren können durch die Bestimmung der Entfernung eines bekannten Objektes (sog. Landmarke) und die Lage dieses innerhalb des Bildes einen wesentlichen Beitrag zur genauen Ermittlung der Position und Orientierung leisten. In der vorliegenden Arbeit sollen daher Methoden recherchiert, untersucht und entwickelt werden, die eine Positionsbestimmung mit Hilfe von einer Kamera und einer detaillierten Innenraumkarte ermöglicht. Ein Schwerpunkt der Arbeit soll in der Evaluierung der umgesetzten Methoden liegen, um sowohl die Genauigkeit als auch Zuverlässigkeit dieser ermitteln zu können.

Die Arbeit ist in folgende Teilaufgaben strukturiert:

- Literaturrecherche über bestehende Verfahren zur Positionsbestimmung auf Basis optischer Landmarken
- Konzepterstellung und Umsetzung einer Datenbank/Karte zur Verwaltung von Landmarken (auf Basis von OpenStreetMap-Daten) mit dem Fokus eines performanten Look-Ups
- Auswahl eines Kamerasystems und Aufnahme eines Landmarkendatensatzes (Referenz- und Echtweltdaten)
- Konzepterstellung und Umsetzung eines Kamera-basiertens Verfahren zur Erkennung von Landmarken und Bestimmung der Entfernung und Orientierung zu diesen.
- Evaluation des umgesetzten Systems anhand einer Auswahl von Landmarken und verschiedener Orientierungen und Entfernungen zu diesen. Dabei soll auch die Bestimmung von besonders geeigneter Landmarken erfolgen.

Kurzbeschreibung

Die Innenraumpositionierung kommt in der heutigen Zeit immer mehr Aufmerksamkeit zu teil. Neben der Navigation durch das Gebäude sind vor allem Location Based Services von Bedeutung, welche Zusatzinformationen zu spezifischen Objekten zur Verfügung stellen Da für eine Innrenraumortung das GPS Signal jedoch zu schwach ist, müssen andere Techniken zur Lokalisierung gefunden werden. Neben der häufig verwendeten Positionierung durch Auswertung von empfangenen Funkwellen existieren Methoden zur optischen Lokalisierung mittels Landmarken. Das kamerabasierte Verfahren bietet den Vorteil, dass eine oft zentimetergenaue Positionierung möglich ist.

In dieser Masterarbeit erfolgt die Bestimmung der Position im Gebäude mittels Detektion von ArUco-Markern und Türschildern aus Bilddaten. Als Evaluationsgeräte sind zum einen die Kinect v2 von Microsoft, als auch das Lenovo Phab 2 Pro Smartphone verwendet worden. Neben den Bilddaten stellen diese auch mittels Time of Flight Sensoren generierte Tiefendaten zur Verfügung. Durch den Vergleich von aus dem Bild extrahierten Eckpunkten der Landmarke, mit den aus einer Datenbank entnommenen realen geometrischen Maßen des Objektes, kann die Entfernung zu einer gefundenen Landmarke bestimmt werden. Neben der optischen Distanzermittlung wird die Position zusätzlich anhand der Tiefendaten ermittelt. Abschließend werden beiden Verfahren miteinander verglichen und eine Aussage bezüglich der Genauigkeit und Zuverlässigkeit des in dieser Arbeit entwickelten Algorithmus getroffen.

Abstract

Indoor Positioning is receiving more and more attention nowadays. Beside the navigation through a building, Location Bases Services offer the possibility to get more information about certain objects in the enviroment. Because GPS signals are too weak to penetrate buildings, other techniques for localization must be found. Beneath the commonly used positioning via the evaluation of received radio signals, optical methods for localization with the help of landmarks can be used. These camera-based procedures have the advantage, that an inch-perfect positioning is possible.

In this master thesis, the determination of the position in a building is achieved through the detection of ArUco-Marker and door signs in images gathered by a camera. The evaluation is done with the Microsoft Kinect v2 and the Lenovo Phab 2 Pro Smartphone. They offer depth data gained by a time of flight sensor beside the color images. The range to a detected landmark is calculated by comparing the object´s corners in the image with the real metrics, extracted from a database. Additionally, the distance is determined by the evaluation of the depth data. Finally, both procedures are compared with each other and a statement about the accuracy and responsibility is made.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abkkürzung	**Bedeutung**
ADB	Android Debug Bridge
AoA	Angle of Arrival
FPS	Frames per Second
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
JRE	Java Runtime Enviroment
JNI	Java Native Interface
LBS	Location Based Service
LM	Levenberg-Marquardt
Lat	Längengrad (Latitude)
Lon	Breitengrad (Longitude)
MILC	Microsoft Indoor Localisation Competition
NaN	Not a Number
PNG	Portable Network Graphics
RSSI	Receive Signal Strength Indicator
SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
SVM	Support Vector Machine
SQL	Structured Query Language
TDOA	Time Difference of Arrival
TDOF	Time Difference of Flight
TOA	Time of Arrival
TOF	Time of Flight
UWB	Ultra-wideband
XML	Extensible Markup Language

Symbolverzeichnis

Symbol	**Einheit**	**Bezeichnung**
A	-	intrinsische Matrix
c	$\frac{m}{s}$	Lichtgeschwindigkeit
D	m	Distanz
E	-	extrinsische Matrix
f_x, f_y	mm	Brennweite in x- und y-Richtung
f_c	Hz	Modulationsfrequenz
F	-	Residuum
H	-	interne Transformationsmatrix
$H(x)$	-	Histrogrammwert an Stelle x
I	-	Einheitsmatrix
I_x, I_y	-	Gradient in x-, y-Richtung
J	-	Jakobi-Matrix
k	-	Verzerrungskoeffizient
k_u, k_v	$\frac{Pixel}{mm}$	Skalierungsfaktoren der internen Transformation
$M(u, v)$		Anstieg im Bildpunkt (u,v)
P	-	perspektivische Transformationsmatrix
R	-	Rotationsmatrix
r	-	Rotationachse
$s(x, y)$	Pixel	Sensorkoordinate
s	m	Strecke
t	s	Zeit
T		Threshold
u, v	Pixel	normierte Bildkoordinaten
u_0, v_0	Pixel	Verschiebungsfaktoren der internen Transformation
$x^{(k)}$	-	k-ter Näherungswert
X_c, Y_c, Z_c	mm	Punkt im Kamerakoordinatensystem
X_w, Y_w, Z_w	mm	Punkt im Weltkoordinatensystem
ε	mm	Distanzfehler
σ^2	-	Varianz
σ_b^2	-	Interklassen-Varianz
σ_W^2	-	Intraklassen-Varianz
ϕ		Phasenverschiebung
μ	-	Dämpfungsfaktor
θ		Drehwinkel
$\theta(u, v)$		Orientierung im Bildpunkt (u,v)

1. Einleitung

Seit vielen Jahren ermöglichen globale satellitengestützte Navigationssysteme (GNSS) wie das amerikanische GPS oder das russische Glonass eine metergenaue Lokalisierung auf der Erdoberfläche. Beispielsweise ortet das GPS die Position auf circa 5 Meter genau [Stand 2017] [SBPNT17]. Durch die große Verbreitung von Smartphones besitzt jeder Anwender auch ein passendes Empfangsgerät, da diese meist mit einem oder mehreren GNSS-Modulen ausgestattet sind. Jedoch benötigt diese Art der Lokalisierung aufgrund der geringen Signalstärke eine Sichtverbindung zu mindestens vier Satelliten, um eine genaue dreidimensionale Ortsbestimmung durchführen zu können. Dies ist beispielsweise in Häusern unmöglich, da das Signal die Wände nicht durchdringt. Doch da der Mensch die meiste Zeit in Gebäuden verbringt und es oft schwierig ist, sich in großen, unbekannten Gebäuden zu orientieren, sollte auch hier eine Möglichkeit zur Navigation vorhanden sein. So könnte man sich in einem Shoppingcenter direkt zu einem bestimmten Geschäft führen lassen oder mithilfe des Smartphones schnell das richtige Gate in einem Flughafen finden. Neben der Innenraumnavigation gibt es bereits eine Vielzahl von sogenannten Location Based Services (LBS). Hierbei werden abhängig vom Standort dem Nutzer unterschiedliche Informationen auf sein Handy übermittelt. Im Außenbereich wird dies schon lange eingesetzt. So wird bei einer Wetter App automatisch das aktuelle Wetter in der nächsten Stadt angezeigt. Auch zeigen Kartendienste wie Google Maps bei einer Suche nur die jeweiligen Geschäfte in der Nähe an. Im Innenraum wird LBS auch schon vereinzelt eingesetzt. Oft sind dies Museen, die den Nutzer nicht nur beispielsweise zu einem Bild navigieren können, sondern auch gleich Informationen darüber mitliefern. Dieses geschieht auch automatisch, wenn man sich in Reichweite eines Objektes befindet. Als Technik wird hierfür oft Bluetooth eingesetzt [inf17b]. Doch gerade auch für den kommerziellen Bereich sind LBS sehr interessant. So können an einem Geschäft vorbeigehende Passanten aktuelle Angebote oder Rabattaktionen übermittelt werden, um sie damit eventuell zu animieren in den jeweiligen Laden zu gehen[eta16]. Auch kann ein Bewegungsprofil der einzelnen Nutzer erstellt werden und so herausgefunden werden, bei welchen Produkten die meiste Zeit verbracht wurde. Einen noch weiteren Schritt geht Amazon mit seinen neuen Amazon Go Läden. Hierbei gibt es keine Kassensysteme. Der Benutzer registriert sich einmal beim Betreten des Ladens mithilfe seines Smartphones und nimmt sich die Waren die er benötigt. Durch Bildverarbeitung werden die einzelnen entnommenen Produkte erkannt. Um diese einer Person zuordnen zu können, muss eine sehr genaue Positionierung stattfinden. Beim Verlassen des Ladens werden schließlich die Warenpreise automatisch vom Amazon-Account abgezogen.[Ama17]

All diese Beispiele zeigen, dass ein enormes Potential im Bereich der Innenraumpositionierung steckt und dieses ähnlich den Alltag verbessern kann, wie es die Außenraumlokalisierung getan hat.

1.1. Motivation

Für die Orientierung sind die Augen das wichtigste Sinnesorgan des Menschen. Anhand bekannter Objekte kann eine Person genau bestimmen wo sie sich gerade befindet, indem er mithilfe der Entfernung und der Orientierung zu dem spezifischen Objekt seine Position ermittelt. Ähnlich läuft es bei der Positionsbestimmung auf Basis von optischen Landmarken. Hierbei werden mithilfe einer Kamera Bilder aufgenommen und nach bekannten Objekten mittels verschiedener Bildverarbeitungsalgorithmen abgesucht. Wird ein bekanntes Objekt gefunden, werden geometrische Informationen aus beispielsweise einer Datenbank extrahiert. Mithilfe dieser und der kameraspezifischen Parameter kann dann wiederum die Entfernung und Orientierung berechnet werden. Der Vorteil dieser optischen Methode gegenüber der weiter verbreiteten Positionierung mittels Funkwellen besteht in der oft genaueren Lokalisierung. Während Funkwellen durch Absorption, Reflexion oder Interferenzen das Ergebnis verfälschen können, liefert die visuelle Bestimmung der Position durch eine bestehende Sichtverbindung und oft auch geringer Entfernung zur Landmarke genauere Ergebnisse. Jedoch spielen die Qualität der Kamera und des Objektes bezüglich der Fehler eine große Rolle, da eine ungenaue Abbildung des Objektes im Bild auch zu einer ungenaueren Entfernungs- und Orientierungsbestimmung führt. Da jedes Smartphone heutzutage auch eine Kamera besitzt, soll das bereits bestehende Indoor-Positionierungsframework an der Professur Schaltkreis- und Systementwurf der technischen Universität Chemnitz um diese Funktion erweitert werden. Somit kann eine relativ genaue Initialisierung der Position ermöglicht werden. Gegebenenfalls kann die optische Lokalisierung um Augmented Reality Anwendungen erweitert werden.

1.2. Aufgabe

Der Schwerpunkt dieser Masterarbeit liegt in der Untersuchung und Evaluierung von geeigneten optischen Positionierungsmethoden auf Basis von Landmarken. Nachdem verschiedene Ansätze bezüglich der Aufgabenstellung recherchiert wurden, soll eine Implementierung auf einem Desktop-Gerät mit Windows Betriebssystem erfolgen. Die Bild- und Tiefendaten werden mithilfe einer Microsoft Kinect v2 gewonnen. Später soll der Algorithmus in die Android Umgebung portiert werden. Als Evaluationsgerät wird hierbei das Lenovo Phab 2 Pro verwendet, welches neben einer RGB- auch eine Fisheyekamera, sowie Time of Flight Sensor zur Tiefenmessung besitzt. Nach einer erfolgreichen Funktionsprüfung werden die Messdaten mit Realdaten verglichen und es wird die Positionierungsgenauigkeit ermittelt. Mithilfe dieser Ergebnisse soll schließlich eine Aussage zur praktischen Verwendbarkeit dieses Algorithmus getroffen werden. Weiterhin soll überprüft werden, ob und in welchem Grad die Tiefenmessung die optische Bestimmung der Distanzen die Ergebnisse verbessern kann.

1.3. Aufbau

Das erste Kapitel befasst sich mit bereits bestehenden Innenraumlokalisierungssystemen. Dabei wird ein grober Überblick der bestehenden technischen Variationen zur Positionierung gegeben. Angelehnt an die jährlich stattfindende Microsoft Indoor Localisation Challenge erfolgt eine Unterteilung in infrastrukturlose und infrastrukturbasierende Methoden. Hierbei werden sowohl bereits kommerzielle, als auch wissenschaftliche Methoden genauer betrachtet.

Die zur Arbeit benötigte Theorie wird im zweiten Kapitel behandelt. Zunächst werden die Begriffe optischen Landmarken, das Kameramodell, sowie Farb- und Tiefenbilder behandelt. Danach werden wichtige, in dieser Arbeit verwendete Bildverarbeitungsfunktionen beschrieben und der nötige theoretische Hintergrund erklärt.

Im dritten Kapitel folgt schließlich die Erläuterung des praktischen Teils zur Erkennung von optischen Landmarken. Nach einer Vorstellung der Hardware, folgt eine Gesamtbetrachtung des Algorithmus. Schließlich werden die einzelnen Blöcke näher erläutert und mit (Pseudo-)Codebeispielen anschaulich dargestellt.

Die Auswertung findet im fünften Kapitel statt. Hierbei wird zuerst die Testumgebung beschrieben. Als nächstes folgt eine Gegenüberstellung von Ground Truth Daten und der Messwerte. Schließlich werden die Differenzen der Distanzwerte zum wahren Wert, sowie die Trefferrate zur Erkennung der Marker ermittelt und ausgewertet. Ebenfalls erfolgt sowohl eine zeitkritische Betrachtung des Gesamtsystems, als auch der einzelnen Komponenten.

Abschließend wird im sechsten Kapitel ein Fazit getroffen. Ausgehend von der Aufgabenstellung wird überprüft, ob das System die notwendigen Anforderungen erfüllen konnte. Es werden positive, sowie negative Aspekte des vorgestellten Algorithmus erläutert und Anregungen zu einer möglichen Fortführung der Arbeit gegeben.

2. Stand der Technik

Seit 2014 veranstaltet der Softwarekonzern Microsoft die *Microsoft Indoor Localisation Competition (MILC)*, bei dem sowohl akademische, als auch kommerzielle Systeme gegeneinander antreten. In der ersten Kategorie mussten die Teams vorgegebene 3D Positionen bestimmen. Hierbei war es erlaubt eigene Hardware wie Laser-Scanner oder Radare zu benutzen. Dementsprechend konnte auch das Empfangsgerät frei gewählt werden. Im Gegensatz dazu, hatten teilnehmende Teams der anderen 2D Kategorien die Aufgabe, zweidimensionale Lokalisierungskoordinaten zu ermitteln. Doch durfte dafür nur die gegebene Infrastruktur aus fünf WLAN Stationen, einem FM-Signal und dem Erdmagnetfeld benutzt werden. In beiden Kategorien gewann das Team, welches die kleinste Abweichung zu den vorher genau bestimmten 15 Referenzpunkten mit ihrem System gemessen hat. Sie durften sich jeweils über ein Preisgeld von 1000$ freuen. In der 3D Kategorie gewann Zhang et al. mithilfe eines LIDARs. Su et al konnten mit der kombinierten Nutzung von geomagnetischen Fingerprinting und Koppelnavigation den ersten Platz in der 2D-Kategorie für sich beanspruchen [Mic16]. Auch 2017 fand vom 18. bis 19. April 2017 in Pittsburgh, PA, USA wieder die *Microsoft Indoor Localisation Competition* statt. Jedoch waren vor Abgabe dieser Arbeit noch keine Ergebnisse verfügbar.
Nachfolgend werden einige Techniken zur Indoor Lokalisierung vorgestellt. Wie bei dem vorher erwähnten Wettbewerb, wird auch in dieser Betrachtung zwischen infrastrukturlosen und nicht infrastrukturlosen Methoden unterschieden. Weiterhin werden nur clientbasierte Technologien betrachtet. Dies bedeutet, dass die Positionierung auf dem Endgerät der zu lokalisierenden Person geschieht und nicht mittels externer Sensoren und Serversystemen. Dieses wird beispielsweise genutzt, um Bewegungsprofile zu erstellen oder Lebewesen beziehungsweise Gegenstände verfolgen zu können und ist in dieser Arbeit nicht von Bedeutung.

2.1. Infrastrukturlose Lokalisierung

Wie bereits erwähnt werden bei dieser Art der Positionsbestimmung nur vorhandene Strukturen verwendet. Dies hat den Vorteil, dass die Realisierung der Systeme günstiger und weniger zeitaufwendig erfolgen kann, da nur eine Software auf dem Empfangsgerät installiert werden muss. Eine sehr beliebte Technik ist hierbei die Ortung über WLAN-Signale, da diese heutzutage in fast allen größeren Gebäuden vorhanden sind. Um die Position ermitteln zu können, kann beispielsweise die Signalstärke (RSSI[1]) gemessen werden. Mithilfe eines Signalpfadverlustmodells, wie zum Beispiel dem des freien Raumes, kann anhand des Verhältnisses von empfangener zu gesendeter Leistung die Entfernung zum Sender ermittelt werden. Ein Beispiel hierfür wird in [MBL$^+$09] beschrieben. Da meist

---

[1] Receive Signal Strength Indicator

jedoch keine Sichtverbindung zum WLAN-Router besteht und auch verschiedene Verlustmodelle nicht die Signalbeeinflussungen wie Interferenzen, Reflexionen und Absorptionen vollständig beschreiben können, wird oft eine Karte mit Referenzwerten der Signalstärken der Router aufgenommen und mit den jeweiligen Positionsdaten verknüpft. Dieses nennt man auch Fingerprinting, da jede Position ihre speziellen Signalstärken (Fingerabdrücke) zugewiesen bekommt. Nachdem dieses für genügend Punkte erstellt wurde, kann in der praktischen Verwendung beispielsweise das Smartphone eines Benutzers den jetzigen Fingerprint berechnen und ihn mit den Referenzwerten vergleichen, wobei die niedrigste Differenz die wahrscheinlichste Position ist. Dieser Ablauf wird in folgender Abbildung 2.1 noch einmal anschaulich dargestellt. Da die Technik relativ einfach zu benutzen ist, ist sie Bestandteil von vielen kommerziellen Anbietern von Indoor-Positionierungstechnik, wie beispielsweise Infsoft [inf17a] oder indoora [ind17a]. Nachteilig ist jedoch, dass die Benutzung nicht sofort erfolgen kann, da erst eine Referenzkarte erstellt werden muss. Diese muss außerdem immer angepasst werden, falls sich an der Routerinfrastruktur etwas ändert. Auch fluktuieren die Werte aufgrund von beispielsweise anderen Personen, da diese ebenfalls die Signalwellen durch Absorption schwächen. Weiterhin kann diese Technik nicht mit einem iOS Betriebssystem verwendet werden, da hier keine Möglichkeit besteht als Nutzer die Signalstärke zu messen. Noch zu erwähnen sind die Techniken Angle of Arrival (AOA), Time of Arrival (TOA), Time Difference of Arrival/Flight (TDOA/TDOF) und Time of Flight (TOF). Hierbei wird der Einfallswinkel des Signals oder die Zeitdifferenz eines oder mehrerer Signale gemessen und mithilfe von Triangulation oder Trilateration die Position bestimmt. Aufgrund der besonderen Hardware, welche entweder zum Bestimmen des Einfallswinkels oder zur genauen Synchronisation mit dem Sender erforderlich ist, soll hierauf nicht weiter eingegangen werden, da nur Techniken für Smartphones beziehungsweise Tablets in diesem Abschnitt betrachtet werden. Da jedoch in der infrastrukturbasierten Lokalisierung viele Techniken darauf basieren, sei für

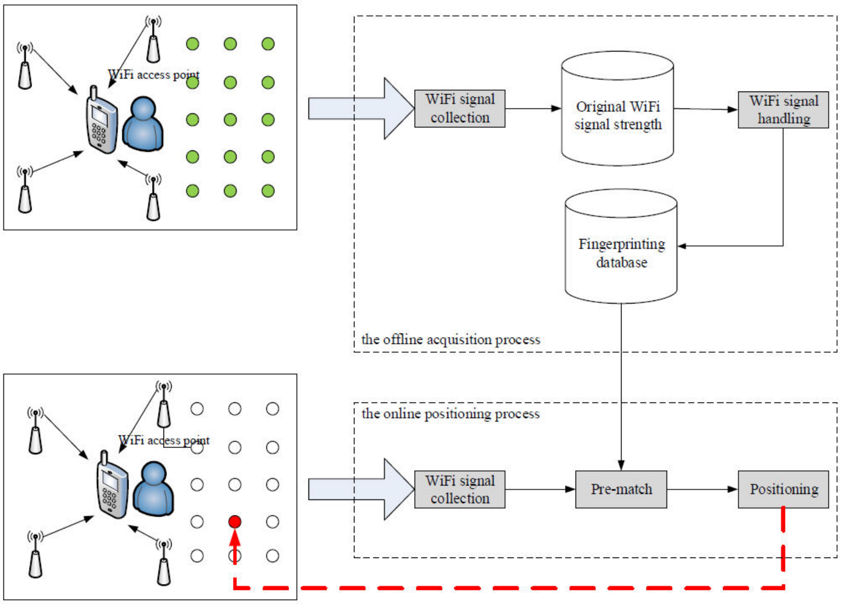


Abbildung 2.1.: Ablauf der Positionierung mittels Fingerprinting, (aus [MGHX15])

weiterführende Informationen auf [Nov12] verwiesen.
Eine weitere Methode, welche auf die Basis des Fingerprinting setzt, ist die Messung des Erdmagnetfeldes. Anders als die drahtlosen Signalmethoden wirken hier die metallischen Strukturen der Gebäude als vorteilhaft, da sie das Erdmagnetfeld individuell beeinflussen. Somit lassen sich im Gebäude einzigartige Magnetstärkefingerabdrücke ermitteln, welches in Abbildung 2.2 anschaulich dargestellt wird. Die Positionierung über das Magnetfeld findet ebenfalls bereits Anwendung in kommerziellen Systemen. Die Firma IndoorAtlas zum Beispiel setzt ausschließlich auf magnetisches Fingerprinting zur Innenraumlokalisierung und erreicht dabei Genauigkeiten von circa einem Meter [Ind17b].

Nachteil der bisher vorgestellten Methoden ist, dass diese stark abhängig von den im Endgerät verbauten Sensoren sind. So können unterschiedliche Geräte beispielsweise verschiedene Magnetfeld- oder Signalstärken von WLAN oder Mobilfunk am gleichen Punkt mit gleichen Bedingungen messen. Deswegen existieren neben der Messung von Signal- oder anderen Feldstärken auch andere Ansätze zur Lokalisierung, wie die des Dead Reckonings[2]. Unter Einsatz des im Smartphone verbauten Beschleunigungssensors können Schritte detektiert werden, indem man die Beschleunigung in z-Richtung misst, siehe beispielsweise [LSVW11]. Hinzu kommt die Orientierungsbestimmung mittels eines Gyroskops oder eines Kompasses. Bei letzterem führt die bereits erwähnte Veränderung des Magnetfeldes durch die Gebäudeinfrastruktur jedoch oft zu falschen Ergebnissen, weswegen er nur unterstützend verwendet werden sollte. Mithilfe eines vorgefertigten Gebäudeplanes kann nun die Bewegung des Benutzers in Echtzeit verfolgt werden. Das heißt mit jedem detektierten Schritt wird die Position im Gebäude aktualisiert. Ein großer Nachteil ist, dass eine manuelle Initialisierung der Position stattfinden muss, da eine Erstbestimmung der Lokalisierung mit dieser Methode nicht möglich ist. Ebenfalls entstehen

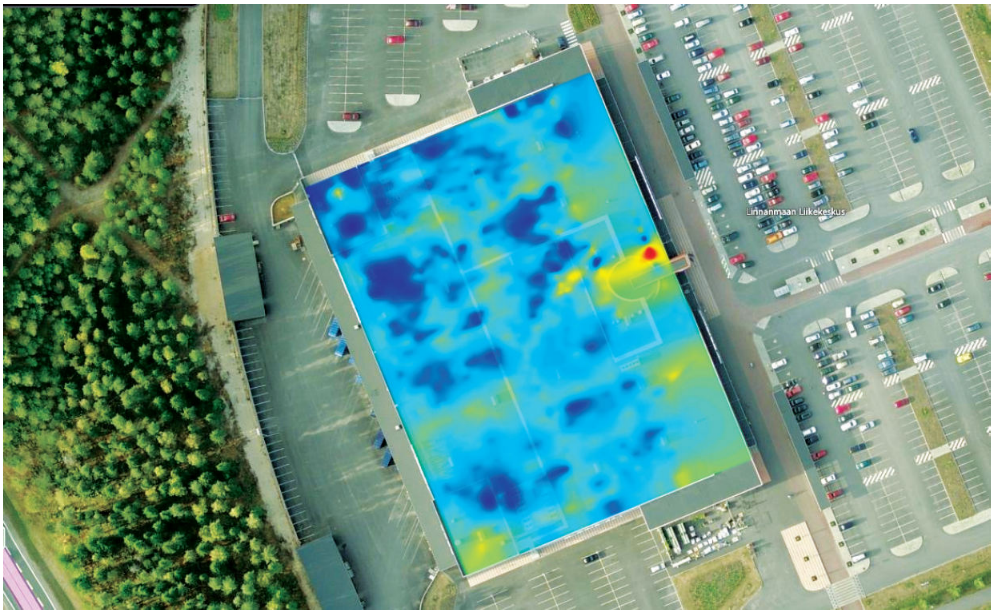


Abbildung 2.2.: Magnetisches Fingerprint eines Gebäudes, (aus [Ste14], Quelle: IndoorAtlas)

---

[2] dt. Koppelnavigation

bei der Koppelnavigation große Integrationsfehler. Wenn beispielsweise einmal die falsche Abbiegung bei einer Kreuzung im Gebäude ermittelt wurde, erfolgt die weitere Positionierung weiterhin entlang des falschen Pfades. Wegen dieser beiden Kritikpunkte, wird das Dead Reckoning oft nur als unterstützende Methode zur Lokalisierung verwendet. Jedoch ist anzumerken, dass bei der Microsoft Indoor Localisation Competition die beiden genauesten Methoden mit einem Fehler von 0,75 Meter beziehungsweise 1,11 Meter auf Koppelnavigation (mit manueller Initialisierung) basierten [Mic16].
Zuletzt sollen noch die Techniken zur optischen Positionsbestimmung ohne zusätzlich angebrachte Hilfsmittel vorgestellt werden. Durch die Veröffentlichung des auch in dieser Arbeit benutzten Projektes Tango von Google wurde es leichter in diesem Bereich Fortschritte zu erzielen. Das seit November 2014 öffentlich zugängliche Framework bietet dem Entwickler die Funktionen des *Motion Trackings*, *Area Learnings* und *Depth Perception*. Jedoch kann Tango nur auf Geräten benutzt werden, welche bestimmte Hardwareanforderungen erfüllen. Dieses sind hauptsächlich die Bereitstellung einer Fish-Eye Kamera und einer Tiefenkamera zusätzlich zur RGB-Rückkamera, sowie eine maximale Bearbeitungszeit für die Sensorrohdaten von 50 μs [Moo17]. Durch die drei genannten Funktionalitäten ist es unter anderem möglich, Augmented Reality[3] Anwendungen zu erstellen. Aber auch Indoor Lokalisierung und sogar Navigation können realisiert werden. Zum einen geschieht dies auf Grundlage des Motion Trackings. Hierbei werden spezielle optische Featurepoints, welche einen hohen Wiedererkennungswert wie beispielsweise Ecken und Kanten besitzen, genutzt. Die Erkennung und Verfolgung geschieht hierbei durch Feature Deskriptoren, wie beispielsweise SIFT[4] oder SURF[5]. Wurden nun in zwei nacheinander folgenden Bildern Features entdeckt und jeweils einander zugeordnet, kann die translatorische und rotatorische Bewegung des Tango Gerätes mithilfe der Differenz der x- und y-Pixelpositionen bestimmt werden. Eine Besonderheit von Tango hierbei ist, dass zusätzlich dazu die Inertialsensoren, also Beschleunigungssensor und Gyroskop, des Gerätes ausgewertet werden. Durch die Fusionierung kann eine noch genauere Bestimmung der Bewegung erfolgen. Um nun eine Positionierung zu ermöglichen, wird das Area Learning benötigt. Hierbei wird ein sogenanntes ADF (Area Description File) erstellt. Dieses beinhaltet die Featurebeschreibung der Punkte, sowie die Entfernung und Orientierung zu einem frei wählbaren Startpunkt. Um das ADF zu erstellen, muss man sich lediglich durch das Gebäude bewegen und dabei die Strukturen von möglichst verschiedenen Blickwinkeln mit der Kamera erfassen. Die Feature Punkte werden automatisch detektiert und mit der Position relativ zum Startpunkt in der Datei abgespeichert, was ähnlich dem des vorher beschriebenen Fingerprintings ist. Der aus der Google I/O Konferenz aufgenommene Screenshot 2.3 verdeutlicht diesen Sachverhalt noch einmal. Befindet man sich nun irgendwo im Gebäude muss man lediglich das ADF laden und sich solange bewegen, bis bereits bekannte Feature Punkte gefunden wurden. Damit ist die Position zum festgelegten Startpunkt bekannt. Ein Nachteil ist jedoch, dass verschiedene Lichteinflüsse eine große Rolle spielen, da diese die Eigenschaften der Feature Punkte ändern. Somit sollte man verschiedene ADFs erstellen und je nach Situation das Richtige laden. Auch müssen im Gebäude genügend Strukturen vorhanden sein, da beispielsweise an ei-

---

[3] dt. erweiterte Realität - Anreicherung eines realen Bildes mit digitalen Informationen

[4] Scale-Invariant Feature Transform, siehe[Ope14a]

[5] Speeded-Up Robust Features, siehe [Ope14b]

ner weißen Wand kaum Feature Punkte ermittelt werden können. Leider war es nicht festzustellen, mit welcher Methode die Features bestimmt werden und welche Daten genau in einem ADF abgespeichert werden. [Goo17b]
Ein ähnliches Verfahren zur optischen Lokalisierung in Gebäuden, jedoch ohne Tango Framework, zeigt [WKM11]. Auch hier wird ähnlich dem des ADFs als Grundlage eine Datenbank mit Bildern aus dem Gebäude von möglichst vielen unterschiedlichen Positionen und Orientierungen erstellt. Aus dieser werden dann dominante Merkmale wie Eck- oder Kantenpunkte mithilfe des SURF-Algorithmus extrahiert und gespeichert. Will man nun seine Position bestimmen, nimmt man ein Bild oder ein Video vom aktuellen Standpunkt auf und sendet es an einen Server. Dieser bestimmt die SURF-Merkmale und ermittelt das Referenzbild, welches die meisten gemeinsamen Merkmale besitzt.

Abbildung 2.3.: links: Gelbe Punkte zeigen die erkannten Features, rechts: relative Position wird mit gematchten Features berechnet und visualisiert (aus Google I/O 2016, Quelle: Youtube)

2.2. Nicht infrastrukturlose Lokalisierung

In diesem Abschnitt werden Lösungen vorgestellt, welche oftmals eine genauere Lokalisierung liefern, jedoch müssen dafür Erweiterungen oder Änderungen an der Gebäudeinfrastruktur vorgenommen werden. Somit wird eine bessere Genauigkeit mithilfe zusätzlicher Hardware und damit höheren Installationskosten erzielt. Die ersten beiden Plätze des MILC belegen Lasersysteme. Zwar erreichen sie Genauigkeiten von 5 beziehungsweise 16 Zentimetern, jedoch sind sie aufgrund des verwendeten LIDAR-Systems[6] für die Navigation mithilfe von Smartphones oder Tablets gänzlich ungeeignet. Weiterhin erfolgreich war die Benutzung von nicht hörbaren akustischen Signalen. Hierbei wird beispielsweise von externen Lautsprechern ein Audiosignal ausgesendet, welches von einem Mikrofon am Empfänger aufgenommen und damit die Position ermittelt werden kann. Des Weiteren kann auch das Gerät des zu Lokalisierenden ein Audiosignal aussenden, wo es schließlich

---

[6] Light detection and ranging, optische Distanzmessung ähnlich der des Radars

von mehreren Empfängern registriert wird. Durch eine TDOF Berechnung kann aufgrund der unterschiedlichen Ankunftszeiten an den Empfängern die Position trianguliert und an das Smartphone gesendet werden. Diese Methode verwendet beispielsweise Telocate ASSIST und erreichte eine Genauigkeit von 1,8 Meter bei dem von Microsoft ausgetragenen Wettbewerb. Eine relativ neue Technik hingegen konnte bessere Ergebnisse erzielen und wurde in der 3D Kategorie des MILC am meisten eingesetzt. Das sogenannte Ultrawideband (UWB)[7] funkt in Deutschland auf einem Bereich von 30 MHz bis 10,6 GHz. Aufgrund der sehr großen möglichen Bandbreite haben diese Wellen nur eine sehr geringe Leistung, weswegen sie andere schmalbandige Signale, wie beispielsweise die eines WLAN-Signales, nicht stärker als normales Rauschen beeinflussen. Aus diesem Grund besitzt UWB nur eine geringe Reichweite von wenigen Metern. Zur Ortung werden sehr kurze Impulse im Nanosekundenbereich gesendet. Dieses beseitigt zwar das Problem der Multipath- Interferenzen, jedoch müssen Sender und Empfänger genau synchronisiert sein, damit zur richtigen Zeit der Sendeimpuls detektiert wird. Zwar ist diese Technik nicht auf aktuellen Smartphones verfügbar, doch ist ein zukünftiger Einsatz, insbesondere durch die mögliche Datenrate von einem Gigabit/Sekunde und mehr, möglich. Die Ortung erfolgt meist über die TDOF oder TOA, ähnlich wie bei der vorher beschriebenen akustischen Lokalisierung. Für weiterführende Informationen zu UWB wird auf [AASAe16] verwiesen.

Ein in dieser Kategorie beim MILC zwar nicht verwendeter Ansatz, jedoch in der Industrie sehr verbreitet, ist die Ortung durch Bluetooth-Beacons. So bieten unter anderem namhafte Hersteller wie Apple mit seinem *iBeacons* System oder Google mit *EddyStone* auf Bluetooth Low-Energy basierende Systeme an. Dank der seit 2010 verfügbaren Spezifikation, können die Beacon-Sendegeräte auf einer Reichweite von 10 bis 30 Meter mit einer Batterie mehrere Jahre funken. Um eine Lokalisierung zu ermöglichen, müssen mehrere Geräte in Reichweite vorhanden sein, um mithilfe der Signalstärke (RSSI) von mehreren Beacons und einer Triangulation die Position zu bestimmen. Um einem Signal das zugehörige Gerät zuordnen zu können, wird eine ID mitgesendet, welche dann in einer Datenbank nachgeschaut werden kann um wiederum die Position des Gerätes bestimmen zu können. Sehr empfehlenswert ist auch hier die vorherige Erstellung einer Karte mit Fingerprints, da dieses wie beim WLAN zu einer Verbesserung der Genauigkeit führt. Ein weiterer Vorteil eines solchen Systems ist, dass Beacons, die nur einen kleinen Bereich abdecken, ebenfalls Informationen zu seiner Umgebung senden können. Dies können in einem Museum beispielsweise Informationen zu einem Gemälde sein [Pro14] oder in einem Einkaufszentrum Rabattcoupons für ein Geschäft in der Nähe, wie es das bekannte Unternehmen Shopkick bereits tut [sho17].

Zuletzt soll die auch in dieser Arbeit verwendete Positionierung basierend auf optischen Markern betrachtet werden. Durch das Anbringen leicht zu erkennender Marker im Gebäude, kann mithilfe einer Kamera oft zentimetergenau die Position bestimmt werden. Diese Marker sind oft in der Art von QR-Codes gehalten, da sie durch ihre einfache Struktur leicht detektierbar sind. Für eine weitere, ausführliche State of the Art Betrachtung, sowie eine Gegenüberstellung von häufig verwendeten Markerarten sei auf [LDMC$^+$16] verwiesen. Die Detektion und weitere Verwendung am Beispiel eines AruCo-Marker wird im Kapitel Durchführung unter 4.4.3 beschrieben. Vorteil dieser Art der Lokalisierung

---

[7] dt. Ultrabreitband

ist, dass sie von jedem Gerät mit einer Kamera erkannt werden können und es keine Beeinflussung der Positionierung aufgrund von Signalstörungen oder anderen Schwankungen gibt. Lediglich die Qualität der Kamera spielt eine minimale Rolle. Weiterhin ist diese Methode gut skalierbar und die Positionierung erfolgt sehr schnell und sehr genau. Dennoch existieren auch einige Nachteile. Das Schwerwiegendste ist, dass diese Art der Lokalisierung nur an wenigen, mit Markern ausgestatteten, Orten verfügbar ist. Somit muss erst ein Marker gefunden werden, bevor man sich positionieren kann. Auch stellt das Anbringen der Marker einen Eingriff in die Ästhetik des Gebäudes dar, da diese an gut sichtbaren Orten angebracht werden müssen. Ein Kritikpunkt ist, dass die Marker leicht entfernbar beziehungsweise zerstörbar sind, jedoch können sie leicht wieder ersetzt werden. Aus diesem Grund werden zusätzlich zu den künstlich angebrachten Markers in dieser Arbeit die Büro- und Raumschilder zur Lokalisierung genutzt, da sie bereits zahlreich in der Infrastruktur vorhanden sind und ihre Positionen fest und bekannt sind.

3. Konzept und theoretische Grundlagen

3.1. Optische Landmarken

Schon immer spielten Landmarken zur Orientierung und Navigation eine wichtige Rolle. Diese konnten entweder natürlich, wie beispielsweise ein Berg oder eine Flussfurt, aber auch künstlich sein. Als Beispiel dafür sei eine Brücke oder ein Kirchturm genannt. Wie essentiell Landmarken für die Orientierung sind, zeigt sich in einem Labyrinth. Durch das Fehlen markanter Punkte scheint es unmöglich festzustellen, ob man an einer Position schon einmal war und wie diese eventuell in Bezug zu anderen Positionen liegt.

Laut *Sorrows* und *Hirtle* [SH99] lassen sich Landmarken in drei Kategorien mit jeweiligen Merkmalen einteilen. Zum einen wären dies *visuelle Landmarken*. Aufbauend auf Lynch [Lyn08, S. 87–91] zeichnen sie sich vor allem durch ihren hohen Kontrast zur Umgebung aus. Dieses kann aufgrund von markanten Umrissen, Formen, Farben oder anderen visuellen Auffälligkeiten geschehen. In diese Gruppe fallen die meisten Landmarken. Eine weitere Kategorie sind die *kognitiven Landmarken*. Hierbei steht die Bedeutung beispielsweise im geschichtlichen, kulturellen oder militärischen Bereich im Vordergrund. So unterscheiden sich das Geburtshaus einer berühmten Person oder das Büro des Geschäftsführers im Aussehen meist nicht von den Infrastrukturen in der Umgebung, sind jedoch aufgrund ihrer Funktion oder Geschichte wiedererkennbar. Zum Schluss werden *strukturelle Landmarken* genannt. Sie definieren sich durch ihre spezielle Rolle oder Position in einer räumlichen Struktur. Als Beispiele werden Kreuzungen oder Plätze wie der Trafalgar Square genannt.

Basierend auf dieser Einteilung lassen sich erforderliche Eigenschaften für Landmarken zur Innenraumpositionierung wie folgt definieren:

- Hoher Kontrast zu Umgebung durch Form / Farbe
- Positionierung an offensichtlichen und gut zugänglichen Orten im Gebäude

Zusätzlich dazu müssen zur eindeutigen Identifikation und anschließender Lokalisierung folgende Eigenschaften gelten:

- Einzigartig im Gebäude oder in einer festgelegten räumlichen Umgebung (zum Beispiel Etage)
- Position und äußerliche Merkmale der Landmarken sind fest und bekannt
- Markante Merkmale zur schnellen Detektion und Identifikation

3.2. Konzept

Um die für die Arbeit benötigten theoretischen Grundlagen besser einordnen zu können, zeigt nachfolgende Grafik 3.1 den prinzipiellen Ablauf des Positionierungsalgorithmus. Da die Lokalisierung auf Basis von optischen Landmarken erfolgen soll, müssen Bilddaten für die Weiterverarbeitung generiert werden. Zum einen sind dies Farbbilder, die in unterschiedlichen Formaten vorliegen können. Eine Übersicht dazu gibt 3.6. Zum anderen werden in dieser Arbeit auch Tiefendaten von Time of Flight Sensoren verwendet, auf deren Gewinnung und Eigenschaften in Abschnitt 3.7 eingegangen wird. Im Detektionsschritt werden anhand der Eingangsdaten mögliche Landmarken in Form von ArUco-Markern (siehe 3.3) oder Türschildern extrahiert. Die Beschreibung einiger dafür benötigter Bildverarbeitungsalgorithmen findet sich unter 3.8. Schließlich kann anhand der gefundenen Landmarken die Position ermittelt werden. Dafür wird die Distanz der Kamera zum Objekt ermittelt. Die Erläuterung des mathematischen, sowie theoretischen Hintergrunds kann in 3.4, beziehungsweise 3.5 nachgelesen werden. Die verwendeten Bildverarbeitungsfunktionen werden in Abschnitt 3.8 beschrieben. Da der Rückgabewert des Algorithmus eine geografische Position in Längen- und Breitengrad ist, erfolgt in 3.9 die Beschreibung dieses Koordinatensystems.

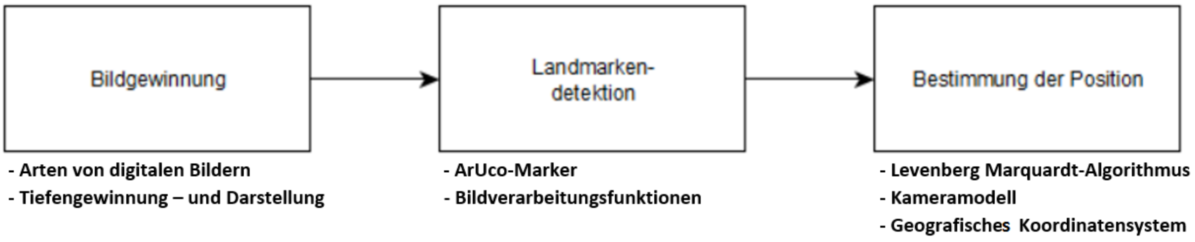


Abbildung 3.1.: Konzept dieser Arbeit

3.3. ArUco-Marker

Die in dieser Arbeit verwendeten Marker basieren auf der Veröffentlichung [GJMSe14]. Äußerlich bestehen sie zur besseren Abgrenzung zum Hintergrund und somit leichteren Detektion aus einem schwarzen Rand. Im Inneren befindet sich eine binäre Matrix, bestehend aus schwarzen und weißen Quadraten. Sie dient zur eindeutigen Identifizierung, da schwarze Quadrate in eine 0 und weiße Quadrate in eine 1 codiert werden. Durch ein sogenanntes Dictionary wird dieser binäre Code mit einer ID verknüpft. Dabei stimmen jedoch binärer und dezimaler Wert nicht überein, sondern es ist allein die Reihenfolge der Marker im Dictionary für die ID von Bedeutung. Abhängig von der Größe der binären Matrix, also der Anzahl der Quadrate in der Mitte, kann eine beliebige Anzahl von Markern generiert werden. Zum Beispiel wurde Abbildung 3.2 aus einem 6x6 Dictionary generiert. Diese Anzahl wird jedoch begrenzt durch folgende Eigenschaften:

- Anzahl der Übergänge von weiß zu schwarz / schwarz zu weiß
- Inter-Marker Distanz

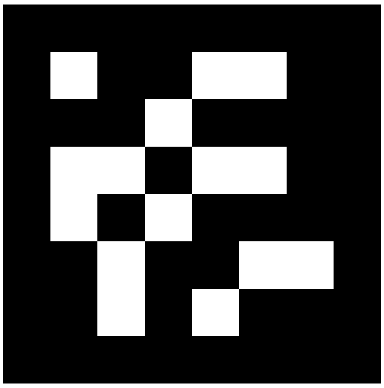


Abbildung 3.2.: 6x6 ArUco-Marker

Für den ersten Punkt gilt, je öfter sich schwarze und weiße Blöcke abwechseln, umso besser wird der Marker erkannt. Bei größeren homogenen Flächen könnte dieser sonst fälschlicherweise für ein anderes beliebiges Objekt und nicht für einen ArUco-Marker gehalten werden. Eine hohe Inter-Marker Distanz hingegen hilft, um die Marker untereinander besser unterscheiden zu können. Zur Bestimmung dieser wird die Hemming-Distanz[1] der Binärwerte gebildet. Somit gilt: Je öfter sich weiße und schwarze Quadrate an der gleichen Position für unterschiedliche Marker innerhalb eines Dictionaries unterscheiden, desto weniger erfolgt eine Fehlerkennung. Dabei ist zu beachten, dass die Marker sich auch in allen vier Ausrichtungen (0°, 90°, 180°, 270°) voneinander unterscheiden müssen, da die ArUcos auch fälschlicherweise verkehrt herum auf gehangen werden könnten und es sonst zu einer Missklassifikation kommt. Zusammenfassend lässt sich sagen: Je größer ein ArUco-Marker ist und je geringer die Anzahl dieser innerhalb eines Dictionaries ist, desto besser können ArUcos richtig klassifiziert werden. Für genauere Informationen zu ArUcos und insbesondere zur Erstellung des Dictionaries sei auf [GJMSe14] verwiesen. Der Algorithmus zur Erkennung der Marker und zur Bestimmung der ID wird in 4.4.3 beschrieben.

3.4. Levenberg-Marquardt-Algorithmus

Der Levenberg-Marquardt (LM) Algorithmus ist eine Erweiterung des Gauß-Newton-Verfahrens und dient zur Lösung nichtlinearer Ausgleichsprobleme mithilfe der Methode der kleinsten Quadrate. Ein Ausgleichsproblem liegt vor, wenn mittels Messdaten unbekannte Parameter einer Funktion oder eines Modells bestimmt werden sollen. Da Messwerte in der Realität fehlerbehafteten Schwankungen unterliegen, wäre eine mathematische Beschreibung auf der alle Messpunkte n abgebildet werden wenig sinnvoll. Aus

---

[1] Anzahl von unterschiedlichen Bitwerten an gleichen Positionen zweier Codefolgen, siehe [Kow12]

diesem Grund versucht man folgende Summe

$$S = \sum_{i=1}^{n} F_i^2 \tag{3.1}$$

zu minimieren. Das sogenannte Residuum

$$F_i(x) = \Phi(x, t_i) - y_i \tag{3.2}$$

beschreibt hierbei die Differenz von Funktionswert Φ am Messpunkt t_i für einen zu bestimmenden Parameter x und dem Messwert y_i. Das in 3.1 definierte Problem lässt sich mithilfe der euklidischen Distanz zu

$$\min \|F(x)\|_2^2 \tag{3.3}$$

umformulieren, wobei $F(x) = (F_1(x), F_2(x) \cdots F_n(x))$ der Vektor aller Residuen darstellt. Wie beim Newton-Verfahren wird durch eine Taylorentwicklung am Punkt $x^{(k)}$ mit Abbruch nach dem linearen Term das nichtlineare Gleichungssystem in eine lineare Ersatzfunktion umgeformt. Die Jakobi-Matrix

$$J(x) = \left[\frac{\sigma F_j}{\sigma x_i}\right] = \begin{bmatrix} \frac{\sigma F_1}{\sigma x_1} & \frac{\sigma F_1}{\sigma x_2} & \cdots & \frac{\sigma F_1}{\sigma x_n} \\ \frac{\sigma F_2}{\sigma x_1} & \frac{\sigma F_2}{\sigma x_2} & \cdots & \frac{\sigma F_2}{\sigma x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\sigma F_m}{\sigma x_1} & \frac{\sigma F_m}{\sigma x_2} & ,\ldots, & \frac{\sigma F_m}{\sigma x_n} \end{bmatrix}, 1 \leq j \leq m; 1 \leq i \leq n \tag{3.4}$$

beinhaltet das partielle Differential des Residuenvektors. Mithilfe dieser lässt sich das linearisierte Ausgleichsproblem wie folgt definieren:

$$\min \left\| F(x^{(k)}) + J(x^{(k)})(x - x^{(k)} \right\|_2^2 . \tag{3.5}$$

Der nächste Näherungswert $x^{(k+1)}$ bestimmt sich nach [Her11, S.289] wie folgt:

$$x^{(k+1)} = x^{(k)} - (J(x^{(k)})^T J(x^{(k)}))^{-1} J(x^{(k)})^T F(x^{(k)})) = x^{(k)} + s_k. \tag{3.6}$$

Mithilfe des Korrekturvektors s_k wird die nächste Iteration $x^{(k+1)}$ bestimmt, welche anschließend in Gleichung (3.5) als neuer Näherungswert eingesetzt wird. Die Iteration wird solange fortgesetzt bis

$$x^{(k)} - x^{(k+1)} < \epsilon \tag{3.7}$$

gilt, wobei ϵ die maximal erlaubte quadrierte Fehlertoleranz zwischen Messwerten und der ermittelten Approximation ist.

Der bisher erläuterte Gauß-Newton-Algorithmus besitzt jedoch den Nachteil, dass die Linearisierung nur für kleine Korrekturwerte s_k anwendbar ist. Weiterhin ist eine Reduzierung der Fehlerquadratsumme nicht unbedingt gewährleistet, besonders wenn die Jakobi-Matrix keinen vollen Rang besitzt. Deswegen wird im LM-Verfahren eine Abstiegsbedingung

$$F(x^{(k+1)}) < F(x^{(k)}) \tag{3.8}$$

eingeführt, welche durch Zufügen eines Dämpfungsfaktor μ bei Bestimmung der neuen Iterierten garantiert werden kann. Weiterhin kann durch Erhöhen der Dämpfungskonstante ein zu hoher Korrekturvektor verhindert werden. Das neue Ausgleichsproblem kann schließlich wie folgt definiert werden:

$$\min \left\| \begin{bmatrix} J(x^{(k)}) \\ \mu I \end{bmatrix} s^k + \begin{bmatrix} F(x^{(k)}) \\ 0 \end{bmatrix} \right\|_2 . \tag{3.9}$$

Wird die Abstiegsbedingung $x^{(k)} < x^{(k+1)} + s^{(k)}$ nicht erfüllt, so muss ein neuer Korrekturwert durch Verändern der Dämpfungskonstante bestimmt werden. Auch beim Levenberg-Marquardt-Verfahren werden die Iterationen so lange durchgeführt, bis die Fehlerquadratsumme kleiner als der Toleranzwert wird. (vgl. [Her11, S. 288–298])

3.5. Kameramodell

Die meisten Kameras basieren auf dem Grundmodell einer Lochkamera, welches in 3.3 zu sehen ist. Das Licht fällt durch ein kleines Loch, den sogenannten Brennpunkt, in die Kamera ein und wird gespiegelt, sowie auf dem Kopf stehend auf eine Bildebene projiziert. Bei einer realen Kamera ist dieser Brennpunkt die Linse. Zur einfacheren Beschreibung wird die Bildebene meist vor die Linse verschoben, da es die Spiegelungen beseitigt, jedoch nichts an den mathematischen Operationen ändert. Weiterhin existieren in dem Modell die drei Koordinatensysteme, welche auch in Bild 3.4 zu sehen sind:

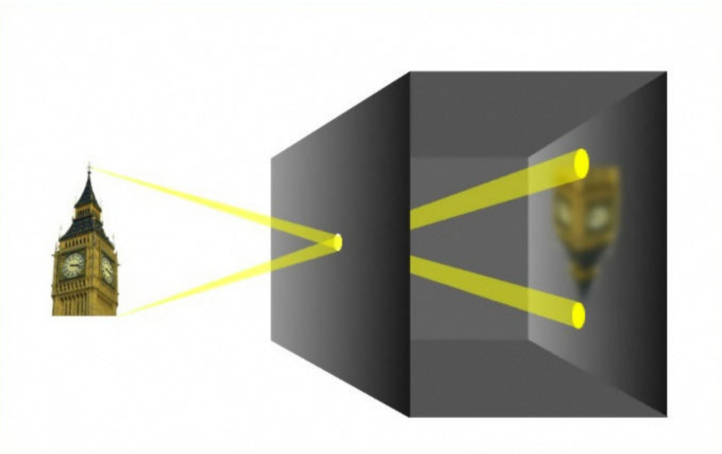


Abbildung 3.3.: Prinzip einer Lochkamera (Quelle: [wik04])

- Bildkoordinatensystem
- Kamerakoordinatensystem
- Weltkoodinatensystem

Das Weltkoodinatensystem beschreibt ein dreidimensionales Koordinatensystem mit frei wählbarem Ursprung. Jeder Punkt P kann somit durch die Koordinaten x, y und z genau beschrieben werden. Um die Koordinaten der Abbildung eines dreidimensionalen Punkt P(x,y,z) im zweidimensionalen Bildkoordinatensystem ermitteln zu können, muss dieser Punkt zuerst mithilfe der externen Matrix vom Weltkoordinatensystem in das Kamerakoordinatensystem, dessen Ursprung im Objektiv der Kamera liegt, überführt werden. Die Umwandlung in Bildkoordinaten erfolgt schließlich durch die Multiplikation des Punktes mit der intrinsischen Matrix. Für eine genauere Beschreibung, sowie eine Erläuterung des mathematischen Hintergrundes wird auf [Sch07, Abschnitt 3.1] verwiesen. Da die intrinsische, sowie extrinsische Matrix essentiell für die optische Distanzermittlung sind, werden diese in den folgenden beiden Unterkapiteln noch einmal ausführlicher behandelt. Ebenfalls wird auf die möglichen Linsenverzerrungen näher eingegangen.

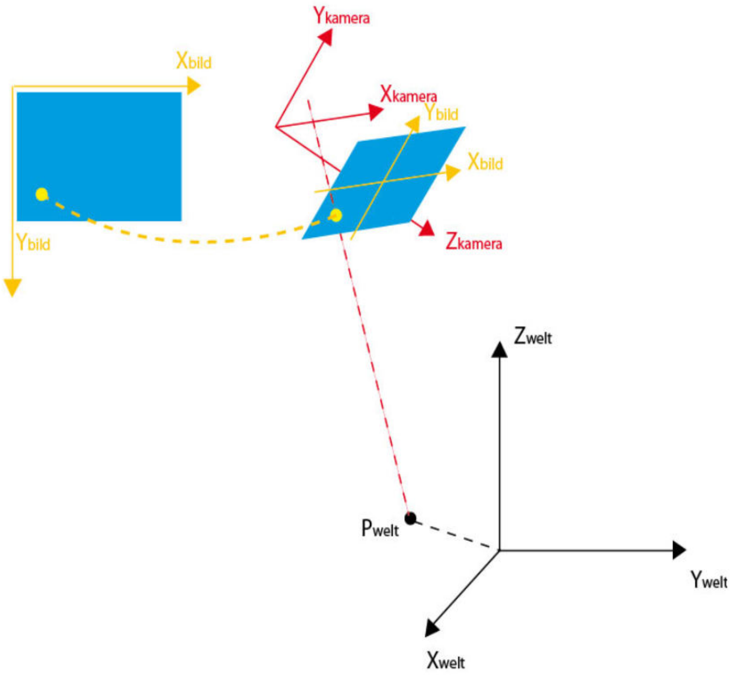


Abbildung 3.4.: Prinzipielle Darstellung der Punkte in verschiedenen Koordinatensystemen (Quelle: [Hof09])

3.5.1. Extrinsische Matrix

Mithilfe der extrinsischen Matrix

$$E = \begin{bmatrix} R_{3x3} & t_{3x1} \end{bmatrix} \tag{3.10}$$

kann eine beliebige euklidische Transformation im dreidimensionalen Raum beschrieben werden. Die Rotationsmatrix R setzt sich aus Matrixmultiplikation der Einzelrotationen um die x-, y-, sowie z-Achse zusammen. Durch den Translationsvektor t kann die Verschiebung in alle drei Dimensionen beschrieben werden. Somit besitzt die extrinsische Matrix E sechs Freiheitsgrade.

Um die Überführung eines in Weltkoordinaten vorliegenden dreidimensionalen Punktes P_w in Kamerakoordinaten mit nur einer Matrixmultiplikation durchführen zu können, muss dieser Punkt um eine Dimension erweitert werden. Die extrinsische Transformation mithilfe des nun in homogen Koordinaten vorliegenden Punktes $\tilde{P} = (P, 1)^T$ erfolgt mittels folgender Gleichung:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{21} & R_{31} & T_1 \\ R_{12} & R_{22} & R_{32} & T_2 \\ R_{13} & R_{23} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}. \tag{3.11}$$

(vgl. [Sch07, Abschnitt 3.1])

3.5.2. Intrinsische Matrix

Die Projektion des homogenen Punktes $\tilde{P_c}$ auf eine zweidimensionale Sensorkoordinate s(x,y) kann ausgehend vom zweiten Strahlensatz in das Gleichungssystem

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}, \text{ mit } x = \frac{u}{w} \text{und } y = \frac{v}{w} \tag{3.12}$$

überführt werden. Die Brennweite f enthaltende Matrix wird auch als perspektivische Transformationsmatrix P bezeichnet. Da die Sensorkoordinaten unter anderem noch in metrischen Einheiten vorliegen, wird eine weitere interne Transformationsmatrix

$$H = \begin{bmatrix} k_u & 0 & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

benötigt. Die Parameter k_u und k_v beschreiben die horizontale, beziehungsweise vertikale Anzahl der Pixel pro Millimeter. Mit diesen Skalierungen können die metrischen Koordinaten in pixelbasierte Koordinaten umgewandelt werden. Weiterhin wird durch u_0 und v_0 der Ursprung des Bildkoordinatensystems von der Mitte des Bildes in die obere linke Ecke verschoben.

Fasst man die beiden Transformationsmatrizen zusammen, erhält man die intrinsische Matrix

$$A = \begin{bmatrix} f * k_u & 0 & u_0 \\ 0 & f * k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.14}$$

welche alle internen Parameter einer Kamera enthält[2]. Um die internen Parameter bestimmen zu können, muss eine sogenannte Kamerakalibrierung durchgeführt werden. Hierbei werden Aufnahmen von (rechtwinkligen) Objekten mit bekannten geometrischen Abmessungen erzeugt, welches in den meisten Fällen ein Schachbrettmuster ist. Da die Kalibrierung jedoch nicht Bestandteil dieser Arbeit ist, wird für nähere Informationen auf [AGD07, S. 78–80] verwiesen. (vgl. [Sch07, Abschnitt 3.1])

---

[2] Anmerkung: Auf den zusätzlichen Skew-Faktor wurde hier verzichtet, da dieser nur bei nicht rechtwinkligen Pixeln betrachtet werden muss. Diese Rechtwinkligkeit ist jedoch bei heutigen Kameras gegeben.

3.5.3. Linsenverzerrung

Im Lochkameramodell wurde bisher von einer perfekten Linse ausgegangen, welche unendlich dünn ist und somit das 3D Objekt mittels eines Strahles durch den Brennpunkt auf der Bildebene abbilden kann. Bei realen Linsen ist dies jedoch nicht der Fall, da sie sowohl eine gewisse geometrische Größe besitzen, als auch in der Mitte und den Kanten unterschiedlich gewölbt sind. Somit werden die Objekte verzerrt auf der Bildebene abgebildet. Dieses kann entweder durch eine radiale oder tangentiale Verzerrung geschehen. Ersteres resultiert aus der bereits erwähnten Krümmung der Linse, da Licht an den Seiten anders gebrochen wird als in der Mitte. Damit kann es je nachdem ob die Krümmung zur Mitte oder zu den Seiten wegläuft zu kissen- oder tonnenförmiger Verzeichnung kommen, wie es in 3.5 abgebildet ist. Um diese Verzeichnung heraus rechnen zu können, müssen die radialen Verzerrungsparameter mittels Kalibration bestimmt werden. Die korrigierten Pixelwerte x' und y' werden mithilfe von

$$x' = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.15}$$

$$y' = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.16}$$

aus den alten, verzerrten Pixelwerten an der Position x und y bestimmt. Der Abstand zum Zentrum der Verzerrung r bestimmt sich aus $\sqrt{x^2 + y^2}$. Schließlich wird mit k_i der jeweilige Verzerrungskoeffizient beschrieben.

Die tangentiale Verzerrung entsteht durch eine nicht parallele Ausrichtung von Linse und Bildebene. Da durch immer genauere werdende Fertigungsprozesse diese wesentlich geringer als die radiale Verzerrung ausfällt, kann die tangentiale Verzeichnung vernachlässigt werden. (vgl.[AGD07, S. 645-649])

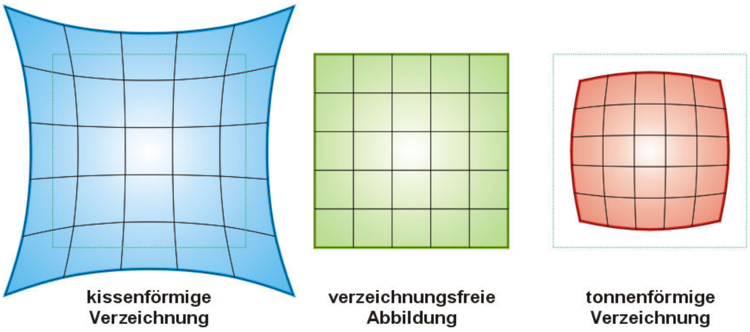


Abbildung 3.5.: Arten der radialen optischen Verzeichnungen (Quelle: [wik09b])

3.6. Arten von digitalen Bildern

In digitalen Bildern besteht jeder Bildpunkt aus Pixeln. Abgebildet werden diese mit einer unterschiedlichen Anzahl an Bits. Zum einen kann ein Pixel mehrere sogenannte Kanäle besitzen. Diese stellen eine Eigenschaft eines Bildpunktes, wie zum Beispiel dessen Helligkeit oder die Intensität eines bestimmten Farbtones wie rot, dar. Zum anderen kann dieser Pixel, je nachdem wie genau diese Abstufung innerhalb eines Kanals stattfinden soll, eine beliebig große Bittiefe besitzen. Im Normalfall beträgt diese Acht Bit. Somit können innerhalb eines Kanales 256 (2^8) Abstufungen abgebildet werden. Dies ermöglicht eine große Anzahl an unterschiedlichen Darstellungsarten eines digitalen Bildes. Nachfolgend werden die vier in dieser Arbeit verwendeten Bildarten beschrieben, deren Eigenschaften näher erläutert und in Abbildung 3.7 abschließend gegenübergestellt. Für weiterführende Informationen zu den Umwandlungen, sowie den spezifischen Werten der Konvertierungsmatrizen sei auf [Ope15a] verwiesen.

RGB-Bild

Die meisten digitalen Farbbilder basieren auf dem Schema dieses dreikanaligen Bildtyps. Hierbei setzt sich ein Pixel aus den Grundfarben Rot, Grün und Blau zusammen und besitzt meist eine Kanaltiefe von 8 Bit. Damit lassen sich alle Farben ausreichend darstellen. Es gilt dabei, je höher der Bytewert eines bestimmen Kanals, desto intensiver und auch heller ist der jeweilige Farbwert. Die unbunten Farben Schwarz und Weiß lassen sich mit {0,0,0}, respektive {255,255,255} als Bytetripel darstellen.

YUV420-Bild

Dieses Modell stammt aus den Anfängen der analogen Farbfernsehübertragungen. Um eine Abwärtskompatibilität zu den älteren Schwarz-Weiß-Fernsehern zu ermöglichen, wurde der Kanal Y für die Helligkeitswerte (Luminanzen) beibehalten. Zusätzlich zu diesem werden in den anderen beiden Kanälen U und V die Farbinformationen (Chrominanzen) übertragen. Zur besseren Veranschaulichung werden das eigentliche Bild, sowie alle drei Kanäle in nachfolgender Abbildung dargestellt. Die zusätzliche Ziffer 420 bedeutet, dass nur jede zweite Zeile und Spalte zusätzlich zur Luminanz Farbwerte enthalten. Dies dient zur Kompression der benötigten Datenmenge zur Darstellung des Bildes und wird auch Downsampling genannt. Beim späteren Upsampling wird die Farbe jeweils von vier Pixeln aus einer Farbquelle gewonnen. Da das menschliche Auge wesentlich empfindlicher auf Helligkeitsunterschiede als auf Farbunterschiede reagiert [Bou16], hat es keinen zu großen Einfluss auf die Qualität des Bildes. Zwar kann es hierbei vereinzelt, besonders bei sehr stark variierenden Farbwerten von Nachbarbildpunkten, zu fehlerhaften Farbdarstellungen kommen, doch sind diese kaum zu bemerken.

Kameras von Android Geräten liefern standardmäßig Bilder im NV21 Format, was dem YUV420sp Modell entspricht. Das sp am Ende steht für semi-planar. Anders als im planaren Modus, wo alle Kanäle nach dem Schema {[Y,Y,...];[U,U,...]; [V,V,...]} getrennt voneinander gespeichert werden (zu sehen in 3.6), werden die Chrominanzwerte bei der semiplanaren Variante abwechselnd gespeichert. Dies sieht wie folgt aus: {[Y,Y,...];[U,V,U,V]}. (vgl. [GM14, S. 17–25])

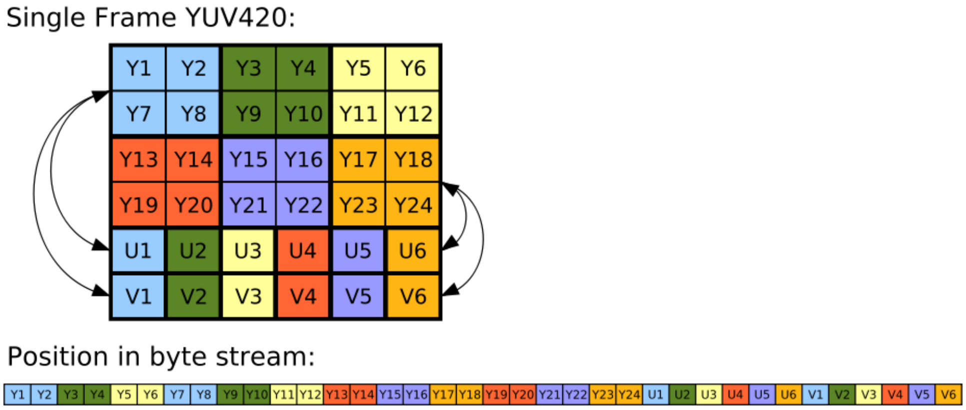


Abbildung 3.6.: Schematische Darstellung der Pixel eines Bildes im YUV420p Format (Quelle: [wik13])

Grauwertbild

Das Grauwertbild enthält nur Helligkeitswerte und sämtliche Farbinformationen sind nicht mehr vorhanden. Somit existiert auch nur noch ein Kanal für dieses Bild. Mit diesem können, bei einer Bittiefe von Acht, 256 Grauabstufungen von Schwarz (Bitwert 0) bis hin zu Weiß (Bitwert 255) dargestellt werden. Eingesetzt werden diese Bilder vor allem wenn man Strukturen wie Ecken und Kanten detektieren möchte.

Binärbild

Dieses wird oft auch als Maske bezeichnet und besteht kurzgefasst nur aus den zwei unterschiedlichen Werten null und eins. Durch die Einfachheit eignet sich dieser Typ sehr gut für schnelle bildverarbeitende Algorithmen. Ein Bild kann somit in verschiedene Bereiche segmentiert werden ohne dass sich diese überlappen. Dies können zum Beispiel eine Einteilung in Vorder- und Hintergrund oder in Kanten und Nicht-Kanten sein.

3.7. Tiefengewinnung und -darstellung

Um die Distanz zu Objekten vor einer Kamera ermitteln zu können existieren unterschiedliche Methoden. Da sowohl die Kinect v2, als auch das Lenovo Phablet 2 Pro die Time of Flight Methode verwenden, soll auch nur diese vorgestellt werden. Das grundlegende Prinzip wird in Abbildung 3.8 dargestellt. Man sieht deutlich, dass die Entfernungsbestimmung mittels der Laufzeitmessung eines ausgesendeten Lichtimpulses erfolgt. Da bekanntlich $s = v * t$ ist, gilt für die Distanz aufgrund des zurückzulegenden Hin- und Rückweges folgende Formel:

$$s = \frac{c * t}{2} \tag{3.17}$$

Hierbei ist c die Lichtgeschwindigkeit im Luftmedium und t die gemessene Zeit von Aussendung des Impulses bis zur Detektion. Aufgrund der sehr hohen Lichtgeschwindigkeit,

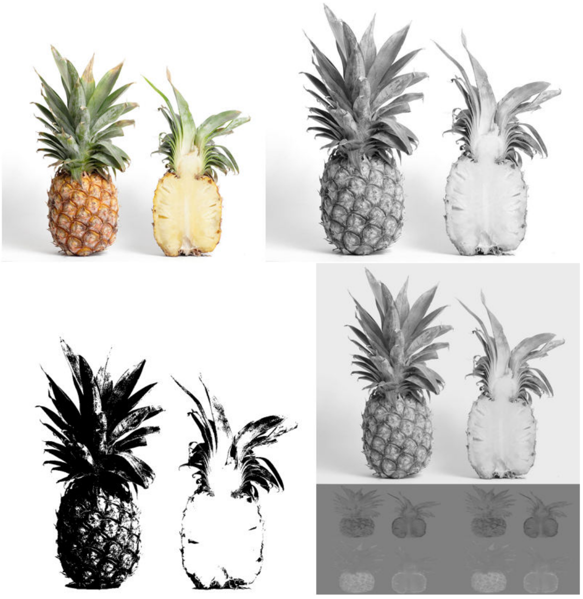


Abbildung 3.7.: Übersicht der Bildarten, links oben: RGB-Bild (Quelle: [wik09a]), rechts oben: Grauwertbild, links unten Binärbild, rechts unten: YUV420-Bild

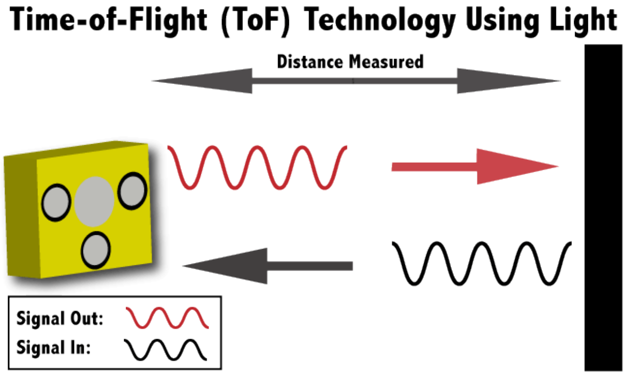


Abbildung 3.8.: Grundprinzip einer Time of Flight Messung [wik04]

werden oft nur sehr kurze Zeitdifferenzen gemessen. So beträgt die Differenz von Austritt zum Eintritt bei einem ein Meter entfernten Objekt gerade einmal rund 6,7 Nanosekunden. Somit müssen Methoden gefunden werden, welche trotz dieser kurzen Zeit eine zuverlässige Aussage über die zurückgelegte Entfernung treffen können. Zwei unterschiedliche Ansätze haben sich hierbei bewährt. Zum einen wäre das Aussenden periodischer Lichtimpulse, wobei die Dauer und die Pause zwischen Impulsen gleich sein müssen. Die Auswertung der eintreffenden Signale ist in Abbildung 3.9 verdeutlicht. Man erkennt, dass hinter jeder Lichtdiode ein Schalter angebracht ist, welcher entsprechend der Pulsdauer periodisch zwischen zwei Auswerteeinheiten hin und her schaltet. Alternativ kann der Pixel auch in zwei Lichtdioden aufgeteilt sein, welche abwechselnd durchlässig geschaltet werden. Wird das Licht nun von einem nahen Objekt reflektiert, trifft es mehrheitlich in der ersten Hälfte ein, wohingegen weiter entfernt reflektiertes Licht vorrangig in der zweiten Hälfte eintrifft. Hierbei wird deutlich, dass es eine Maximalentfernung abhängig

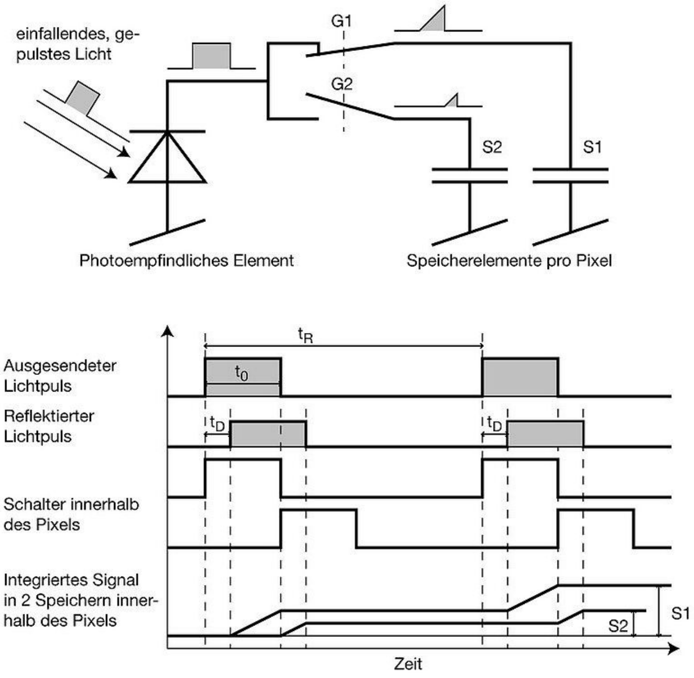


Abbildung 3.9.: Prinzip der Auswertung einer ToF Distanzmessung (Quelle: [wik08])

von der Pulsdauer gibt, da Reflexionen welche erst Mitte oder Ende der zweiten Hälfte eintreffen, nicht mehr detektiert werden können und erst in der nächsten Periode ausgewertet werden. Ausgehend von Formel 3.17 bestimmt sich die Maximalentfernung D_{max} aus:

$$D_{max} = \frac{c * t_L}{2}. \tag{3.18}$$

Zwar könnte D_{max} durch eine längere Pulsdauer erhöht werden, doch führt dieses, unter anderem durch thermisches Rauschen, zu höheren Ungenauigkeiten bei der Auswertung. Diese erfolgt nämlich durch Vergleich der beiden Einheiten, welche an die Fotodiode angeschlossen sind. Da die Diode bei Absorption des reflektierten Lichtes einen Strom liefert, kann dieser, beispielsweise über einen Kondensator, in eine Spannung umgewandelt werden. Die Entfernung D kann mithilfe von

$$D = D_{max} * \frac{U_2}{U_1 + U_2} \tag{3.19}$$

bestimmt werden. U_1 und U_2 sind die jeweiligen Spannungswerte der Fotodiode nachgeschalteten Einheiten. Diese Art der Distanzberechnung wird unter anderem in der Kinect v2 eingesetzt. (vgl. [Lau13])

Die andere Methode zur Entfernungsbestimmung bei Time of Flight Kameras erfolgt über die Detektion der Phasenverschiebung des reflektierten Lichtsignals. Hierbei wird das Infrarotlicht kontinuierlich ausgestrahlt und auf ein Trägersignal aufmoduliert. Dieses kann beispielsweise durch eine Amplitudenmodulation realisiert werden, indem man die Intensität periodisch ändert. Die maximale Entfernung ist auch hier abhängig von der Frequenz f_c des Trägersignals und kann basierend auf 3.18 folgend bestimmt werden:

$$D_{max} = \frac{c}{2 * f_c} \tag{3.20}$$

Wird das reflektierte Licht von den Fotodioden detektiert, erfolgt zur Phasenbestimmung eine Autokorrelation mit dem internen Referenzsignal, welches die gleiche Frequenz wie das Trägersignals besitzt. Hierfür werden mindestens vier Messpunkte benötigt. Diese werden wie bei der vorher bereits vorgestellten Methode durch Integration der Spannung ermittelt. Da vier Werte benötigt werden, startet eine neue Integration bei Durchlaufen der Phasenwinkel 0°, 90°, 180° und 270° des Referenzsignales. Mithilfe der Werte $A_1 ... A_4$, kann die Phasenverschiebung ϕ wie folgt ermittelt werden:

$$\phi = \frac{A_1 - A_3}{A_2 - A_4} \tag{3.21}$$

Schließlich wird die Distanz D durch

$$D = \frac{c * \phi}{4 * \pi * f_c} \tag{3.22}$$

bestimmt. Diese Methode hat den Vorteil, dass durch Modulieren des Signales fremde, nichtmodulierte Signale herausgefiltert werden können. Damit eignet sich diese Methode auch außerhalb von geschlossenen Räumen, da in der Sonnenstrahlung infrarote Anteile enthalten sind. Eine Anwendung dieses Verfahren befindet sich beispielsweise im auch in dieser Arbeit verwendeten Lenovo Phab 2 Pro Smartphone [Inf16].
Nachdem die Tiefen berechnet wurden, müssen diese nur noch adäquat gespeichert werden, damit sie zur weiteren Bildverarbeitung verwendet werden können. Dieses geschieht entweder mithilfe einer Tiefenkarte oder einer Punktwolke. Beide sind in Abbildung 3.10 zu sehen. (vgl. [RH07])

Tiefenkarte

Diese Darstellung der Tiefe ähnelt dem Grauwertebild. Die Pixelwerte beschreiben hier oftmals die Distanz in Millimeter und besitzen eine Pixeltiefe von 16 Bit. Somit sind theoretisch Entfernungen bis 65,535 Meter abbildbar. Konnte für einen Bildpunkt keine Tiefe ermittelt werden, wird dieser entweder auf null oder auf NaN[3] gesetzt. Dies passiert häufig bei runden oder spitzen Formen wie Ecken und Kanten, da das Licht nicht wieder zum Sensor, sondern in eine beliebig andere Richtung reflektiert wird. Vorteil dieser Art der Tiefendarstellung ist, dass einem Farbpixel direkt ein Tiefenwert zugeordnet werden kann. Je nach unterschiedlicher Auflösung von Farbbild und Tiefenbild können dabei einem Punkt mehrere Punkte aus dem anderen Bild höherer Auflösung zugeordnet werden.

Punktwolke

Hierbei handelt es sich um eine Ansammlung von gemessenen Tiefenpunkten. Durch bekannte intrinsische Parameter kann jedem Punkt eine x-, y- und z-Koordinate zugewiesen werden. Damit bietet sich die Möglichkeit ein 3D-Modell aus diesen Punkten zu

---

[3] Not a Number - ein nicht definierter Wert

generieren. Doch auch hier existieren Löcher im Modell, da nicht jede Struktur erfasst werden kann. Neben der 3D-Visualisierung kann auch eine Darstellung in 2D erfolgen, indem man den Punkten den jeweiligen Pixelwert der Fotodiode zuordnet, welche durch die Tiefe gemessen hat. Somit lassen sich die Modelle auch in ihrer natürlichen Farbe darstellen, sofern eine zusätzliche RGB-Kamera vorhanden ist.

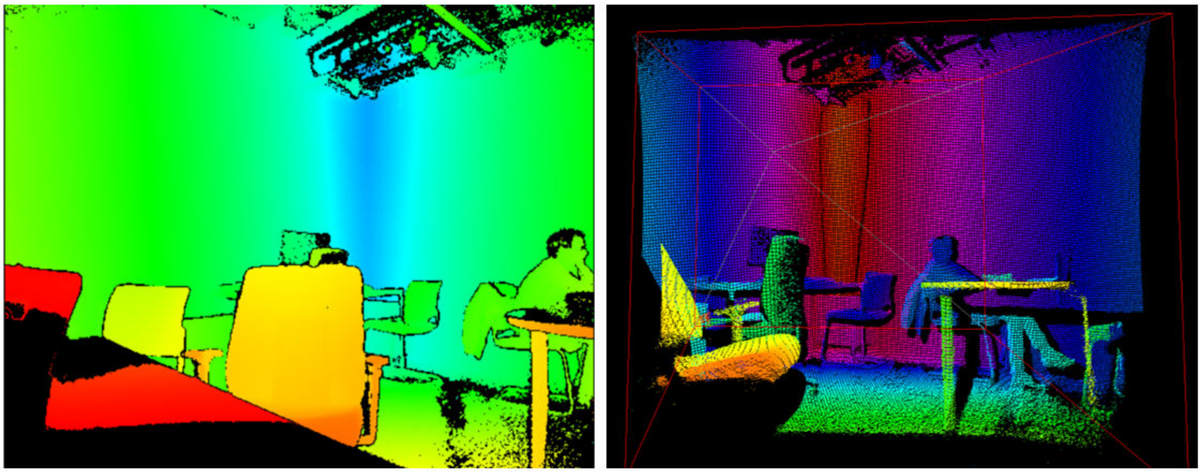


Abbildung 3.10.: Verschiedene Arten der Tiefenvisualisierung, links: Tiefenkarte; rechts: Punktwolke

3.8. Bildverarbeitungsfunktionen

Um den im nächsten Kapitel beschriebenen Algorithmus besser verstehen zu können, werden folgend die wichtigsten Funktionen beschrieben und falls nötig der mathematische Hintergrund erläutert.

3.8.1. Thresholding

Mit dem Thresholding oder auch Schwellwertverfahren kann ein Einkanalbild in eine binäre Maske überführt werden. Dabei wird überprüft, ob ein Pixel größer oder kleiner als ein vorgegebener Schwellwert ist. Bei dem globalen Thresholding T_{glob} gilt für jeden Bildpunkt $s(x, y)$ folgende Vorschrift:

$$s'(x, y) = \begin{cases} 1 & \text{wenn } s(x, y) > T_{glob} \\ 0 & \text{sonst} \end{cases}$$

Otsu-Thresholding

Da ein festdefinierter globaler Schwellwert nicht immer ein Bild optimal in zwei Gruppen teilt, ermittelt die Methode nach Otsu den besten Threshold für jedes Bild einzeln aus dessen Histogramm. Dieses ist eine zweidimensionale Darstellung der Häufigkeiten der jeweiligen Bitwerte von null bis zum Maximalwert. Um den optimalen Schwellwert bestimmen zu können, muss man die Intraklassen-Varianz σ_w^2 minimieren. Dies bedeutet, dass die Summe der Varianzen σ_0^2, sowie σ_1^2 für die jeweiligen Klassen minimal werden muss. Die Berechnung geschieht für jeden möglichen Thresholdwert T_k und sieht folgendermaßen aus:

$$\sigma_w^2 = P_0(T_k) * \sigma_o^2 + P_1(T_k) * \sigma_1^2 \tag{3.23}$$

wobei:

$$\sigma_0^2 = \sum_{i=0}^{T} (H(x) - \mu_0)^2 \tag{3.24}$$

$$\sigma_1^2 = \sum_{T}^{M*N} (H(x) - \mu_1)^2 \tag{3.25}$$

, sowie:

$$P_0(T) = \frac{1}{M * N} * \sum_{i=0}^{T} H(i) \tag{3.26}$$

$$P_1(T) = \frac{1}{M * N} * \sum_{i=T}^{M*N} H(i) \tag{3.27}$$

P gibt die Wahrscheinlichkeit an, dass ein Histogrammwert H(x) in der jeweiligen Gruppe liegt. Diese wird ermittelt, indem man die Anzahl der Pixel einer Klasse durch die Gesamtanzahl teilt. Die Intraklassen-Varianzen werden durch das Quadrat der Differenz von jedem Histogrammwert einer Klasse zum Klassenmittelwert μ_0 oder μ_1 gebildet. Analog dazu existiert die Interklassen-Varianz μ_b^2. Diese trifft eine Aussage darüber, wie weit die jeweiligen Klassenmittelwerte vom Gesamtmittelwert entfernt sind. Da

$$\sigma^2 = \sigma_w^2 + \sigma_b^2 \tag{3.28}$$

gilt und σ^2 für jedes Histogramm eine Konstante ist, kann der optimale Schwellwert durch ein maximales σ_b^2 bestimmt werden, weil σ_w^2 an dieser Stelle am minimalsten ist. Die Interklassen-Varianz wird wie folgt berechnet:

$$\sigma_w b^2 = P_0(T_k) * (\mu_0 + \mu)^2 + P_1(T_k) * (\mu_1 + \mu)^2 \tag{3.29}$$

Durch das Einsetzen von

$$mu = P_0(T_k) * \mu_0 + P_1(T_k) * \mu_1 \tag{3.30}$$

erhält man schließlich folgende Gleichung:

$$\sigma_b^2 = P_0(T_k) * P_1(T_k) * (\mu_0 - \mu_1)^2 \tag{3.31}$$

Der gesuchte Schwellwertindex k_opt ist somit

$$k_{opt} = \max_{0<k<M*N} \sigma_b^2(k) \tag{3.32}$$

Da das Verfahren mithilfe der Interklassenvarianz nur Mittelwerte zur Bestimmung von k_opt benötigt, ist dieses wesentlich schneller als die Berechnung mittels Intraklassen-Varianzen.

3.8.2. Canny Edge Operator

Die Methode nach Canny wird häufig zur Kantendetektion verwendet, da es durch verschiedene Schritte eine zuverlässige Kantendetektion ermöglicht. Als Grundlage hierfür dient meist ein Grauwertbild, welches mithilfe der unterschiedlichen Helligkeitswerte in ein binäres Bild, welches die berechneten Kanten enthält, überführt wird. Als Vorverarbeitungsschritt erfolgt eine Glättung mittels eines Gaußfilters. Dieser entfernt Rauschen aus dem Bild, welches sonst aufgrund der hohen Differenz des Bitwertes zu Nachbarschaftspixeln als Kante detektiert werden könnte. Als nächster Schritt erfolgt eine Faltung mittels Sobel-Operator. Wie in Abbildung 3.11 zu sehen, existiert dieser für die x-, als auch y-Richtung und bestimmt die erste Ableitung zu den seitlichen, beziehungsweise oberen und unteren Pixelnachbarn.

Mithilfe der so ermittelten Gradienten $I_x(u, v)$ und $I_y(u, v)$ zu den jeweiligen Nachbarpixeln können für jeden Bildpunkt (u,v) der Anstieg $M(u, v)$ und die Orientierung $\theta(u, v)$ durch folgende Formeln berechnet werden:

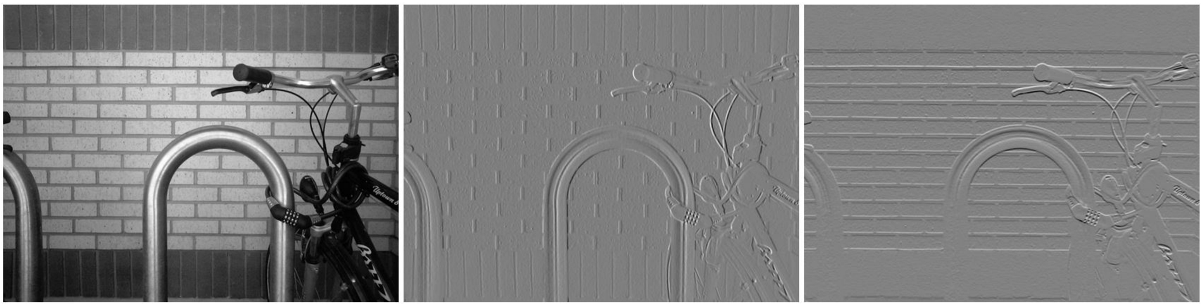


Abbildung 3.11.: Anwendung des Sobel-Operators (Quelle: [wik07]), links: Referenzbild; Mitte: Sobel-Operator in x-Richtung; rechts: Sobel-Operator in y-Richtung

$$M(u, v) = \sqrt{I_x^2(u, v) + I_y^2(u, v)} \tag{3.33}$$

$$\theta(u, v) = \mathrm{atan2}(\frac{I_y(u, v)}{I_x(u, v)}) \tag{3.34}$$

Da ein Pixel in seiner Umgebung nur acht Nachbarn besitzt, werden die Orientierungen auf ganzzahlige Vielfache von 45° gerundet. Da der Wertebereich nur von 0° bis ausschließlich 180° geht, kann die Orientierung somit nur vier unterschiedliche Werte annehmen. Die eindimensionale Kante besitzt damit die vier verschiedenen Ausrichtungen: Nord-Süd, Nordost-Südwest, Ost-West, Südost-Nordost mit jeweils unterschiedlicher Magnitude. Im nächsten Schritt folgt die sogenannte *non-maximum suppression*[4]. Hierbei wird tangential entlang der Kante der maximale Steigungswert gesucht. Alle anderen Werte werden auf null gesetzt. Damit wird die Kante auf eine Pixelbreite mit dem Maximalwert reduziert. Zuletzt erfolgt die Kantenbestimmung mittels Hysterese. Die Pixel

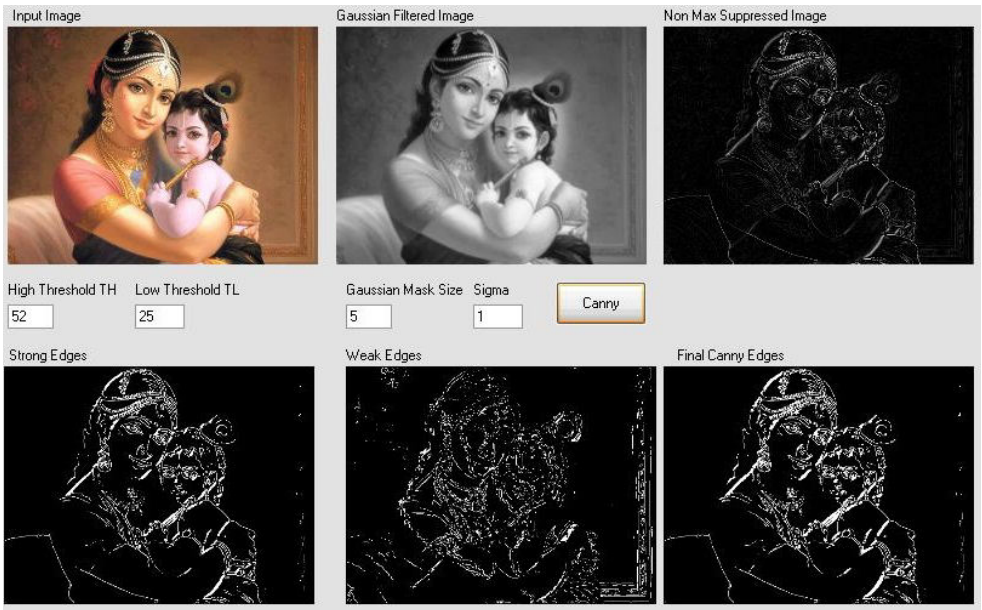
---

[4] dt. Non-Maxima-Unterdrückung

werden mittels einem unterem Schwellwert T_l, sowie einem oberen Schwellwert T_h auf folgende Arten eingeteilt:

- $M > T_l \rightarrow$ akzeptiere Pixel (starke Kante)
- $M < T_l \rightarrow$ verwerfe Pixel und setzte ihn auf null
- $T_l < M < T_h \rightarrow$ akzeptiere Pixel wenn er mit anderem Pixel verbunden ist, auf den erster Punkt zutrifft, sonst verwerfe ihn (schwache Kante)

Der letzte Punkt ist so zu verstehen, dass falls ein Pixel mit einem Anstieg über T_h gefunden wurde, die dementsprechende Kante entlang gegangen wird und alle Pixel mit einer Magnitude größer dem unteren Schwellwert auch akzeptiert werden. Falls eine Kante keinen Bildpunkt mit großem Anstieg besitzt, werden alle Punkte verworfen. Zum besseren Verständnis werden alle Schritte des Canny Edge Algorithmus nachfolgend noch einmal anschaulich dargestellt.



High Threshold TH: 52
Low Threshold TL: 25
Gaussian Mask Size: 5
Sigma: 1

Abbildung 3.12.: Schritte des Canny Edge Operators (Quelle: [Bha12])

3.8.3. Dilatation

Die Dilatation ist neben der Erosion eine der wichtigsten morphologischen Transformationen. Diese arbeiten (in OpenCV) auf Grundlage von Binärbildern und ändern die Gestalt der weißen Pixelanordnungen. Im konkreten Fall der Dilatation wird ein binäres Objekt vergrößert, wie man es in Abbildung 3.13 erkennen kann. Notwendig ist diese Operation beispielsweise, nachdem die vorher erläuterte Kantendetektion durchgeführt wurde. Da aufgrund von Rauschen oder nicht genau definierten Strukturen Teile der Kante nicht als solche detektiert werden, kann ein eigentlich zusammengehörendes Objekt im Bild durch kleine Löcher zerstückelt sein. Durch die Dilatation schließen sich diese Lücken und es

kann wieder eine zusammenhängende Kontur erkannt werden (siehe folgenden Abschnitt 3.8.4).
Bildlich lässt sich die Funktionsweise der Dilatation wie folgt darstellen. Es wird ein sogenannter Kernel, in diesem Fall eine nxn Matrix gefüllt mit Einsen, über jeden Pixel des zu bearbeitenden Bildes I geschoben. Für jeden überlappten Bildpunkt im Bereich um $I(u, v)$ wird nun geschaut, ob ein Pixel ungleich null vorliegt. Falls dies der Fall ist wird der aktuelle Überlappungsbereich auf die Form des Fensters ausgedehnt, was einer logischen Oder-Verknüpfung entspricht. Danach wird das Fenster zum nächsten Pixel geschoben. Mathematisch gesehen wird die Dilatation von zwei Mengen A und B um den Faktor b wie folgt beschrieben:

$$A \oplus B := \cup_{b \in B}(A + b) \tag{3.35}$$

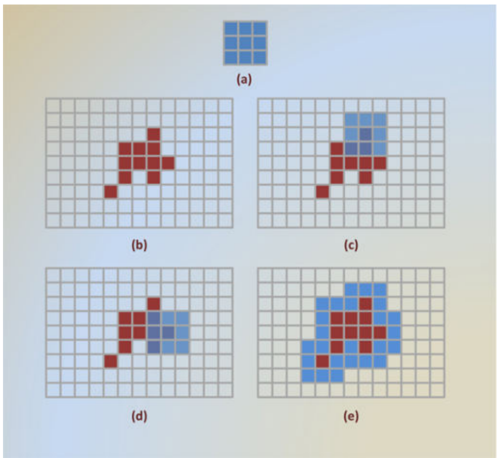


Abbildung 3.13.: Prinzip der Dilatation, rot = Ausgangspixel, blau = Kernel, sowie durch Dilatation hinzugefügte Pixel (aus: [MHST09])

3.8.4. OpenCV-Funktion: findContours

Diese OpenCV-Funktion sucht in einem binären Bild nach zusammenhängenden Konturen. Hierbei wird überprüft, ob einer der acht umliegenden Werte eines Vordergrundpixels nicht null ist. Falls dies zutrifft, fährt man mit diesen Punkten solange fort, bis keine validen Nachbarn mehr vorhanden sind. Danach wird die Kontur abgespeichert und das Bild wird weiter nach zusammenhängenden Punkten abgesucht. Bereits betrachtete Punkte werden dabei übersprungen. Weiterhin können mithilfe verschiedener Parameter unterschiedliche Einstellung getroffen werden. Zum Beispiel kann die Kontur im Speicher komprimiert werden, indem statt alle Punkte zu speichern nur die Eckpunkte gespeichert werden. Eine weitere wichtige Einstellmöglichkeit ist die Art der Etablierung einer Hierarchie. Dabei sind folgende Parameter möglich:

- CV_RETR_EXTERNAL
- CV_RETR_LIST

- CV_RETR_CCOMP
- CV_RETR_TREE

Mit der ersten Einstellung werden nur die äußersten Konturen gespeichert. Das heißt, alle Konturen die sich innerhalb einer Größeren befinden werden ignoriert. Dies dient später dazu, das Türschild einer Raumnummer zu ermitteln. Mit CV_RETR_LIST hingegen wird keine Hierarchie gebildet, alle Konturen sind nicht miteinander verknüpft. Dies sollte normalerweise die Standardeinstellung sein, um Speicher- und Rechenaufwand zu minimieren. Die letzten beiden Einstellungen werden zwar nicht in dieser Arbeit verwendet, sollen jedoch trotzdem der Vollständigkeit halber vorgestellt werden. Beim dritten Parameter wird eine Zweier-Hierarchie aufgebaut. Eine innere Kontur würde dann an zweiter Stelle unter der äußeren Umrandung stehen. Da es eine dritte Ebene nicht mehr gibt, würden weitere innere Konturen abwechselnd an erster und zweiter Stelle gespeichert werden. Mit der letzten Einstellung CV_RETR_TREE wird ein vollständiger Hierarchiebaum gebaut. Dies bedeutet, dass für jede Kontur sofort ermittelt werden kann, wie viele und welche Eltern - und Kinderkonturen es gibt.

3.8.5. Support Vector Machine

Ein wichtiger Bestandteil der Raumschilderkennung in dieser Arbeit ist die richtige Identifikation der Buchstaben und Ziffern. Eine bewährte Methode hierfür ist die Support Vector Machine (SVM). Als eine Untergruppe der Machine Learning-Algorithmen teilt die SVM ausgehend von einem Trainingsdatensatz den Eingang in zwei verschiedene Klassen ein. Die Entscheidungsfunktion sollte dabei so gewählt werden, dass der Abstand zwischen den beiden Gruppen maximal wird. Zur Ermittlung dieser Funktion werden Trainingsdaten benötigt, bei denen die Einteilung in die jeweilige Klasse bereits bekannt ist. Die Zuordnung eines Elements in die richtige Klasse wird auch als Label bezeichnet. Ausgehend von den Trainingsdaten können die sogenannten Stützvektoren jeder Klasse bestimmt werden. Wie in Abbildung 3.14 zu sehen ist, sind die Stützvektoren diejenigen Objekte, welche die geringste Distanz zu der Trennebene und somit anderen Klasse besitzen. Durch die in [Bil14] beschriebenen Schritte kann eine solche Entscheidungsfunktion ermittelt werden, die eine größtmögliche Marge zwischen den jeweiligen Stützvektoren besitzt. Damit ist gewährleistet, dass auch verrauschte Eingänge der richtigen Klasse zugeordnet werden. Da in der Realität die wenigsten Objekte linear separierbar sind, muss eine Erweiterung gefunden werden, damit die SVM solche nichtlinearen Entscheidungsprobleme lösen kann. Lösungsansätze hierfür sind sogenannte Soft-Margins, welche einzelne Missklassifizierungen erlauben oder die Überführung des nichtlinearen Problems in einen höherdimensionalen Raum, in welchem dieses linear separierbar ist. Für weiterführende Informationen sei erneut auf [Bil14] verwiesen.

Da in der Texterkennung zwischen vielen verschiedenen Buchstaben und Ziffern unterschieden werden muss und die SVM ein binärer Klassifikator ist, werden abschließend zwei mögliche Ansätze zur nicht binären Einteilung erläutert werden. Dies ist zum einen die *one-against-all* und zum anderen die *one-against-one* Strategie. Bei der *one-against-all* Methode wird für jede Klasse eine eigene SVM kreiert. Zur Erstellung der Trennfunktion werden alle der jeweiligen Klasse zughörigen Trainingsdaten den positiven Labels

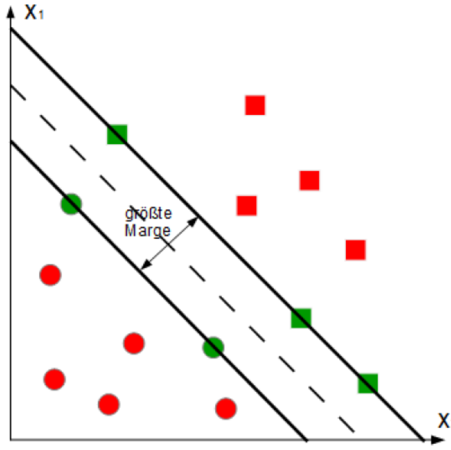


Abbildung 3.14.: Lineare Trennung zweier Klassen mit maximaler Marge. Die gestrichelte Linie stellt die Hyperebene dar. Die äußeren beiden durchgezogenen Linien sind die Klassengrenzen. Alle Stützvektoren wurden grün eingezeichnet. (aus: [Bil14])

und sämtliche andere Beispiele den negativen Labels zugeordnet. Somit wird eine Klasse gegen den gesamten Rest trainiert. Ein großer Nachteil dieses Ansatzes ist, dass durch die unterschiedliche Anzahl der Labels auf jeder Seite ein Ungleichgewicht entsteht und die eigentliche Klasse unterrepräsentiert ist. Daher ist es möglich, dass ein Eingang immer dem Rest zugeteilt wird und keine Klassifizierung stattfinden kann. Um dieses zu umgehen, nutzt OpenCV beispielsweise die *one-against-one* Methode [Lor16] . Hierbei wird jede Klasse einzeln gegen jede andere trainiert. Es entstehen also $k\frac{k-1}{2}$ unterschiedliche SVMs, wobei k die Anzahl der verschiedenen Gruppierungen darstellt. Soll nun ein Objekt richtig klassifiziert werden, wird der Eingang in jede Support Vector Maschine gegeben. Es gewinnt schließlich die Klasse, welche die meisten Zuordnungen für sich entscheiden konnte. Weiterführende Informationen zur Multiklassen-Klassifikation finden sich in [CY11].

3.8.6. OpenCV-Funktion: solvePnP

Mithilfe von solvePnP lassen sich sogenannte Perspective-n-Point Probleme lösen. Hierbei wird versucht die Pose[5] von einem 3D-Weltpunkt zu einer projektiven 2D-Abbildung auf der Bildebene zu finden. Um diese bestimmen zu können, muss bei einer gegebenen intrinsischen Matrix die korrekte extrinsische Matrix berechnet werden. Für jede Projektion eines Punktes wird entsprechend [Ope14c] das Gleichungssystem

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} k_u f & 0 & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{21} & R_{31} & T_1 \\ R_{12} & R_{22} & R_{32} & T_2 \\ R_{13} & R_{23} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.36}$$

---

[5] bezeichnet in der Bildverarbeitung die Entfernung und Orientierung eines Punktes zu einem Referenzpunkt

aufgestellt. Um ein besseres Ergebnis zu erzielen, sollten statt der hier verwendeten Bildpunkten u und v sollten die entzerrten Pixelkoordinaten u' und v' verwendet werden (siehe 3.5.3). Zur Bestimmung der sechs Freiheitsgrade der extrinsischen Matrix, müssen mindestens drei Punktkorrespondenzen gefunden werden, da jede Korrespondenz jeweils zwei linear unabhängige Gleichungen besitzt. Oft wird zusätzlich eine vierte Gleichung gefordert, um mehrdeutige Lösungen zu verhindern. Das Gleichungssystem wird nun mithilfe eines Verfahrens zur Lösung nichtlinearer Ausgleichsprobleme, wie dem in 3.4 vorgestellten LM-Verfahren gelöst. Hierbei werden die 3D-Weltpunkte durch Verändern der Rotation und Translation solange auf die Bildebene projiziert, bis der Fehler zu den vorliegenden Bildpunkten minimal wird. Als Lösung werden schließlich die Rotationsmatrix, sowie der Translationsvektor ausgegeben.

3.8.7. Rodrigues-Formel

Als Ergebnis der solvePnP wird unter anderem ein Rotationsvektor der Form $[r_x r_y r_z]^T$ geliefert. Dieses ist die kompakteste Darstellung für eine dreidimensionale Rotation, da die drei Werte implizit die Rotationsachse r und den Drehwinkel in mathematisch positiver Richtung θ enthalten. Bestimmt werden diese durch:

$$\theta = \mathrm{norm}(r) \tag{3.37}$$

$$r = \frac{r}{\mathrm{norm}(r)} \tag{3.38}$$

Da die Norm des Vektors keinen Einfluss auf die Richtung der Drehachse r hat, lässt sich der Drehwinkel über die Länge des Vektors einfach realisieren. Für weitere Berechnungen muss diese Darstellung jedoch oft wieder in eine Rotationsmatrix konvertiert werden. Die Gleichung zur Umwandlung lautet wie folgt:

$$R = \cos(\theta) * I + (1 - \cos(\theta)) * rr^T + sin(\theta) * \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{3.39}$$

(vgl.[AGD07, S. 643–644]).

3.9. Geografisches Koordinatensystem

Mithilfe des geografischen Koordinatensystems kann ein beliebiger Punkt auf der Erdoberfläche genau definiert werden. Die Koordinaten werden mithilfe von Längen- und Breitengraden angegeben, welche im Englischen auch als Longitude, beziehungsweise Latitude bezeichnet werden. Wie in Abbildung 3.15 zu sehen ist, erfolgt die Beschreibung der Punkte durch Gradangaben. In horizontaler Richtung wird die Erde durch die Längenkreise in 360° unterteilt, wobei die Nulllinie durch die britische Gemeinde Greenwich verläuft. In dieser Arbeit werden alle Punkte östlich der Nulllinie bis zur gegenüberliegenden Linie (180°) positiv und die westlich liegenden Punkte negativ gezählt. Die Unterteilung der Längengrade geschieht ausgehend vom Äquator in Richtung der Pole.

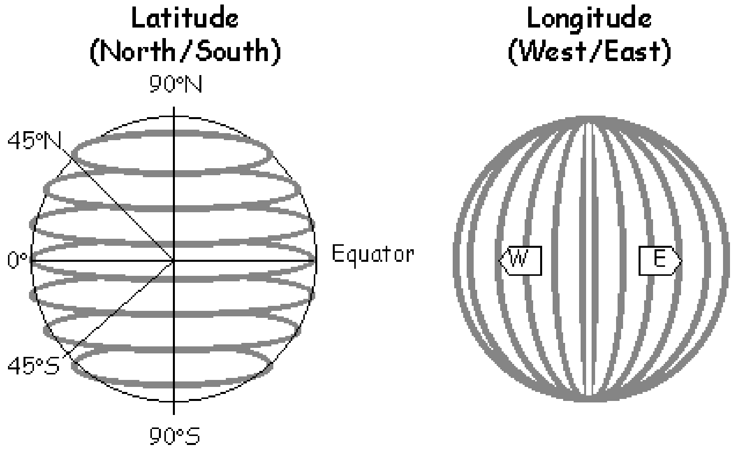


Abbildung 3.15.: links: Längengrade rechts: Breitengrade, (Quelle: [wik17b])

Hierbei liegt der Nordpol bei 90° und der Südpol bei -90°. Durch beliebig viele Nachkommastellen kann ein Punkt so genau wie nötig beschrieben werden (Beispiel: 50,371623°, 10,1452964°).
Da eine Distanz zwischen zwei Punkten meist in metrischen Werten angegeben wird, müssen die geografischen Werte umgewandelt werden. Die Entfernung von einem Punkt $P_1(\text{lon}_1, \text{lat}_1)$ zu $P_2(\text{lon}_2, \text{lat}_2)$ in y-, also vertikaler Richtung bestimmt sich aus

$$dy = 111,13 * (\text{lat}_1 - \text{lat}_2), \tag{3.40}$$

wobei die Konstante 111,13 dem Abstand zwischen zwei Breitenkreisen in Kilometern angibt. Dieses entspricht dem einhundertachtzigsten Teil des halben Erdumfangs von Pol zu Pol[6]. Die horizontale Differenz dx lässt sich mithilfe von

$$dx = 111,32 * cos(\text{lat}) * (\text{lon}_1 - \text{lon}_2). \tag{3.41}$$

Bei dieser Umformung ist zu beachten, dass die Abstände zwischen den Längenkreisen ausgehend vom Äquator in Richtung der Pole immer kleiner werden. Somit ist dx abhängig vom gemittelten Längengrad der beiden Punkte. Weiterhin unterscheidet sich die Umformungskonstante $111,32$ leicht zur Vorherigen, da die Erde keine perfekte Kugel ist, sondern eher einem Rotationsellipsoid entspricht. Beide Werte dx und dy liegen nach der Umwandlung in Kilometern vor. Abschließend ist noch anzumerken, dass diese Methode der Umwandlung nur für relativ kleine Werte hinreichend genau ist, da von einem rechtwinkligen und nicht der Erdoberfläche entsprechenden kugelförmigen Koordinatensystem ausgegangen wird.

---

[6] entspricht circa 20.003,9 km [wik17a]

4. Durchführung

4.1. Überblick

Der hier vorgestellte Algorithmus wurde in C++ geschrieben und verwendet zur Anbindung an das Lenovo Smartphone die Programmiersprache Java. Des Weiteren wurde für sämtliche Bildverarbeitungsfunktionen die frei verfügbare Bibliothek OpenCV verwendet. Daneben wurde aus der Boost Bibliothek der XML-Parser und das Filesystem verwendet. Schließlich fand die SQL-Erweiterung Spatialite Anwendung, um die Datenbankabfrage um geografische Zusatzfunktionen erweitern zu können.
Der Gesamtablauf des Programmes ist auf der folgenden Seite in Abbildung 4.1 dargestellt. Nach der Initialisierung, in der unter anderem die Pfade zu Datenbank und SVM-Parameter übergeben werden, startet die Bilderfassung. Diese erfolgt je nach Umgebung mit der Kinect, dem Phab 2 Pro oder durch das Laden einer abgespeicherten Bildsequenz. Die Rohdaten werden in eine OpenCV Matrix und falls nötig in das RGB Format umgewandelt. Nach dieser Vorverarbeitung erfolgt die parallele Detektion von möglichen ArUco-Markern oder Raumschildern. Falls eines dieser Objekte gefunden wurde, wird überprüft, ob ein dementsprechender Eintrag in der Datenbank vorliegt. Mithilfe der in der Datenbank gespeicherten geometrischen Maße, kann die optische Distanzermittlung mittels der OpenCV-Funktion *solvePnp* erfolgen. Schließlich wird überprüft, ob zu Mittel- oder Eckpunkten des Markers Tiefendaten vorliegen. Diese können zu einer Korrektur der bisher ermittelten Entfernungswerte verwendet werden. Zum Schluss wird überprüft, ob innerhalb der letzten n Frames das gefundene Objekt oft genug detektiert wurde, um durch Mittelwertbildung etwaige Schwankungen beseitigen zu können. Sollte dies der Fall sein, wird eine Position in Längen- und Breitengrad ausgegeben, indem man zu der in der Datenbank eingetragenen Position die ermittelte Distanz zum Objekt hinzu addiert.

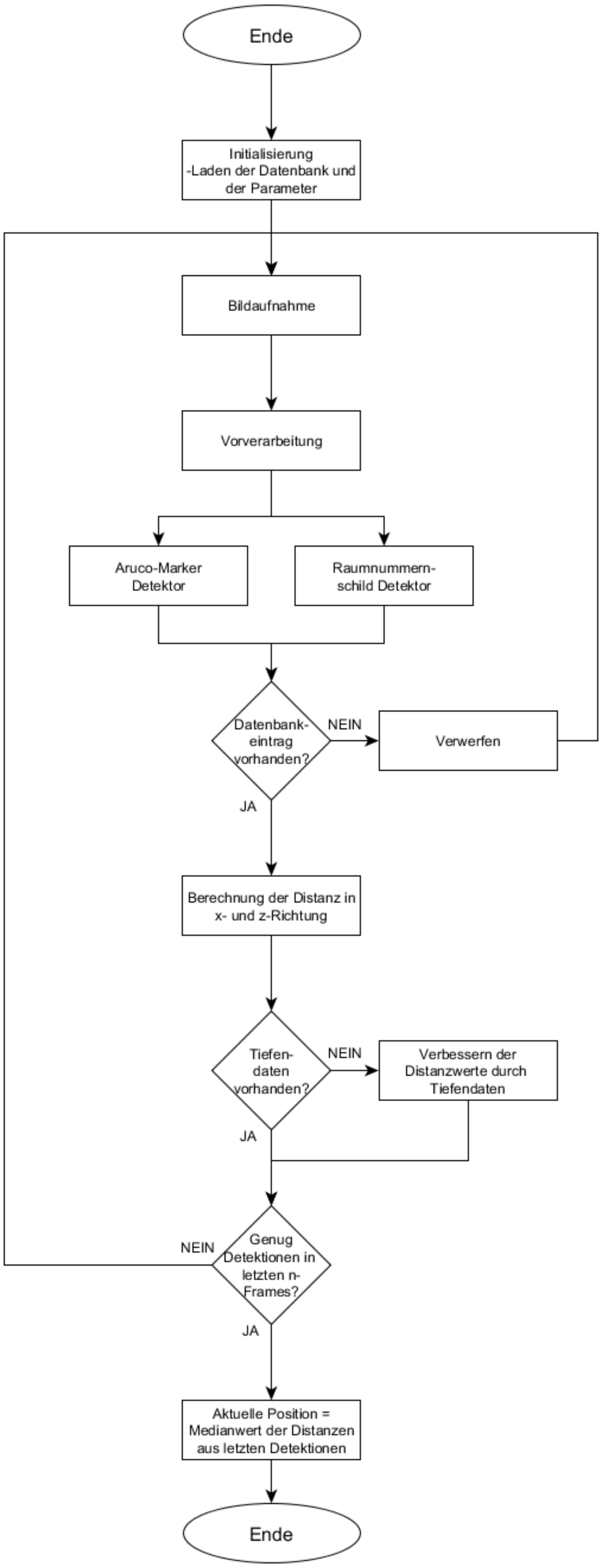


Abbildung 4.1.: Ablaufplan des Positionierungsalgorithmus

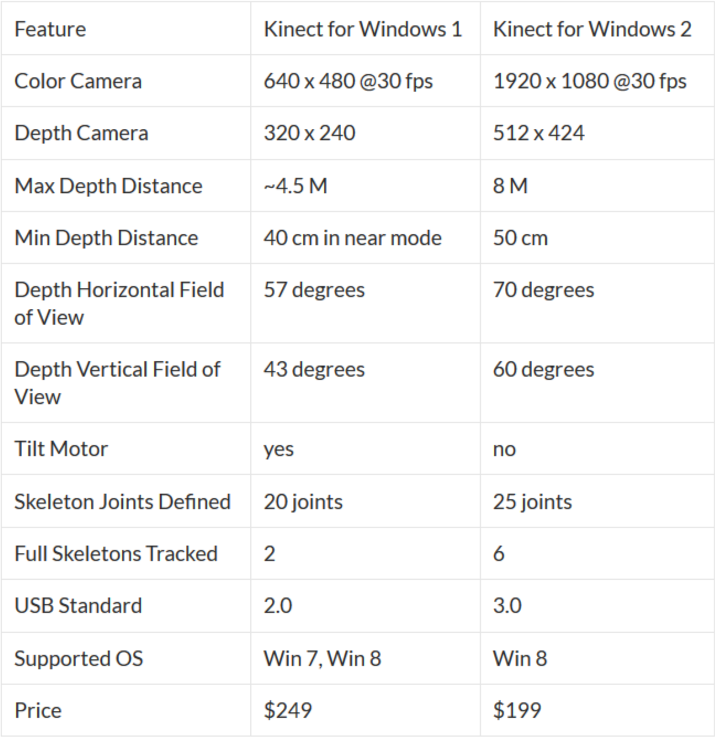
4.2. Verwendete Hardware

Nachfolgend werden die beiden zur Bildaufnahme verwendeten Geräte näher vorgestellt. Zum einen ist dies eine Micorosoft Kinect der zweiten Generation, auf welcher die Validierung des Algorithmus vorrangig stattfinden soll. Zum anderen wurde das Lenovo Phab 2 Pro verwendet, um die Funktionalität des Positionsalgorithmus auf einem Android Smartphone überprüfen zu können. Zu sehen sind beide Geräte in nachfolgender Grafik 4.2.

Abbildung 4.2.: links: Microsoft Kinect v2 (Quelle: [wik14]); rechts: Lenovo Phab 2 Pro (Quelle: [Len17])

4.2.1. Microsoft Kinect v2

Die in dieser Arbeit verwendete Kinect ist die zweite Version der von Microsoft entwickelten Kameras. Sie wird seit 2014 verkauft und wurde vorrangig zur Ergänzung der Spielekonsole XBox One als Eingabegerät entwickelt. Durch einen zusätzlichen Adapter kann dieses Gerät via USB 3.0 auch an den PC angeschlossen werden. Die Kinect liefert sowohl ein RGB-, als auch ein Tiefenbild. Das Farbbild liegt in Full HD Auflösung vor und wird 30-mal pro Sekunde aktualisiert. Das Tiefenbild hingegen besitzt nur eine Auflösung von 512x424, welches jedoch aufgrund der eingesetzten ToF Technik zur Tiefengewinnung ein sehr guter Wert ist. Der messbare Bereich liegt bei 0,5 bis 4,5 Meter. Alle technischen Details werden noch einmal in folgender Grafik 4.3 dargestellt und zusätzlich mit der vorherigen Kinect Version verglichen.



Feature	Kinect for Windows 1	Kinect for Windows 2
Color Camera	640 x 480 @30 fps	1920 x 1080 @30 fps
Depth Camera	320 x 240	512 x 424
Max Depth Distance	~4.5 M	8 M
Min Depth Distance	40 cm in near mode	50 cm
Depth Horizontal Field of View	57 degrees	70 degrees
Depth Vertical Field of View	43 degrees	60 degrees
Tilt Motor	yes	no
Skeleton Joints Defined	20 joints	25 joints
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0
Supported OS	Win 7, Win 8	Win 8
Price	$249	$199

Abbildung 4.3.: Vergleich von Kinect v1 und Kinect v2 (Quelle: [Ash14])

4.2.2. Lenovo Phab 2 Pro

Das von Lenovo entwickelte Smartphone ist seit 2016 auf dem Markt erhältlich und das erste Consumer-Gerät, welches Googles Tango-Technik (siehe 2) benutzt. Dafür besitzt es neben einer normalen RGB-Rückkamera weiterhin einen ToF-Tiefensensor und eine Fisheye-Kamera. Somit werden zusätzlich zu den 1920x1080 Pixel auflösenden Farbbildern Tiefeninformationen mittels einer Punktwolke geliefert. Diese Daten werden aus dem Real3 ToF-Sensor von Infineon/PMD generiert, welcher eine Auflösung von 224 x 172 Pixel bei einer Aktualisierungsrate von bis zu 100 Frames pro Sekunde bietet. Jedoch ist die Aktualisierungsrate, mit welcher die Punktwolken geliefert werden, wesentlich geringer, da für eine bessere Genauigkeit der Daten mehrere Frames fusioniert werden. Aus diesem Grund wird auch die Pulsrate des Lasers für nacheinander folgende Bilder geändert, um möglichst genaue Tiefeninformationen bei gleichzeitig hoher Reichweite zu ermöglichen. Schließlich wäre ausgehend von Gleichung 3.18 bei der Standardmodulationsfrequenz von 100 Megahertz und somit einer Pulsbreite von 10 Nanosekunden die maximale Reichweite bei gerade einmal 1,5 Metern. Bei Verringerung der Frequenz sind dementsprechend weiter entfernte Texturen bei geringerer Genauigkeit verfügbar. Mithilfe der Datenfusion lässt sich schließlich eine Punktwolke aus den unterschiedlichen Tiefenbildern generieren. (vgl. [Mö16])

4.3. Anbindung an die Geräte

Um mit der Kinect kommunizieren zu können, stellt Microsoft ein SDK für Windows zur Verfügung. In diesem sind sowohl der Treiber, als auch alle notwendigen Funktionen enthalten die zur Datengewinnung benötigt werden. Dabei hat man eine Auswahl zwischen folgenden Quellen, welche die Kinect bereitgestellt:

- RGBA-Bild[1]
- Tiefenkarte
- Infrarotbild
- Skelett mit Gelenkpunkten[2]
- Audioquelle

Die erforderlichen Schritte zur Bildgewinnung werden in 4.4.2 beschrieben.
Die Entwicklung für das Lenovo Phablet fand in der Entwicklungsumgebung *Android Studio* statt. In dieser ist bereits das Android SDK, welches alle Android spezifischen Funktionen bereitstellt und die Android Debug Bride (ADB) integriert. Letzteres wird zur Kommunikation mit dem Smartphone via USB benötigt und erlaubt zusätzlich das Debuggen in der IDE. Um den in C++ geschriebenen Positionierungsalgorithmus auf Android-Geräten[3] verwenden zu können, muss zusätzlich das Native Developer Toolkit (NDK) installiert werden. Dieses bietet neben dem *Clang* Compiler einen Debugger, welcher beispielsweise das Setzen und Ausführen von Breakpoints im C++ Code erlaubt. Der Aufruf einer nativen (C++) Funktion von Java aus erfolgt mithilfe des Java Native Interfaces (JNI). Dieses übergibt dem nativen Code neben dem aktuellen Java Objekt[4] einen Java-Enviroment Pointer, damit auch in der C++ Umgebung alle Funktionen der virtuellen Java Maschine (JVM) zur Verfügung stehen. Dieses sind zum Beispiel die Umwandlung von Javatypen in native Typen oder das Auslösen von Exceptions. Das folgende Beispiel

```
void SynchronizationApplication::OnCreate(JNIEnv* env, jobject activity)
```

zeigt eine JNI Funktionsdeklaration. Eine ausführlichere Erläuterung des Java Native Interface ist unter [Mik06] zu finden.

---

[1] RGBA ist ein RGB-Bild mit einem zusätzlichen vierten Alpha-Kanal, welcher die Transparenz des Pixels beschreibt

[2] wird automatisch extrahiert wenn Person erkannt wurde, siehe [Mic13]

[3] die Programmiersprache für Android Applikationen ist Java

[4] Objekt, welches durch das JNI die C++ Funktion aufruft

4.4. Beschreibung der Module

Ausgehend von der Erstinitialisierung werden nachfolgend alle wichtigen Zwischenschritte bis hin zur Endpositionierung näher erläutert. Ein besonderes Augenmerk liegt dabei auf der Raumnummererkennung, die einen großen Teil der Eigenimplementierung ausmacht. Sofern es erforderlich ist, wird weiterhin bei jedem Punkt auf die unterschiedliche Realisierung in der Android- beziehungsweise Windows-Umgebung eingegangen.

4.4.1. Start und Initialisierung

Abhängig von der Plattform auf der die Positionsbestimmung betrieben werden soll, sind auch verschiedene Vorgehensweisen beim Start zu berücksichtigen. Während bei der Android Anwendung nichts zu beachten ist, muss in der Desktop Umgebungen das Programm via Kommandozeile mit verschiedenen Argumenten gestartet werden. Dabei werden die folgenden drei verschiedenen Möglichkeiten geboten:

- **Bildquelle Kinect** : -k <Pfad Kalibrierdaten> <Pfad Datenbank> <Pfad SVM>
- **Bildquelle Datei (Video/Bildfolge)** : -f <Dateipfad> <Pfad Kalibrierdaten> <Pfad Datenbank> <Pfad SVM>
- **Training SVM** : -t <Dateipfad>
- **Hilfe** : -help

Alle Pfade müssen den Dateinamen und die Dateiendung inkludieren. Die ersten beiden Punkte werden nachfolgend behandelt, währenddessen auf das Trainieren der SVM später in 4.5 eingegangen wird. Mit der Option "Hilfe" werden schließlich alle Möglichkeiten mit den entsprechenden Argumenten angezeigt. Zusätzlich sei noch eine Anmerkung zu den Kalibrierdaten gemacht. Weil diese in der Desktop Umgebung aus einer Datei ausgelesen werden, müssen diese vor Ausführung des Algorithmus ermittelt werden. Da dieses jedoch nicht Bestandteil der Arbeit ist, wird zur Bestimmung der Kameraparameter auf [AGD07, S. 78–80] verwiesen. Nach diesem Algorithmus wurden auch die Parameter der Kinect ermittelt. Dies gilt jedoch nur für die Desktop Umgebung, da für Android Geräte die intrinsischen Parameter, sowie Verzerrungskoeffizienten bereits im Gerät vorliegen.

Wird nun die Kinect als Bildquelle ausgewählt und die jeweiligen Pfade als Argument übergeben, erfolgt die Extraktion der benötigten Informationen aus den Dateien. Die Kalibrierdaten werden mit Hilfe des Boost XML-Parser ausgelesen. Da dieser nach bestimmten Tags sucht, muss diese Datei eine festgelegte Struktur besitzen. Ein Beispiel findet sich unter B im Anhang, wo das in dieser Arbeit genutzt XML-File mit den Kalibrationswerten abgebildet ist. Mithilfe der ausgelesenen Daten können schließlich die intrinsische Matrix, sowie die Verzerrungskoeffizienten in einer *cv::Mat*[5] gespeichert werden. Außerdem muss die Datenbank in einem SQL[6] kompatiblen Format wie *.sqlite* oder

---

[5] der Standardcontainer von OpenCV für Matrizen wie Bilder, Gleichungssysteme et cetera

[6] Structured Query Language - Programmsprache zum Lesen, Schreiben und Manipulieren von Datenbanken

.sql vorliegen. Der Pfad wird folgend der DatenbankHandler-Klasse übergeben. Diese öffnet die Datenbank und erweitert die SQL-Befehle um den SpatiaLite Befehlssatz. Somit wird zum Beispiel das Extrahieren der Koordinaten eines bestimmten Objektes oder das Anzeigen aller Einträge ermöglicht, welche sich in einem bestimmten Umkreis einer definierten Position befinden. Schlägt das Öffnen der Datei fehl, da beispielsweise ein falscher Pfad angegeben wurde, wird eine Fehlermeldung ausgegeben.

Das letzte Argument dient schließlich zur Initialisierung der SVM in der Raumschilderkennung. In dieser XML-Datei sind neben den Stützvektoren auch die Anzahl der Klassen und die Parameter zur Klassifizierung enthalten. Außerdem wird der Raumschilderkennung eine Mindest- und Maximalanzahl der Zeichen eines Schildes übergeben, da der Algorithmus so beschleunigt werden kann. Diese Parameter werden aus der Datenbank extrahiert, indem alle Marker-IDs abgefragt werden und die jeweilige Anzahl der Zeichen gezählt wird.

Falls eine Datei als Bildquelle ausgewählt wird, muss zusätzlich zu den bereits genannten Parametern der Dateipfad des Videos oder der Bildfolge als Argument übergeben werden. Bei einer Bildfolge ist zu beachten, dass die Bilder nach einem bestimmten Schema nummeriert sein müssen und die Nummerierung im Dateipfad angegeben werden muss (zum Beispiel *Bild%02d.jpg*, wobei nach dem % die Anzahl der Ziffern der Nummerierung angegeben wird).

Im Gegensatz zur Desktop Anwendung werden alle benötigten Daten in der Android-Applikation gleich mitgeliefert. Hierfür wird das sogenannte Asset verwendet, in welchem alle zusätzlich zur Anwendung benötigten Dateien gespeichert werden können. Da diese komprimiert in der Applikation inkludiert sind, werden sie für die weitere Benutzung in den temporären Ordner extrahiert. Somit können den Modulen die benötigten Dateipfade übergeben werden.

4.4.2. Bildgewinnung und -vorverarbeitung

Nachdem die Initialisierung aller Teile erfolgreich war, werden die Bilder je nach Bildquelle unterschiedlich generiert. Der einfachste Fall stellt eine vorhandene Video- oder Bilddatei dar. Hierfür wird die von OpenCV bereitgestellte VideoCapture-Klasse verwendet. Nachdem die Datei geöffnet wurde, wird mit dem überladenen Operator ">>" ein neues Frame geladen und zur weiteren Verwendung in einer Matrix gespeichert.

Falls die Kinect als Eingang ausgewählt wurde, müssen nach Initialisierung des Sensors zum einen die benötigten *FrameReader* geöffnet werden. In dieser Arbeit werden sowohl der Farb- als auf Tiefen-Framereader benötigt, welche kontinuierlich die benötigten Bilder liefern. Zum anderen müssen im Initialisierungsschritt die Buffer bereitgestellt werden. Durch sogenannte *FrameDescriptions* wird die entsprechende Höhe und Breite der beiden Bildarten ermittelt, welches der Auflösung der Bilder entspricht. Diese beträgt für das RGB-Bild 1920x1080 Pixel und für die Tiefenkarte 512x424 Pixel. Somit ergeben sich inklusive der Kanal- und Bitbreiten folgende Buffergrößen:

- RGB-Buffer := 1920x1080x4x8 Bit [Höhe x Breite x Kanäle x Bittiefe]
- Depth-Buffer := 512x424x1x16 Bit [Höhe x Breite x Kanäle x Bittiefe]

Schließlich wird der *CoordinateMapper* initialisiert. Mithilfe dieser Klasse erfolgt später, unter Verwendung der intrinsischen Daten, die Zuweisung der Tiefenpixel zu den zugehörigen RGB-Pixeln. Soll nun ein neues Bild erzeugt werden, wird die Funktion *aquireLatestFrame* für beide *FrameReader* aufgerufen. Die so gewonnenen Frames werden anschließend in die jeweiligen Buffer kopiert und wieder freigegeben. Die Freigabe ist hierbei sehr wichtig, da sonst keine neuen Frames mehr gewonnen werden würden. Schließlich werden für jeden Bildpunkt des RGB-Bildes sogenannte *DepthSpacePoints* generiert, in welchem die Tiefe für den jeweiligen Pixel gespeichert ist. Erzeugt werden diese mit Hilfe des *CoordinateMappers* durch folgenden Aufruf:

```
coordinateMapper->MapColorFrameToDepthSpace(depthWidth * depthHeight
    , depthBuffer, 1920 * 1080, depthSpacePoints);
```

Zum Schluss muss das Farbbild um die y-Achse gedreht werden. Dies beruht darauf, dass die Kinect vorrangig als Kamera für die Spielekonsole XBox-One entwickelt wurde. Damit die Bewegungen der Interagierenden auf dem Bildschirm nicht spiegelverkehrt sondern seitengetreu angezeigt werden, wird das Bild intern bereits gespiegelt. Daher muss das RGB-Bild wieder in die native Darstellung zurücktransformiert werden.

Als dritte Variante der Bildgewinnung wird abschließend das Erzeugen der Bilder mit Hilfe des Phablet 2 Pro betrachtet. Hierbei müssen alle benötigten Kameras aktiviert werden, welche im Falle der vorliegenden Arbeit die Farb- und Tiefenkamera sind. Dieses geschieht durch Ausführen der folgenden Funktionen:

```
TangoConfig_setBool(tango_config_, "config_enable_depth", true);
TangoConfig_setBool(tango_config_, "config_enable_color_camera",
    true);
```

Im nächsten Schritt werden die Buffer allokiert. Die Größe der Punktwolke wird durch die maximal bestimmbare Anzahl an Punkten festgelegt, welche durch eine Systemvariable definiert sind. Für das Farbbild wird ein Speicher von 1080 x 1920 x 3 Byte benötigt. Zum Schluss werden die Callbacks[7] initialisiert, welche aufgerufen werden, sobald die jeweilige Kamera ein Bild geliefert hat. Innerhalb dieser Funktionen werden die Sensorwerte in die Buffer überschrieben. Die Hauptroutine auf dem Android Gerät wird durch den GL[8]-Renderer gesteuert. Immer wenn ein neues Bild auf dem Gerät angezeigt werden kann, wird das im Buffer vorliegende Farbbild, sowie die Punktwolke in einer Variable gespeichert. Zusätzlich dazu werden zur späteren Pose-Ermittlung der beiden Kamerasysteme die jeweiligen Zeitstempel gesichert. Für die weitere Verarbeitung wird schließlich das vorliegende Farbbild in eine cv::Mat vom Typ YUV420sp konvertiert.

Da die Bilder aus unterschiedlichen Quellen stammen können, müssen sie für eine zentrale Weiterverarbeitung in das OpenCV Standard-Farbformat BGR[9] durch Nutzen der *cvtColor*-Funktion umgewandelt werden. Im Falle der Kinect wird zur Konvertierung nur der zusätzliche Alpha-Kanal entfernt. Für die Bilder aus der Android Kamera muss das Gesamtbild vom YUV420sp-Format in das BGR Format überführt wird. Ausgehend von den Kommentarzeilen 7475 bis 7477 im Quelltext der OpenCV Funktion [Ope16]

---

[7] dt. Rückruffunktionen

[8] Graphic Library, siehe [Goo17a]

[9] der Unterschied zum RGB-Format besteht durch Vertauschen des roten und blauen Kanals

geschieht dies mit folgenden Berechnungen:

$$\begin{aligned} B &= 1.164(Y-16)+2.018(U-128), \\ G &= 1.164(Y-16)-0.813(V-128)-0.391(U-128), \\ R &= 1.164(Y-16)+1.596(V-128). \end{aligned} \tag{4.1}$$

Die Variablen Y, U, V, sowie B, G, R stehen für die jeweiligen 8-Bit Kanalwerte. Nach der Berechnung werden die konvertierten BGR-Werte noch auf den Bereich von 0 bis 255 angepasst, indem die Zahlen, welche außerhalb des Bereichs liegen, auf den jeweiligen nächsten Wert gesetzt werden. Für negative Zahlen wäre dieses null und für alle zu großen Zahlen der Wert 255.

Nach der Konvertierung muss die Bildmatrix noch gedreht werden, da das Smartphone insgesamt vier mögliche Orientierungen bietet (jeweils zwei Hoch- und Querformate).

4.4.3. Aruco-Marker Detektor

Die Erkennung von Aruco-Markern und Raumnummernschilder erfolgt parallel, da diese sich nicht gegenseitig beeinflussen und beide ungefähr die gleiche Zeit benötigen (siehe 5.4). Zur Detektion der Marker wurde die OpenCV ArUco-Klasse verwendet, welche nicht in der Standard Bibliothek enthalten ist, jedoch über die zusätzliche contrib-Bibliothek[10] geladen werden kann. Die folgende Beschreibung des Detektionsalgorithmus folgt im wesentlichen der OpenCV-Dokumentation [Ope15c]. Für tiefergehende Informationen sei deshalb auf diese verwiesen.

Bevor mögliche ArUcos erkannt werden können, muss ein Dictionary definiert werden, damit die Art der zu detektierenden Marker fest definiert ist. Anschließend wird ein adaptiver Threshold auf das Bild angewandt. Dieser teilt das Bild in kleine Teilbereiche auf und sucht für jeden Teilbereich den besten Threshold. Somit existiert kein globaler Schwellwert, sondern ein für verschiedene Bereiche unterschiedlicher Threshold. In diesem so entstandenen Binärbild wird nun nach möglichen Marker Kandidaten gesucht. Dafür wird eine polygonale Näherung für jede gefundene Kontur angewandt. Hat diese Annäherung Ähnlichkeiten mit einem Quadrat, so gilt sie als möglicher Kandidat. Falls die so ermittelten Kandidaten eine sehr kleine Distanz zu einem anderen Kandidaten besitzen, werden diese zusammengefügt, da sie wahrscheinlich zu einem Marker gehören. Schließlich erfolgt die Bit-Extraktion der verbliebenen Konturen. Hierfür wird zuerst die perspektivische Verzerrung des Bildes korrigiert und anschließend ein Gitter, entsprechend der im Dictionary definierten Größe, darübergelegt. Ist der Inhalt der jeweiligen Kästchen mehrheitlich schwarz, so wird eine null extrahiert. Eine eins liegt im entgegengesetzten Fall vor. Nachdem alle Bitwerte evaluiert wurden, erfolgt die Marker Identifikation. Zuerst wird hierbei überprüft, ob der äußere Rand aus Nullen besteht. Dabei dürfen einzelne Bits auch nicht null sein, da ein schwarzes Bit durch Rauschen oder optische Verzerrungen fälschlicherweise als weißes Bit erkannt werden kann. Dieses ist abhängig von der Dictionary-Größe. Ist die Überprüfung positiv verlaufen, so werden die inneren Bitwerte extrahiert und mit möglichen Dictionary-Einträgen verglichen. Durch die vorhandene Fehlerkorrektur ist es sogar möglich, die richtige ID des Markers trotz einiger weniger Falschklassifizierung der Bitwerte zu ermitteln. Wurde ein Wörterbucheintrag

---

[10] siehe [Ope17]

gefunden, erfolgt zum Schluss noch eine verbesserte Eckenerkennung, da diese essentiell für die spätere optische Distanzermittlung sind. Hierfür werden die Gradienten zu den Nachbarpixeln betrachtet und entsprechend der Größe, wird die Ecke erweitert oder in ihrer Position belassen (siehe [Ope15b]).

Der aus dem Dictionary extrahierten ID wird im finalen Schritt noch ein XX vorn hinzugefügt. Der Grund hierfür liegt darin, dass besonders zwei- oder dreistellige IDs mit sich im Gebäude befindlichen Raumnummern übereinstimmen können. Um diese Doppeldeutigkeit zu verhindern, wird die Buchstabenkombination hinzugefügt, da keine Raumnummer eine derartige Zeichenfolge besitzen sollte. Beispielsweise würde aus der vom ArUco-Marker extrahierten ID *50* die Zeichenfolge *XX50* werden. Falls einmal eine ähnliche Raumnummer vorliegen sollte, muss dieser Zusatz geändert werden. Die ermittelten IDs werden schließlich an den DatenbankHandler übergeben, der eine mögliche Existenz in der Datenbank überprüfen kann. In nachfolgender Übersicht 4.4 werden alle Schritte vom RGB-Bild bis zur ID Extraktion des ArUco-Markers noch einmal anschaulich dargestellt.

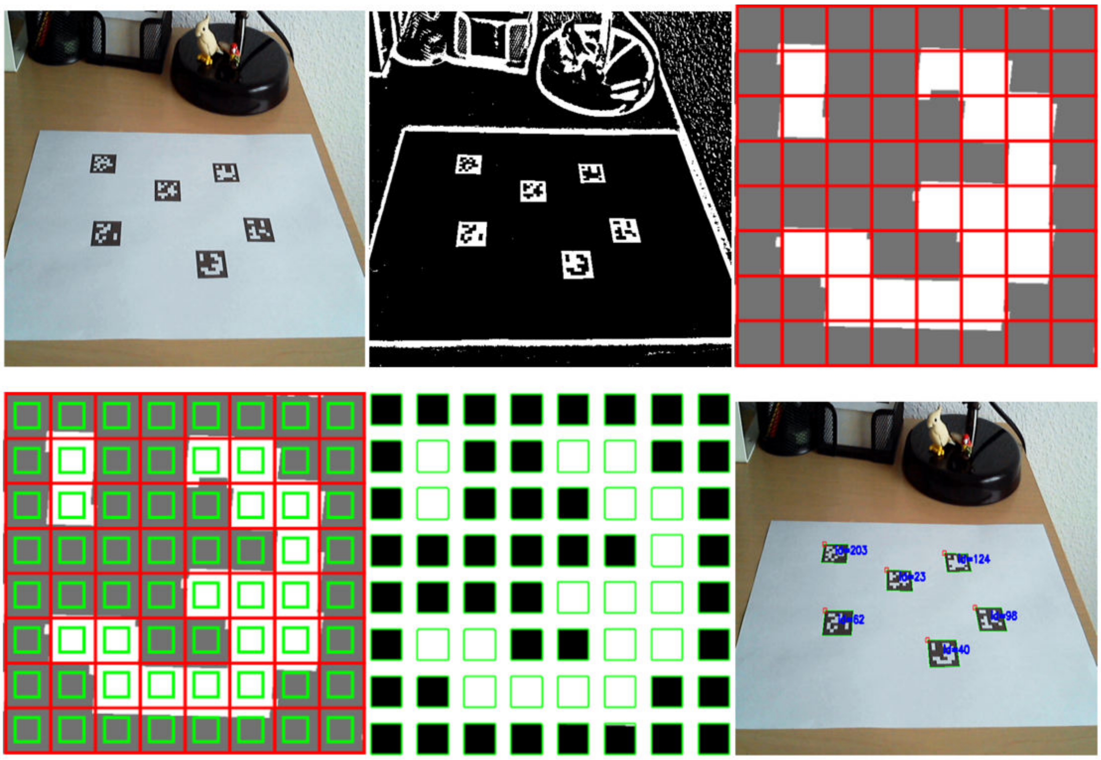


Abbildung 4.4.: Ablauf des OpenCV ArUco-Marker Detektionsalgorithmus (Quelle: [Ope15c])

4.4.4. Raumnummernschilderkennung

In diesem Modul werden aus einem Bild mögliche Raumschilder extrahiert. Im Listing 4.1 wird mittels Pseudo C++ Funktionen ein grober Überblick des Algorithmus gegeben, welcher folgend detailliert beschrieben wird.

Listing 4.1: Algorithmus zum Generieren von Trainingsdaten mit anschließendem SVM-Training

```
convertToGray(image);
canny(image);
FindOuterContours(image, contours);
for(i = 0, i < contours.size(), ++i){
  if(CheckSizeandRatio(contour[i]){
    ImproveContour(contour[i])
    InvestigateInnerContours(contour[i])
    DrawInnerContours(contour[i], contourImg);
  }
}
findInnerContours(contourImg, newContours);
for(j = 0, j < newContours.size(), ++j){
  if(nearToSimularContour(newContour[j], newContours){
    possibleIDs.push_back(findSimularContoursRight(newContour[j],
       newContours));
    possibleIDs.push_front(findSimularContoursLeft(newContour[j],
       newContours));
  }
}
if(possibleIDs.size() > minDigits && possibleIDs.size() < maxDigits)
    {
  GetSuroundingRoomSign(possibleIDs);
  RoomNumbers.push_back(predict(possibleIDs));
  GetRelatedChars(roomNumbers);
}
```

Nach der Umwandlung in ein Grauwertbild folgt die Konturenextraktion mittels des in Abschnitt 3.8.2 vorgestellten Canny-Algorithmus, zu sehen in Abbildung 4.5. Da das Bild nun in binärer Form vorliegt, kann es anschließend dilatiert werden, um mögliche Lücken in den Umrandungen zu schließen. Danach werden alle äußeren Konturen extrahiert. Für die weitere Betrachtung muss das sogenannte *boundingRectangle*, also das kleinste, die Kontur umschließende Rechteck, folgende Bedingungen erfüllen:

- Fläche > 100
- $0,3 <$ Verhältnis Höhe zu Breite < 5

Diese Werte sind typisch für eventuelle Raumnummern und wurden empirisch ermittelt. Wenn beide Bedingungen erfüllt wurden, wird die Kontur "verbessert". Zuerst wird der optimale Schwellwert der Kontur bestimmt, indem man auf den Ausschnitt des Grauwertbildes das Otsu-Thresholding (siehe 3.8.1) anwendet. Als nächstes wird überprüft,

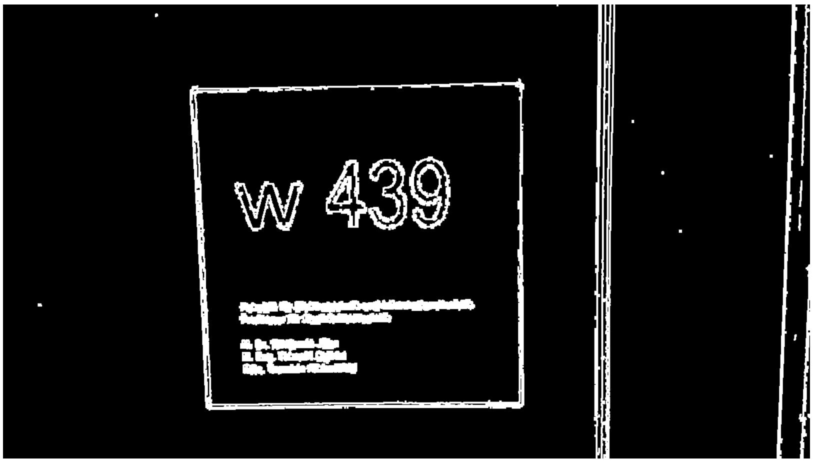


Abbildung 4.5.: Konturenextraktion durch Canny mit anschließender Dilatation

ob die inneren Konturen des Rechtecks dem binären Vordergrund zugeordnet sind, weil die Raumnummer innerhalb des Schildes schwarz beziehungsweise weiß aufgetragen sein kann. Dazu werden die vier Eckpixelwerte betrachtet, da diese immer dem realen Hintergrund des Schildes entsprechen. Sind mindestens drei von diesen nicht null und ist mindestens ein Viertel der Kontur dem Vordergrund zugeordnet, ist davon auszugehen, dass die Zeichen innerhalb des Schildes schwarz aufgedruckt wurden. Aus diesem Grund muss das Bild invertiert werden, damit die Ziffern und Buchstaben immer dem binären Vordergrund zugeordnet werden. In der folgenden Abbildung 4.6 wird dieser Sachverhalt zum besseren Verständnis visualisiert.

Abbildung 4.6.: Umwandlung Grauwertbild in binäres Bild durch Otsu-Thresholding. In der oberen Reihe erfolgt aufgrund weißer Eckpunkte anschließend eine Invertierung

Nachdem die Raumnummer dem Vordergrund zugewiesen wurde, wird erneut die *findContours* Funktion aufgerufen, dieses Mal jedoch nur auf den die Kontur umschließenden Teilbereich des gesamten Bildes. Die so gefundenen inneren Konturen werden erneut auf Minimalgröße und das richtige Höhen- zu Seitenverhältnis überprüft. Sind die Bedingungen erfüllt, so wird die jeweilige innere Kontur in ein neues binäres Bild kopiert. Hierbei wird zusätzlich darauf geachtet, dass keine feineren Konturen überschrieben werden. Das bedeutet, wenn beispielsweise bereits ein Türschild aus dem Bild extrahiert wurde, so wird bei Betrachtung der Türkontur der Bereich des Schildes ausgespart. Andersherum überschreibt die Schildkontur die jeweilige Stelle der Türkontur, da diese die kleinere und somit feinere Konturbetrachtung liefert. Möglich wird dies durch ein binäres Bild, welches zu Anfang nur aus Einsen besteht. Für jede gefundene Kontur wird der dementsprechende Bereich dem Hintergrund zugeordnet. Dieser Hintergrund wird bei jeder weiteren Betrachtung ausgespart, es sei denn, die gesamte Kontur liegt im Hintergrund des binären Bildes. Dann überschreibt diese Kontur die vorherige Kontur an der Stelle, da diese kleiner und feiner ist. Sind alle Konturen in das neue Bild kopiert, wird ein letztes Mal die *findContours* Funktion aufgerufen. Für alle gefunden Konturen werden neue *boundingRects* gebildet und abgespeichert. Bei diesen wird nun überprüft, ob sich ein anderes Rechteck mit geometrischen Maßen in der Nähe befindet. Dafür wird eine Suchumgebung definiert, welche viermal so breit und anderthalb mal so hoch ist wie das momentane Rechteck. Ein Beispiel hierfür ist in Abbildung 4.7 zu sehen.

Liegt eine andere Kontur mindestens zur Hälfte in dieser Suchumgebung, wird ein neuer Bildausschnitt generiert. Dieser ist 1,4 mal so hoch wie das betrachtete Rechteck ist und umfasst die gesamte Breite des Bildes. In dem in Abbildung 4.7 grün umrandeten Bildausschnitt wird für alle sich darin befindlichen Konturen überprüft, ob diese sich links oder rechts der Hauptkontur innerhalb einer bestimmten Pixeldistanz befinden. Diese Distanz wurde empirisch ermittelt und entspricht der dreifachen Höhe der Hauptkontur. Hierbei wird zuerst die rechte Seite überprüft. Wurde eine Kontur gefunden, wird deren Position im Konturenvektor in einem neuen Vektor gespeichert. Weiterhin erfolgt ein

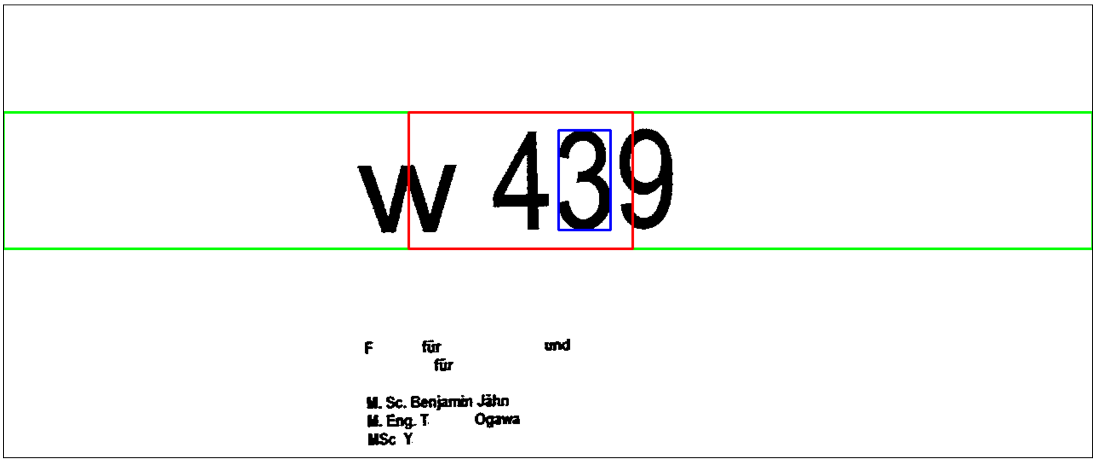


Abbildung 4.7.: rotes Rechteck: Suchradius um Startziffer 4; blaues Rechteck: vollständig inkludierte Ziffer 3, grünes Rechteck: Bildausschnitt für weiter Suche bei gefundener Nachbarkontur

neuer rekursiver Aufruf der Funktion, in welcher die gefundene Kontur die neue Hauptkontur wird. Die Rekursion wird so lange ausgeführt, bis keine Objekte mehr detektiert werden. Danach erfolgt die Überprüfung der linken Seite ausgehend von der Startkontur. Dies folgt dem gleichen Schema, nur dass die jeweiligen Positionen der Konturen dem neuen Vektor vorn hinzugefügt werden. Der so entstandene Vektor enthält alle Indizes der gefundenen Konturen von links nach rechts. Als nächstes wird für die so ermittelten zusammenhängenden Strukturen in dem dilatierten Ausgangsbild nach einem sie umgebenden Schild gesucht. Falls ein Schild gefunden wurde, werden jeweiligen Eckpunkte für die spätere Distanzermittlung gespeichert. Schließlich erfolgt das Schätzen der gefundenen Zeichen mittels einer bereits trainierten SVM. Dafür wird jede Kontur in einen Zeilenvektor transformiert und in die SVM gegeben. Als Ergebnis wird die wahrscheinlichste Klasse geliefert, die mittels des in 3.8.5 beschriebenen Multiklassen-Verfahrens bestimmt wurde. Das in einer Integer Zahl vorliegende Ergebnis wird anschließend anhand des Labels in die jeweilige Ziffer oder den jeweiligen Buchstaben umgewandelt. Da sich jedoch einige Buchstaben und Zahlen zum Verwechseln ähnlich sehen, werden dem Schätzvektor zusätzlich ähnlich aussehende Raumnummern hinzugefügt. Dabei werden folgende Zeichen um die jeweiligen ähnlichen Ziffern und Buchstaben ergänzt:

- 0 → I,L
- 2 → Z
- 5 → S
- 8 → G
- 9 → G
- G → 8,9
- I → 1,L
- L → 1,I
- O → 0
- S → 5
- Z → 2

Damit die Umwandlung anhand der obigen Auflistung erfolgen kann, müssen alle Buchstaben in Großbuchstaben konvertiert werden. Dies ist auch für die spätere Datenbankabfrage notwendig, da alle Zeichen einer ID auch hier in Großbuchstaben vorliegen. Damit wird verhindert, dass es zu unnötigen Abfragen kommt, da zahlreiche Buchstaben wie beispielsweise S,U,V... bei der Erkennung nicht zwischen Groß- und Kleinschreibung unterschieden werden können. Nach erfolgter Konvertierung wird, falls ein ähnlicher Buchstabe vorliegt, dieser ersetzt und der Schätzvektor um die neue Zeichenfolge ergänzt. Dies geschieht für alle möglichen Kombinationen. Ein Beispiel für die Zeichenfolge *A1B2* ist in Abbildung 4.8 zu sehen. Der so entstandene Vektor[11] von Vektoren[12] wird schließlich den DatenbankHandler zur Überprüfung auf ein eingetragenes Raumnummernschild übergeben.

4.4.5. Distanzermittlung mittels Bildverarbeitung

Wurden die Ecken der gefundenen Marker extrahiert, kann die Entfernungs - und Orientierungsbestimmung erfolgen. Dafür wird die reale Höhe und Breite aus der Datenbank geladen (siehe nächsten Absatz). Mittels der im theoretischen Teil der Arbeit vorgestellten Formel 3.36 lässt sich die benötigte externe Kameramatrix bestimmen. Hierfür werden

---

[11] enthält Ausgänge der SVM

[12] enthält ähnlich aussehende Raumnummern

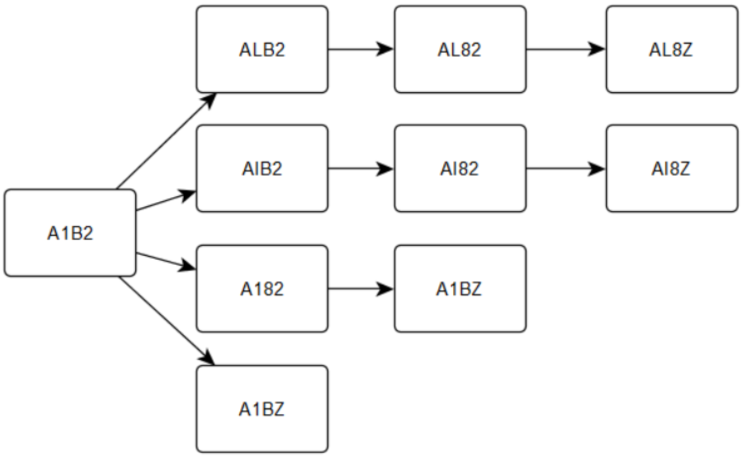


Abbildung 4.8.: Ermittlung ähnlich aussehender Zeichen wie A1B2

insgesamt acht Gleichungen gebildet (4 Eckpunkte à zwei linear unabhängigen Gleichungen). Da die externe Matrix nur sechs zu bestimmende Freiheitsgrade besitzt (jeweils drei Rotationen und Tanslationen), dienen die zwei zusätzlichen Gleichungen zur Beseitigung von eventuellen mehrdeutigen Lösungen die in nichtlinearen Ausgleichsproblemen auftreten können. Der Aufruf der von OpenCV bereitgestellten solvePnP-Funktion sieht schließlich wie folgt aus:

```
solvePnp(realCornerPoints, imageCornerPoints, intrinsicMat,
    distortionVec, rVec, tVec, useExtrinsicGuess, CV_ITERATIVE)
```

wobei die Bedeutung der Variablen folgend gelistet ist:

- **realCornerPoints** - Eingangsvektor aus vier 3D-Eckpunkten des realen Objektes
- **imageCornerPoints** - Eingangsvektor aus vier 2D-Bildeckpunkten
- **intrinsicMat** - intrinsische Kameramatrix (Eingang)
- **distortionVec** - Eingangsvektor mit radialen Verzerrungskoeffizienten
- **rVec** - Ausgangsvektor mit rotatorischer Bewegung von Kamera zu Objekt
- **tVec** - Ausgangsvektor mit translatorischer Entfernung von Kamera zu Objekt, Einheit wie realCornerPoints
- **useExtrinsicGuess** - boolesche Eingangsvariable zur Benutzung eines Startwertes bei der Berechnung, auf *false* gesetzt
- **CV_ITERATIVE** - Flagge zur Bestimmung der Methode zur Lösung des nichtlinearen Problems, hier LM-Algorithmus (siehe 3.4)

Die realen Objektkoordinaten werden hierbei vom Mittelpunkt aus gemessen, welcher gleichzeitig der Referenzpunkt der Distanzbestimmung ist. Das heißt, ein Objekt der Größe von 10x10 Zentimeter besäße die 3D-Punkte {(-5,-5, 0), (5, -5, 0), (5, 5, 0), (-5,

5, 0)}. Entsprechend der Definition von OpenCV, liegt der Ursprung des Koordinatensystems hierbei in der oberen linken Ecke. Die z-Koordinaten sind immer null, weil die Distanz vom Objekt zur Kamera bestimmt werden soll.
Nach dem Aufruf der solvePnP Funktion muss anschließend der Rotationsvektor in eine 3x3 Matrix überführt werden, damit die Rotationswinkel um die Achsen extrahiert werden können. Dies geschieht Anwendung der in beschriebenen 3.8.7 Rodrigues-Formel. Des Weiteren muss eine zusätzliche Betrachtung des Translationsvektors vorgenommen werden. Dazu soll der Zusammenhang von Kamera- zu Weltkoordinaten ausgehend von Formel 3.36 zu

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} R_{11} & R_{21} & R_{31} \\ R_{12} & R_{22} & R_{32} \\ R_{13} & R_{23} & R_{33} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{4.2}$$

umformuliert werden. Es ist zu erkennen, dass die translatorische Verschiebung in Kamerakoordinaten vorliegt, weil die Rotation vor der Translation durchgeführt wird. Wie in Abbildung 4.9 jedoch ersichtlich ist, sind die Koordinatenachsen der Systeme nicht identisch.

Da die Entfernung der Kamera zum Objekt gesucht ist, muss eine Beschreibung des Ursprungs des Kamerakoordinatensystems in Weltkoordinaten gefunden werden. Laut [Sim13] soll deswegen folgender Zusammenhang gelten:

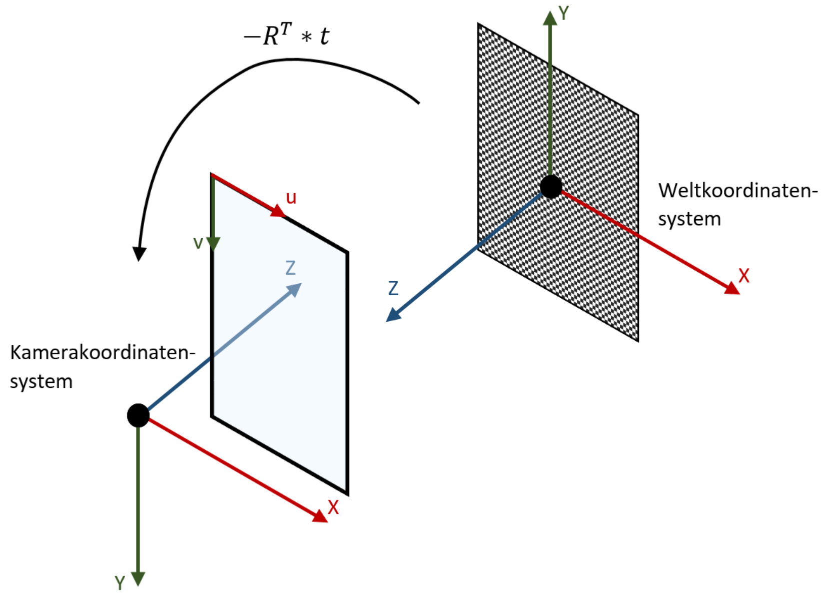


Abbildung 4.9.: Welt- und Kamerakoordinatensystem

$$\begin{bmatrix} R_w & R_{21} & R_{31} & T_1 \\ R_{12} & R_{22} & R_{32} & T_2 \\ R_{13} & R_{23} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_C & R_C & R_C & C \\ R_C & R_C & R_C & C \\ R_C & R_C & R_C & C \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} . \tag{4.3}$$

Hierbei stellt C den Ursprung des Kamerakoordinatensystems in Weltkoordinaten dar. R_C und R_W sind die Rotationen um die jeweiligen Koordinatensysteme. Da die Rotationsmatrix eine orthogonale Matrix ist, ist die Inverse gleich der Transformierten [And10] lässt sich die vorher genannte Gleichung schließlich zu

$$C = -R^t * \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{4.4}$$

umformulieren. Somit liegt die Distanz C von Objekt zur Kamera in metrischen Weltkoordinaten vor, welches ihren Ursprung in der Mitte des Markers hat.

4.4.6. Geodatenbank

Wie bereits im Abschnitt Initialisierung 4.4.1 erläutert wurde, wird eine SQLite Datenbank verwendet, welche durch SpatiaLite um geografische Objekte und Funktionen erweitert wurde. Die Abfrage der Einträge erfolgt über SQL. Die Befehle werden hierbei immer nach dem folgenden Schema gebildet:

```
SELECT Spalte(n) FROM Tabelle WHERE Bedingung=true
```

Die DatabaseHandler-Klasse stellt zwei Funktionen zur Verfügung. Als Erstes erfolgt die Abfrage der Existenz eines Markers, welcher eine spezifische ID besitzt. Falls ein Marker mit der gewünschten ID in der Datenbank vorhanden ist, werden die realen geometrischen Maße zurückgegeben, um eine optische Distanzermittlung zu ermöglichen. Die Abfrage hierfür sieht folgendermaßen aus:

```
SELECT indoor_marker_size_x, indoor_marker_size_y FROM points WHERE indoor_marker_id
    = ID
```

Wichtig hierbei ist, das sämtliche Buchstaben der ID in Großbuchstaben vorliegen müssen. Die zweite Funktion des DatabaseHandlers liefert die geografische Beschreibung des jeweiligen Markers. Mit Hilfe der ID wird die Position in Längen- und Breitengrad, sowie die Orientierung des Markers extrahiert. Die Orientierung wird hierbei durch einen Wert von 0 bis ausschließlich 360 angegeben, wobei 0 für den geografischen Norden und 180 dementsprechend für den geografischen Süden steht. Die beschriebenen Eigenschaften werden mit Hilfe von

```
SELECT ST_X(GEOMETRY), ST_Y(GEOMETRY), indoor_marker_orientation FROM
    fdo_points WHERE indoor_marker_id = ID
```

aus der Datenbank gewonnen. Die Longitude und Latitude werden mit Hilfe der Spatialite-Befehle *ST_X(GEOMETRY)* und *ST_Y(GEOMETRY)* aus der Datenbank extrahiert.

4.4.7. Kameratiefendaten

Wurde ein Marker erkannt und die Distanz zum Objekt durch die Funktion *solvePnP* ermittelt, kann diese unter Einbeziehung der Distanzdaten verbessert werden. Es werden sowohl die Tiefendaten des Mittelpunktes, als auch der Eckpunkte des Markers ermittelt. Dabei unterscheiden sich die Vorgehensweisen zwischen Desktop und Smartphone Anwendung. Bei der Kinect können die Distanzen durch Abfrage des *DepthSpacePoints* an dem jeweiligen Bildpunkt(x,y) ermittelt werden. Besitzt dieser an der Stelle einen

endlichen Wert, so wird er als Tiefenwert für den gegebenen Bildpunkt gesetzt.
Im Gegensatz dazu wird beim Android-Gerät zuerst die Pose zwischen Tiefen- und Farbkoordinatensystem berechnet. Dieses ist in so fern erforderlich, da die jeweiligen Daten zu unterschiedlichen Zeiten aufgenommen sein können und durch die zwischenzeitliche Bewegung des Smartphones einen unterschiedlichen Ursprung besitzen. Deswegen wird anhand der spezifischen Zeitstempel die externe Transformationsmatrix zwischen beiden Systemen bestimmt. Für jeden Wert in der Tiefenpunktwolke wird nun der zugehörige Bildpunkt im Farbbild ermittelt. Dieses geschieht durch Multiplikation des jeweiligen Punktwolkenwertes mit der externen und anschließend internen Transformationsmatrix. Nun werden die so generierten x- und y- Pixelwerte mit den des gegebenen Farbpixels verglichen. Liegen beide Werte innerhalb eines 5-Pixel Durchmessers, so kann davon ausgegangen werden, dass die Werte der Punktwolke dem des Farbwertes entsprechen. Als Tiefenwerte werden schließlich die in das Farbkoordinatensystem transformierten x-,y-,z-Werte der Punktewolke geliefert. Die beschriebene Vorgehensweise wird noch einmal zur besseren Verständlichkeit durch folgenden Pseudocode verdeutlicht.

Listing 4.2: Algorithmus zur Extraktion der Tiefenwerte in Farbkoordinatensystem für gegebene Pixelkoordinaten x,y

```
Require: desired color coordinates x,y
pose := PoseFromDepthToColorCamera
for each depth_point in point_cloud do
  color_point := depth_point*pose
  color_pixel := color_point * intrinsic
  if abs(color_pixel.x - x < 5) and abs(color_pixel.y - y < 5) then
    return color_point  * MeterToMillimeter
  end if
end for
```

Die Tiefenwerte der Mitte und Ecken des Markers fließen unterschiedlich in die Korrektur der optischen Distanz ein. Konnten allen vier Ecken des Markers durch Time of Flight Messung eine zugehörige Tiefe zugeordnet werden, so erfolgt die Bestimmung der Distanz ähnlich dem Prinzip der Satellitenortung. Mit Hilfe des Gleichungssystems[13]

$$\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2} = d_i, 0 \leq i \leq 4 \tag{4.5}$$

kann der Translationsvektor $(x, y, z)^T$ von Kamera zu Marker bestimmt werden. Mit Hilfe der Variablen x_i, y_i, z_i werden die geometrischen Positionen der Eckpunkte innerhalb des Markers beschrieben, wobei z den Wert 0 besitzt. Die Distanz zu den jeweiligen Ecken wird mittels d angegeben. Da hier ein nichtlineares Ausgleichsproblem vorliegt, kann wie bei der optischen Distanzermittlung der LM-Algorithmus eingesetzt werden. Zur schnelleren Bestimmung der Werte werden die bereits ermittelten Distanzen als Startwerte vorgegeben.
Zur Evaluierung der Distanz durch den Mittelpunkt des Markers wird erneut zwischen Kinect und Lenovo Phablet unterschieden. Bei der wird Kinect nur der jeweilige Distanzwert zurück geliefert. Dieser wird folgend als Korrekturturwert der optisch ermittelten

---

[13] siehe [All03]

Distanz benutzt. Dazu bildet das Verhältnis der durch Time of Flight gemessenen Tiefe und der euklidischen Distanz der optisch bestimmten x-,y- und z-Werte. Dieses Verhältnis fließt schließlich als Korrekturwert für jeden Wert des Translationsvektors ein, zu sehen in folgenden Gleichungen:

$$\begin{gathered} d_{\text{cor}} = \frac{\sqrt{x^2 + y^2 + z^2}}{d} \\ \begin{bmatrix} x_{\text{new}} \\ y_{\text{new}} \\ z_{\text{new}} \end{bmatrix} = d_{\text{cor}} * \begin{bmatrix} x_{\text{old}} \\ y_{\text{old}} \\ z_{\text{old}} \end{bmatrix}. \end{gathered} \tag{4.6}$$

Im Unterschied dazu liefert das Phab 2 Pro bereits einen Distanzwert mit x-, y- und z-Komponente. Dieser ersetzt schließlich den optisch ermittelten Wert, da er eine bessere Genauigkeit besitzt (siehe 5.1).

4.4.8. Bestimmung der Position

Nach jeder erfolgreichen Distanzermittlung wird ein Markerobjekt mit den Eigenschaften:

- ID
- Orientierung
- xyz-Distanz

zusammen mit einer Zählvariable in einem Vektor abgespeichert. Die Zählvariable ist abhängig davon, wie viele Markerobjekte der letzten Frames zur Positionsermittlung berücksichtigt werden sollen. Im Fall dieser Arbeit beträgt der Wert dieser Variable 20. Vor jeder Bearbeitung des Bildes wird die Zählvariable um Eins inkrementiert. Falls sie ihre maximale Größe überschreiten würde, wird sie auf null gesetzt. Anschließend werden alle Markerobjekte im Vektor gelöscht, die dessen Zählvariable mit dem inkrementierten Wert übereinstimmt. Somit basiert die Positionsermittlung nur auf den neuesten Werten und wird nicht durch ältere verfälscht, da sich die Position des Aufnahmegerätes mittlerweile verändert haben könnte.

Wurden alle gefundenen Markerobjekte in den Vektor eingefügt, wird anschließend überprüft, ob eine Mindestanzahl an gleichen IDs vorliegt. Dieser Wert ist im verwendeten Algorithmus auf 10 gesetzt. Eine Mindestanzahl wird benötigt, um etwaige Schwankungen und Fehlmessungen ausgleichen zu können. Wurde eine Mindestanzahl von gleichen ID-Einträgen gefunden, werden alle Distanzen in x- und z-Richtung, sowie Orientierungen zum Marker in verschiedene Vektoren gespeichert und der Größe nach sortiert. Für die mittleren drei Werte wird schließlich der Mittelwert gebildet. Als nächstes wird die geografische Position des gefundenen Markers aus der Datenbank extrahiert. Zu diesem, in Längen- und Breitengraden vorliegenden Wert, werden die Distanzwerte unter Berücksichtigung der Orientierung ebenfalls in die geografische Größe umgerechnet und zu dem Datenbankwert hinzu addiert. Schließlich wird die Variable *m_position* mit den ermittelten Längen- und Breitengradwerten beschrieben. Folgend wird der Pseudocode zu der bereits beschriebenen Vorgehensweise aufgezeigt:

Listing 4.3: Algorithmus zur Berechnung der Position aus gefundenen Markern der letzten Frames

```
counter := (counter+1) % MAX_FRAMES_COUNTER
for each markerObject in detectedMarkers do
  if markerObject.counter = counter then
    delete markerObject
  end if
end for
Detect roomnumbers and ArUcos
Update detectedMarkers with found_IDs
for each found_ID do
  if detectedMarkers[ID].count > MIN_AMOUNT_MARKER_OCCURENCES then
    sort xPosVec, zPosVec and Orientations of detectedMarkers[ID]
    xDist := (xPosVec[midSize-1] + xPosVec[midSize] + xPosVec[
       midSize+1]) / 3
    zDist := (zPosVec[midSize-1] + zPosVec[midSize] + zPosVec[
       midSize+1]) / 3
    orientation = (oriVec[midSize-1] + oriVec[midSize] + oriVec[
       midSize+1]) / 3
    orientation += DB orientation entry of found_ID
    if (orientation < 0 then
      orientation += 360
    else if (orientation > 360) then
        orientation -= 360
    end if
    Extract Longitude and Latitude of ID from DB
    Convert xDist and zDist to Lon and Lat
    return position of added values
  end if
end for
```

Damit war die Positionierung erfolgreich und das Programm kann beendet werden.

4.5. SVM-Training

Zusätzlich zum Positionsalgorithmus wird in der Desktop Umgebung die Möglichkeit geboten, Trainingsdaten für die SVM zu erstellen und deren Stützvektoren durch Training zu bestimmen. Hierfür müssen die Argumente “-t” , sowie der Pfad zu den Trainingsrohdaten angegeben werden. Die Rohdaten sollten in einem Bildformat vorliegen, welches außer der Buchstaben und Ziffern keine weiteren Objekte enthält. Dieses wäre beispielsweise, wie es in Abbildung 4.10 zu sehen ist, eine weiße Seite mit allen Arten von Buchstaben und Ziffern.

Um den Datensatz zu generieren wird das Rohdatenbild zuerst in ein Grauwert- und dann schließlich in ein Binärbild umgewandelt. Nach diesen Umformungen werden alle Konturen (Buchstaben und Ziffern) detektiert und in einem Container gespeichert, über welchen schließlich iteriert wird. Bei jedem Iterationsschritt wird das aktuelle Zeichen auf die Größe 15x30 geändert, in ein Binärbild mit weißem Vordergrund und schwarzen

Abbildung 4.10.: Trainingsdatenerzeugung für SVM. Grüne Rechtecke stehen für bereits gelabelte Zeichen, blaue Rechtecke für das aktuelle Zeichen, welches im linken Fenster vergrößert dargestellt wird

Hintergrund umgewandelt und schließlich mit einem blauen Kästchen umrandet und in einem weiteren Fenster vergrößert dargestellt. Der Benutzer muss nun die Zahl oder den Buchstaben eingeben, wobei auch auf Groß- und Kleinschreibung geachtet wird. Entsprechend der Eingabe wird das Label und die Kontur intern in einem Vektor gespeichert. Ebenfalls wird für die spätere Verwendungsmöglichkeit das Bild in einem Ordner gespeichert. Dabei besitzt jede Ziffer oder Buchstabe einen eigenen Ordner. Die Überprüfung der Existenz der jeweiligen Ordner, sowie deren eventuelle Erstellung geschehen unter Zuhilfenahme der Boost-Bibliothek. In diesen Ordnern liegen schließlich alle entsprechenden Konturen durchnummeriert im PNG-Format[14] vor. Falls die aktuelle Kontur mangelhaft ist, kann sie durch betätigen einer anderen Taste (wie zum Beispiel der Leertaste) übersprungen werden. Die gerade beschriebene Vorgehensweise ist im folgenden Listing 4.4 auch noch einmal als Pseudocode dargestellt.

---

[14] Portable Network Graphics

Listing 4.4: Algorithmus zum Generieren von Trainingsdaten mit anschließendem SVM-Training

```
convert color_image to gray_image
threshold gray_image
find contours in threshold_image
for each contour_entry  in found_contours do
  rectangle contour_entry
  display contour_entry and wait for user input
  if input = letter or input = number then
    resize contour_entry to 15{,}30 Pixels
    save input in labelVec
    save contour_entry in trainingVec
  else
    continue
  end if
end for
trainingLabelsMat := labelVec
trainingImagesMat := trainingImageVec;
train SVM with trainingImagesMat and trainingLabelsMat
```

Wurden alle Konturen gelabelt, kann die SVM trainiert werden. Die Konturbilder dienen mit den entsprechenden Labels als Eingang. Auch die Parameter sind fest vorgegeben, welche im Anhang C zu finden sind. Nach abgeschlossenem Training wird eine XML-Datei unter dem angegebenen Pfad angelegt.

Falls bereits ein Trainingssatz vorhanden ist, kann die Erstellung des Selbigen übersprungen werden. Dies wird durch das Setzen der Bool-Variable *createTrainingData* auf *false* erreicht. Somit wird nur die SVM mit den im Pfad vorliegenden Daten trainiert.

5. Test und Auswertung

Die Auswertung des Algorithmus zur Positionsbestimmung erfolgt in zwei Teilen. Zum Einen soll überprüft werden, wie genau sowohl die optische, als auch die physische Distanzermittlung ist. Hierfür wurde ein ArUco-Marker als Referenzpunkt gewählt, da dieser sich, anders als die Raumnummerschilder, beliebig aufhängen lässt. Daher konnte eine ausreichend große Testumgebung gewählt werden, welche in Abbildung 5.1 zu sehen ist.
Zum Anderen soll die Trefferrate bestimmt werden, mit der die Ziffern und die Kontur des Schildes der Raumkennzeichnung detektiert werden. Dazu wurden verschiedene Raumschilder aus unterschiedlichen Gebäudeteilen der technischen Universität Chemnitz fotografiert. Hierbei wurde sowohl die Distanz, als auch die Orientierung zum Türschild variiert.
Schließlich soll noch eine Geschwindigkeitsbetrachtung vollzogen und eine Aussage darüber getroffen werden, in wie fern die Echtzeitfähigkeit auf den verschiedenen Plattformen gegeben ist.

Abbildung 5.1.: Testumgebung zur Messwertaufnahme

5.1. Distanzermittlung

Um eine Aussage über die Genauigkeit des Algorithmus in Bezug auf die Distanz und Orientierung zu einem Marker aufstellen zu können, wurden 15 verschiedene Messpunkte für die Microsoft Kinect und neun unterschiedliche Messpunkte für das Lenovo Phablet aufgenommen. Dafür wurden beide Geräte auf einem Stativ fixiert, um eine stabile

Voraussetzungen während eines Messvorgangs schaffen zu können. Sonst wäre eine qualitative Auswertung aufgrund zu stark fluktuierender und somit ungenauer Werte nicht möglich. Um verschiedene Szenarien betrachten zu können, variieren die Positionen in x- und z-Richtung, sowie in der Orientierung in Bezug auf den Aruco-Marker. Eine Betrachtung der Distanzwerte in y-Richtung findet nicht statt, da zur Positionierung die Höhe des Gerätes keine Rolle spielt. Der Referenzwert für jeden Punkt wurde mittels eines optischen Laserabstandsmesser ermittelt. Mit diesem Gerät konnte zwar die Distanz in z-Richtung sehr genau bestimmt werden, jedoch können die Referenzwerte in x-Richtung fehlerbehaftet sein, da hierbei keine direkte Messung mithilfe des Abstandsmessers möglich war. Eine weitere Fehlerquelle ist die nicht genaue achsparallele Ausrichtung von Kamera zu Marker.

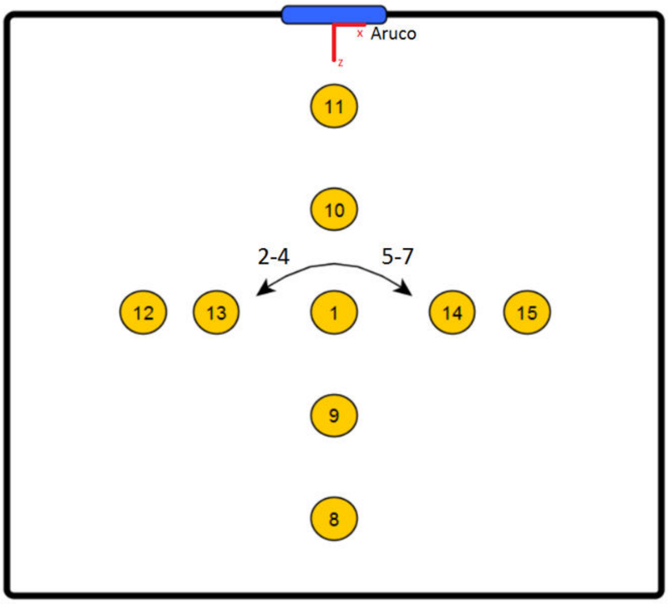
Für jeden Punkt wurden die Werte der optischen Distanzermittlung, sowie die gemessenen Tiefenwerte für den Mittelpunkt und die Ecken des Markers gespeichert. Um Messungenauigkeiten kompensieren zu können wurden für jede Messung elf Frames verarbeitet und die Endposition als der gemittelte Median bestimmt. Aufgrund der technischen Grenzen der Time of Flight Messung wurden hierbei vereinzelt weniger als die vorgegebenen elf Messungen gespeichert. Beispielsweise konnte bei einer Distanz von fünf Metern zum Aruco-Marker kein Wert für die Distanzmessung beim Phablet ermittelt werden.

Nachfolgend werden die Ergebnisse der Messungen entsprechend der verwendeten Hardware aufgezeigt und ausgewertet. Die folgende Tabelle 5.1 enthält die wichtigsten statistischen Eigenschaften der aufgenommenen Messwerttabellen, welche im Anhang A zu finden sind. So werden neben der Summe der absoluten Fehler ε_{abs} auch der relative Fehler ε_{rel}, kleinster und größter absoluter Fehler ε_{min} und ε_{max}, sowie die Varianz σ^2 dargestellt.

Tabelle 5.1.: Vergleich der charakteristischer Eigenschaften der Messergebnisse

	$\sum \varepsilon_{abs}$	ε_{rel}	$\sigma^2(\varepsilon)$	$\lvert\varepsilon_{min}\rvert$	$\lvert\varepsilon_{max}\rvert$
Kinect:					
Diff D_{x_opt}[mm]	8035,01	535,67	626596,55	31,08	1799,97
Diff D_{x_mid}[mm]	7368,48	491,23	534827,51	16,82	1718,74
Diff D_{x_corner}[mm]	10512,13	750,87	855827,33	1,16	1526,58
Diff D_{z_opt}[mm]	1668,68	111,25	17776,77	3,97	361,88
Diff D_{z_mid}[mm]	3267,65	217,84	43017,98	2,75	657,92
Diff D_{z_corner}[mm]	2891,26	206,52	97339,40	4,51	826,04
Diff Ori [°]	156,71	10,45	109,15	0,27	28,86
Phab 2 Pro:					
Diff D_{x_opt}[mm]	1799,30	199,92	53959,13	47,68	383,71
Diff D_{x_mid}[mm]	1823,83	227,98	154173,47	6,94	782,86
Diff D_x_corner[mm]	1163,37	145,42	41267,31	33,14	548,50
Diff D_{z_opt}[mm]	735,48	81,72	2171,18	8,97	152,93
Diff D_{z_mid}[mm]	253,715	31,71	2237,30	2,68	138,02
Diff D_{z_corner}[mm]	191,859	23,98	618,88	1,95	49,5
Diff Ori [°]	31,78	3,53	45,49	0,02	7,35

Microsoft Kinect v2



Mess-reihe	Orientierung / Distanz (mit Richtung) zum Marker
1	3m (z)
2	-35°
3	-30°
4	-15°
5	15°
6	30°
7	35°
8	5m (z)
9	4m (z)
10	2m (z)
11	1m (z)
12	-1,5m (x)
13	-1m (x)
14	1m (x)
15	1,5m (x)

Abbildung 5.2.: Position der Microsoft Kinect v2 für die jeweiligen Messreihen

Zur Auswertung der Genauigkeit des Positionierungsprogrammes wurden mithilfe der Kinect 15 Messreihen á elf Messwerte aufgenommen. Dabei wurde in Messung 1 die Referenzposition bestimmt, um welche alle anderen Messungen mit jeweils einer Veränderlichen variieren. Bei den Aufnahmen 2 bis 7 wurde die Orientierung zum Marker geändert, wobei um einen Bereich von -35° bis +35° zum Objekt gemessen wurde. Als nächstes wurde der Abstand in z-Richtung in ein Meter-Schritten geändert. Hierbei waren fünf Meter das Maximum und ein Meter Entfernung zum Objekt das Minimum. Die Messungen bei veränderlicher Distanz in x-Richtung fanden schließlich in den Aufnahmen 12 bis 15 statt. Dabei wurden vier Messungen im Bereich von -1,5 Meter bis +1,5 erstellt. Die Positionen sind in Grafik 5.2 abgebildet.
Um die Endposition zu ermitteln, wurden jeweils elf Werte mittels optischer Positionsbestimmung und Time of Flight Messung von Mittelpunkt und Ecken des Markers pro Position aufgenommen. In Grafik 5.3 sind die Abweichungen der elf Messwerte zum Referenzpunkt für die erste Messung (3m Abstand in z-Richtung) zu sehen.
Es ist deutlich zu erkennen, dass die jeweiligen Messpunkte eine relativ geringe Varianz innerhalb der jeweiligen Methoden zur Distanzermittlung besitzen. Lediglich ein Ausreißer liegt bei der optischen Entfernungsermittlung und Distanzmessung durch Mittelpunkt vor. Da dieser Punkt jedoch der erste aufgenommene Wert der Messreihe ist, kann ein noch vorliegendes starkes Bildrauschen eine Erklärung hierfür sein. Weiterhin ist ein Schwanken der Werte der Distanzmessung mittels Eckpunkte in x-Richtung zu erkennen. Da dieses Rauschen auch in den weiteren Messreihen vorhanden ist, kann davon ausgegangen werden, dass diese Art der Positionierung zumindest hinsichtlich der x-Richtung ungeeignet ist. Zwar können die fluktuierenden Werte durch Median- und Mittelwertbildung optimiert werden, jedoch ist anhand Abbildung 5.4 erkennbar, dass

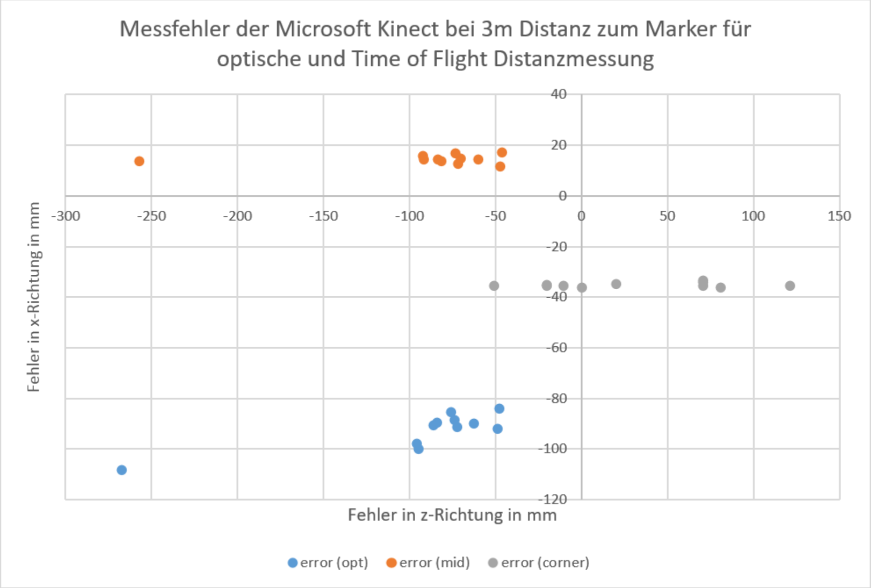


Abbildung 5.3.: Abweichung der Messwerte zum Referenzpunkt (0,0) für 3m Distanz zum Marker

die Positionsbestimmung mithilfe der Eckpunkte des Markers die größten Abweichungen liefert. Besonders bei Veränderung der Orientierung, sowie der horizontalen Distanz zum ArUco-Marker, ist der Fehler zum Referenzwert mit bis zu 1,5 Metern sehr groß. Weiterführende Tests bestätigten die Vermutung, dass der Tiefensensor der Kinect unzureichende Werte liefert, sobald sich das Objekt weiter als circa 15° vom Bildmittelpunkt entfernt befindet. Die Tiefenmessung verharrt bei circa drei Metern, trotz Variation der Distanz in x-Richtung. Aus diesem Grund ist auch die Positionierung mittels Distanzmessung der Mitte nicht zu empfehlen. Zwar bietet diese Methode den kleinsten Fehler von allen, doch resultiert dies nur aufgrund der schlechten Werte der optischen Bestimmung. Da ab einem Winkel von 30° der Positionierungsfehler der optischen Methode sehr groß wird, wird dieser Fehler durch die zu kleine Tiefendistanzmessung nach unten korrigiert. Somit muss zur Bestimmung der x-Position die optische Distanzermittlung verwendet werden. Die große durchschnittliche Abweichung von circa einem halben Meter kommt vor allem durch die Fehlerkennung in Messreihe 14 und Messungen 2-3, sowie 6-7 zustande. Jedoch sollten diese in der Realität zu vernachlässigen sein, da sich der Nutzer zur besseren Positionierung auf das Objekt zubewegen sollte und somit keine großen Orientierungsunterschiede beziehungsweise Differenzen in horizontaler Position mehr vorliegen. Die Abweichungen von maximal 16 Zentimeter bei frontaler Betrachtung des Markers unter 4 Meter ist für eine exakte Positionierung ausreichend. Nichts desto trotz enttäuschen die ermittelten Werte, da besonders die Tiefenbestimmung durch die Time of Flight Methode nur aufgrund von Fehlmessungen zu einer Verbesserung der optisch bestimmten Werte führten. Die Auswertung der Distanzfehler in z-Richtung fällt wesentlich positiver aus, da der absolute und somit auch durchschnittliche Fehler bei allen verwendeten Methoden um mindestens die Hälfte reduziert werden konnte. Für die optische Distanzbestimmung konnte sogar ein Verbesserung um den Faktor Fünf er-

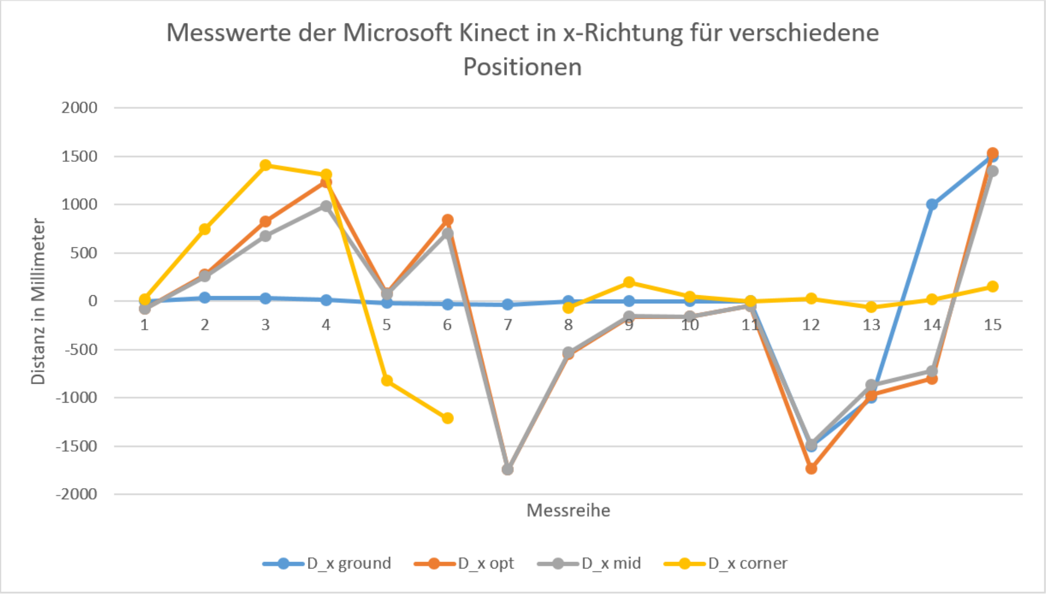


Abbildung 5.4.: Distanzwerte in x-Richtung bezüglich des ArUco-Markers für alle Messreihen der Kinect

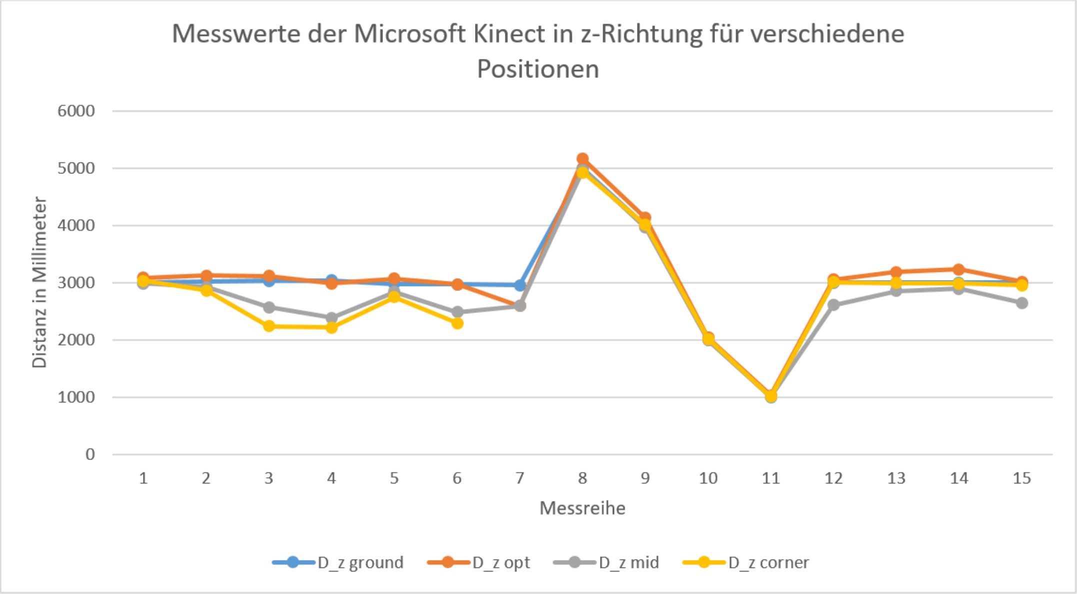


Abbildung 5.5.: Distanzwerte in z-Richtung bezüglich des ArUco-Markers für alle Messreihen der Kinect

zielt werden. Dies beruht vor allem darauf, dass die Distanz in z-Richtung nicht durch Auswertung der perspektivischen Verzerrung des Markers bestimmt wird. Vielmehr wird die Skalierung von der Projektion des Aurco-Markers auf das Bild zu den gegebenen geometrischen Maßen bestimmt. Hierbei fallen Abbildungsungenauigkeiten und -fehler wesentlich weniger ins Gewicht.
Die optische Distanzbestimmung konnte bei allen Messungen überzeugen, der größte Feh-

ler lag lediglich bei rund 36 Zentimetern. Da dieser Wert jedoch bei einer Verdrehung von 35° der Kamera zum Objekt ermittelt wurde und dieser Winkel sich durch Eindrehen zum Objekt nach dessen Erkennung verkleinern würde, ist dieser Wert eher zu vernachlässigen. Alle anderen Messungen ergaben einen Fehler kleiner als 24 Zentimeter, was für eine optische Distanzermittlung ein sehr zufriedenstellender Wert ist.

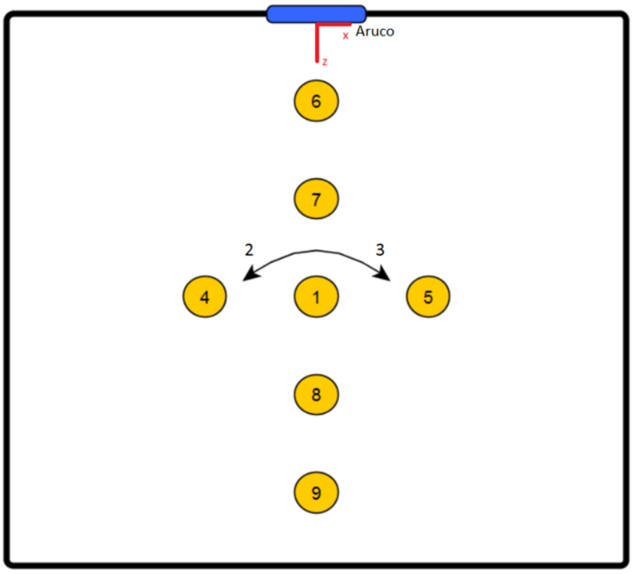
Bei den Distanzermittlungen mittels Time of Flight ergaben sich die größten Abweichungen, wenn die Kinect verdreht zum Objekt stand, ähnlich wie bei vorheriger Betrachtung der Differenzen in x-Richtung. Bei allen anderen Messreihen konnte vor allem die Entfernungsbestimmung durch Auswertung der Ecken des Markers überzeugen. Kein Fehler war größer als 5 Zentimeter, welches bei einer maximalen Entfernung von 5 Metern zum Objekt sehr beachtlich ist. Auch die Korrektur der Werte durch die Auswertung der Mittendistanz von Objekt zu Kamera brachte eine Verbesserung der Positionierung. Lediglich bei zu großer Verschiebung in x-Richtung zum Marker wurden die optisch bestimmten Entfernungen verschlechtert. Daher können zumindest bei der Bestimmung der z-Position in Bezug auf den Marker die Time of Flight Messungen zu einer Verbesserung hinzugezogen werden. Jedoch sollte beachtet werden, dass die Orientierung zum betrachteten Objekt nicht größer als 15° ist.

Aus den bisherigen Betrachtungen zur Kinect lässt sich zusammenfassend sagen, dass die optische Bestimmung der Distanz zum betrachteten ArUco-Marker sich als die beste Methode herausgestellt hat. Da die Fehler ab einem Winkel von 15° von Marker zur Bildmitte mit bis zu zwei Metern relativ groß werden, ist die Positionierung mittels Kinect jedoch für große Orientierungsdifferenzen von Kameralinse zum Objekt ungeeignet. Bei einer frontalen Betrachtung hingegen, kann die Positionierung mithilfe der Time of Flight Methode verbessert und eine Genauigkeit der Position von unter 10 Zentimetern erreicht werden. Leider haben sich in den meisten sonstigen Fällen Tiefenwerte der Kinect als unbrauchbar herausgestellt, da diese die Positionierung eher verschlechtert als verbessert haben.

Lenovo Phab 2 Pro

Auch für das Lenovo Phablet wurden Messungen von verschiedenen Positionen mit variierender Orientierung und Distanz zum Aruco-Marker durchgeführt. Da die Microsoft Kinect jedoch primär ausgewertet werden sollte und das Smartphone nur ergänzend zur Betrachtung der Positionierung auf mobilen Geräten hinzugezogen wurde, wurden nur neun Messreihen aufgenommen. Weiterhin ist der optische Bereich, welcher die RGB-Kamera des Phablet aufnehmen kann nicht so groß wie bei der Kinect, weswegen viele der vorherigen Messungen nicht durchgeführt werden konnten. Die Positionen zur Aufnahme der Messdaten sind in Abbildung 5.6 zu sehen.

Wie auch bei der Kinect wurden jeweils elf Werte mittels optischer Positionsbestimmung und Time of Flight Messung von Mittelpunkt und Ecken des Markers pro Position aufgenommen. Hierbei konnten als Tiefeninformation jedoch die Werte der Punktwolke verwendet werden, weswegen die Distanzbestimmung über den Mittelpunkt des ArUco-Markers unabhängig von der optischen Positionsbestimmung ist. In der Abbildung 5.1 sind für die erste Messung (3m Abstand in z-Richtung) die Differenzen der elf Messwerte zum Referenzpunkt aufgetragen.



Mess-reihe	Orientierung / Distanz (mit Richtung) zum Marker
1	3m (z)
2	-15°
3	15°
4	-1m (x)
5	1m (x)
6	1m(z)
7	2m(z)
8	4m (z)
9	5m (z)

Abbildung 5.6.: Position des Lenovo Phab 2 Pro für die jeweiligen Messreihen

Es zeigt sich, dass sowohl die Messwerte für optische Entfernungsbestimmung, als auch die Distanzermittlung durch Auswertung der Punktwolke der Mitte des Markers kaum Schwankungen unterliegen. Bei der optischen Methode liegen sogar nur zwei verschiedene Wertepaare vor, welche einen Abstand von circa zwei Zentimetern besitzen. Dies zeigt, dass die Ergebnisse sehr zuverlässig sind. Anders sieht es bei der Positionsbestimmung durch Auswertung der Ecken des Markers aus. Sowohl in x-, als auch in z-Richtungen fluktuieren die Werte um viele Zentimeter. Interessanterweise zeigt sich in der späteren Betrachtung, dass durch die vorgenommene Median- und Mittelwertbildung sehr gute Ergebnisse bezüglich der Genauigkeit erzielt wurden.

Das Lenovo Phab 2 Pro konnte mit wesentlich genaueren Positionierungen gegenüber der Kinect überzeugen. Zwar wurden nicht so viele Extremszenarien wie Verschiebungen von über einem Meter in horizontaler Richtung zum Marker oder Orientierungswerte von über 15° von Kamera zum Objekt aufgenommen, jedoch konnten bei sämtlichen anderen Messreihen durchweg bessere Ergebnisse erzielt werden. Dies liegt wahrscheinlich an der neueren Technik, welche im Lenovo Smartphone verbaut ist, da dieses zwei Jahr später auf den Markt kam als die Microsoft Kamera. Der geringste durchschnittliche Fehler der horizontalen Distanzbestimmung von 22,8 Zentimetern wurde mittels Distanzbestimmung durch Eckenauswertung des Markers erzielt. Die größte Abweichung mit jeweils rund 75 Zentimetern wurde mit der zweiten Methode bei Variation der Orientierung von Kamera zu Objekt gemessen. Dies liegt daran, dass hierbei reine Tiefendaten mit Bezug auf die Bildmitte ausgewertet werden. Da sich diese durch ändern der Orientierung verschiebt, ändern sich dementsprechend auch die Werte der Punktwolke. Lässt man jedoch die Messreihen 2 und 3 außen vor, ist die Methode der Punktwolkenextraktion der Mitte des Markers den anderen beiden vorzuziehen, da der maximale Fehler nie einen Wert von zehn Zentimetern überschreitet und mit einer Ausnahme immer die genauesten Positionen im Bezug zur Referenz ermittelt werden konnten. Ähnlich der vorangegangenen Auswertung zur Microsoft Kinect sind die Fehler bei Bestimmung der z-Position beim

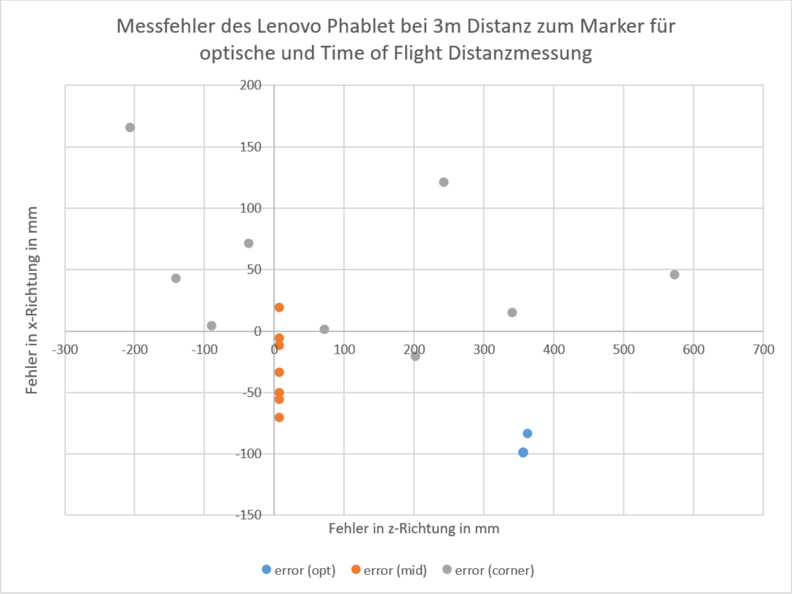


Abbildung 5.7.: Abweichung der Messwerte zum Referenzpunkt (0,0) für 3m Distanz zum Marker

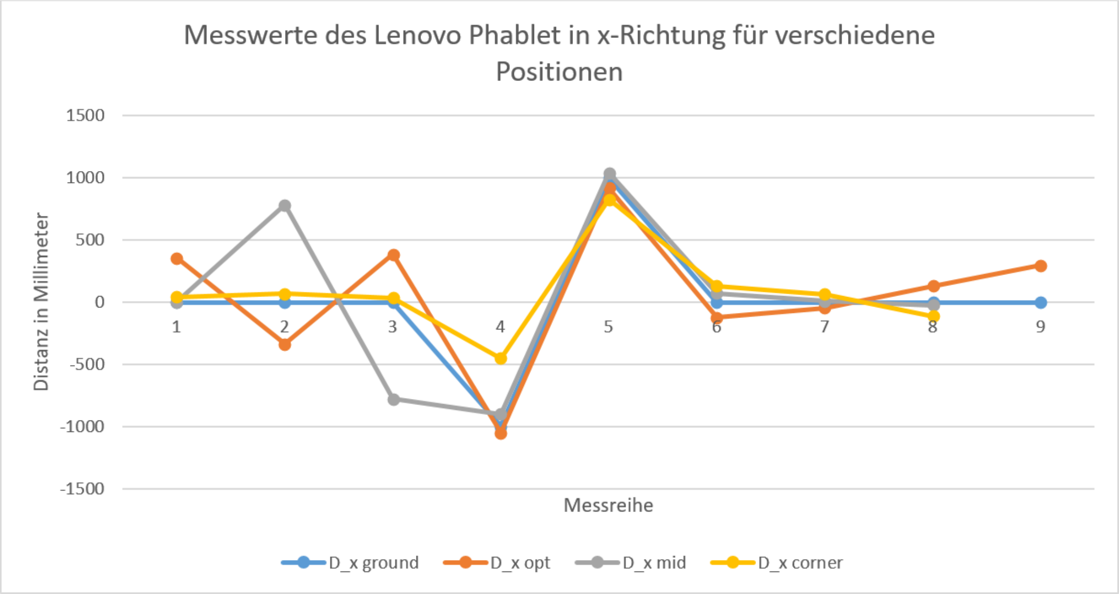


Abbildung 5.8.: Distanzwerte in x-Richtung bezüglich des ArUco-Markers für alle Messreihen des Lenovo Phablet

Lenovo Phab 2 Pro geringer als in horizontaler Richtung zum Objekt. Die Messergebnisse sind in Abbildung 5.9 zu sehen. Da keine Methode einen größeren durchschnittlichen Fehler als neun Zentimeter besitzt, kann jede als sehr zuverlässig betrachtet werden. Mit einem Fehler von zwei beziehungsweise drei Zentimetern sind die Time of Flight Messungen fast perfekt. Lediglich der Distanzbereich ist auf maximal 4,5 Meter zum Objekt beschränkt. Da die optische Ermittlung jedoch selbst bei fünf Metern nur knapp zehn

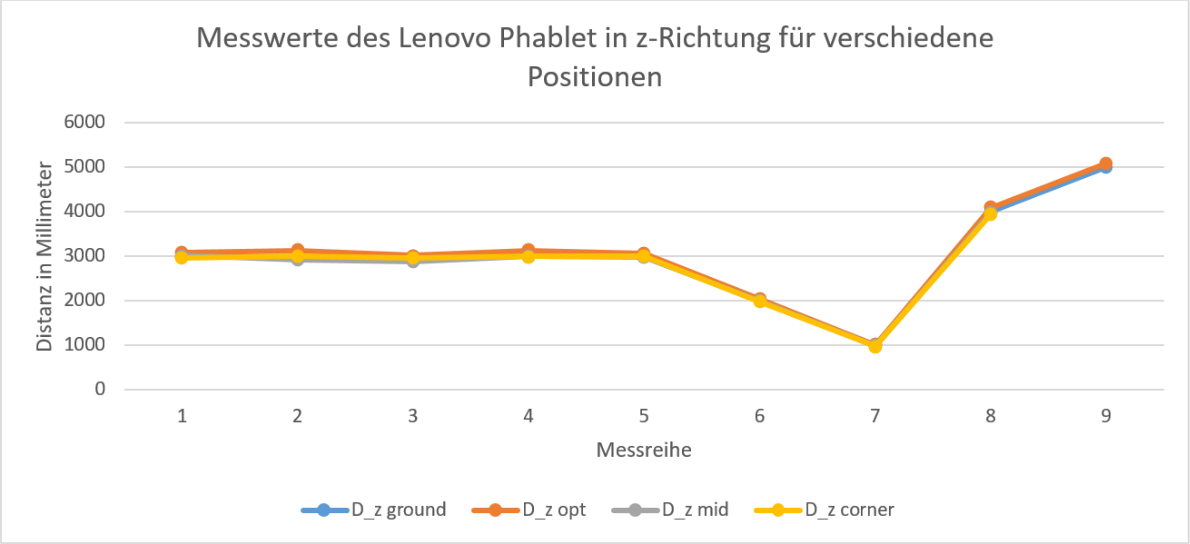


Abbildung 5.9.: Distanzwerte in z-Richtung bezüglich des ArUco-Markers für alle Messreihen des Lenovo Phablet

Zentimeter vom realen Wert abweicht, kann diese Einschränkung außer Acht gelassen werden.
Somit zeigt sich erneut, dass die hier verbauten Kameras wesentlich bessere Ergebnisse als die Kinect liefern. Mit einer durchschnittlichen Abweichung von 2,4 Zentimeter stellte sich abermals die Positionierung über Auswertung der Ecken als die beste Methode heraus. Jedoch ist bei Auslassen der zweiten und dritten Messreihe die Mittenmessung der Eckenmessung vorzuziehen, da hier durchweg genauere Ergebnisse erzielt wurden.
Nach Auswertung der Messergebnisse kann festgestellt werden, dass mithilfe des Lenovo Phab 2 Pro durchweg gute, bis sehr gute Positionierungsergebnisse erzielt wurden. Besonders in z-Richtung konnten die Tiefendaten des Time of Flight Sensors überzeugen, welche mit einer Ausnahme eine maximale Abweichung von sechs Zentimetern besaßen. Doch auch bei Betrachtung der Ergebnisse der horizontalen Distanzermittlungen kann ein positives Fazit gezogen werden. Durch die Kombination von Eckenauswertung des Markers bei Orientierungen größer als 10° von Kamera zu Marker, sowie der Positionierung durch Punktwolkeninformation aus der Mitte des Markers würde der maximale Fehler bei sieben Zentimetern liegen. Die optische Bestimmung der Distanz erzielte auch akzeptable Resultate. Der durchschnittliche Fehler von 28 Zentimetern ist für eine Positionierung hinreichend genau und kann deswegen auch auf Smartphones mit gleicher RGB-Kameraqualität ohne ToF Sensor verwendet werden.

5.2. Bestimmung der Orientierung

Neben der Distanz wurde außerdem die Orientierung von Kamera zu ArUco-Marker bestimmt. Anhand der Tabelle 5.2 ist zu erkennen, dass die geringsten Abweichungen vorliegen, wenn die Kamera frontal auf den Marker ausgerichtet war. Der einzige Aus-

reißer liegt bei Messreihe 14 vor. Hierbei berechnete der Algorithmus einen Winkel von 28,9°, obwohl nur eine Verschiebung in x-Richtung zum Marker stattgefunden hat. Der Grund dafür liegt in einer fehlerhaften Projektion der Eckpunkte des Markers auf das Bild.
Wurde die Kamera zur Landmarke verdreht, ist bis auf eine Ausnahme bei beiden Systemen die berechnete Orientierung immer geringer als die reale Orientierung.

Tabelle 5.2.: Orientierungsbestimmung zum Referenzpunkt

Referenz-	**Orientierung**	
orientierung [°]	**Kinect** [°]	**Phab2Pro** [°]
0	1,84	6,83
15	9,01	8,58
30	15,68	
35	22,0	
-15	-1,53	-7,65
-30	-18,55	
-35	-42,23	
0	7,36	4,31
0	1,28	-1,53
0	3,72	0,18
0	0,26	0,01
0	2,18	
0	-1,69	1,19
0	28,85	1,48
0	-0,495	

5.3. Raumnummernerkennung

Als zweiter Aspekt soll neben Betrachtung der Positionierungsgenauigkeit in x- und z-Richtung eine Auswertung der Raumnummernschilderkennung stattfinden. Anders als ArUco oder ähnliche Marker besitzen Raumschilder keine besonderen Muster die eine vereinfachte Erkennung oder sogar Fehlerkorrektur ermöglichen. Jedoch sind Türschilder fest in der Gebäudeinfrastruktur vorhanden, was eine erfolgreiche Erkennung die Positionierung im Gebäude erheblich erleichtert, da keine oder nur wenige zusätzliche Marker angebracht werden müssen.
Die Auswertung findet in zwei Schritten statt. Zuerst wird eine Aussage darüber getroffen, mit welcher Genauigkeit die Zeichen der Raumidentifikation erkannt und richtig klassifiziert werden. Schließlich ermöglicht nur eine exakte Extraktion und Vorhersage der Buchstaben und Ziffern eine korrekte Positionierung, da jedem Schild anhand seiner ID die jeweilige Position in der Datenbank zugeordnet ist. Als zweiter Schritt folgt eine Überprüfung der richtigen Erkennung des Türschildes. Dieses wird zur Distanzermittlung von Kamera zum Objekt benötigt und folgt der gleichen Methode, welche bei den ArUco-Markern angewandt wird. Aus diesem Grund werden für die Türschilder keine

weiteren Auswertungen bezüglich der Genauigkeit der Positionierung von Kamera zum Türschild durchgeführt.

Erkennung und Klassifizierung der Raumnummer

Für die Auswertung wurde ein Datensatz von 128 Raumnummern mit und ohne Türschildern generiert. Dieser beinhaltet neun verschiedenen Arten von Schildern, welche in den Gebäudeteilen der TU Chemnitz aus unterschiedlichen Positionen aufgenommen wurden. Der Datensatz wurde schließlich als Eingang in die Türschilderkennung gegeben und überprüft, ob die Vorhersage der SVM oder einer der Vorhersage ähnlichen Zeichenfolge (siehe 4.4.4) mit der Türnummer übereinstimmt. Falls dies zutrifft, wurde es als richtige Erkennung gewertet. Anderenfalls folgte eine Unterscheidung zwischen *keine Türnummer erkannt* oder *falsche Zeichenfolge klassifiziert.* Falls in einem Bild mehrere Schilder abgebildet waren, so reichte eine richtige Klassifizierung um als positive Erkennung in die Auswertung einzugehen. Das Ergebnis ist nachfolgend in Tabelle 5.3 abgebildet.

Es zeigt sich, dass die Raumnummern mit einer Trefferrate von rund 71% richtig erkannt werden. Dies ist ein guter Wert wenn man in Betracht zieht, dass auch einige schwer detektierbare Türschilder in der Datenbank vorlagen. Am häufigsten wurde das Glasschild fehl klassifiziert. Hierbei wurden vor allem zusätzliche Zeichen aufgrund der Halterung erkannt, welches in Abbildung 5.10 zu sehen ist. Auch wurde diese Schildart wegen der Ähnlichkeit zum Hintergrund und somit fehlender Konturen nicht als mögliches Türschild erkannt.

Tabelle 5.3.: Vergleich von richtig und falsch erkannten Raumnummern

Art der Detektion	**Anzahl**	**Anteil am Gesamtdatensatz [%]**
richtig erkannt	91	71,09
Falschklassifikation	5	3,91
zusätzlichen Zeichen erkannt	14	10,94
Nichtdetektion der Nummer	18	14,06

Ebenfalls machen zusätzliche Zeichen vor und hinter der Raumnummer eine richtige Klassifizierung schwierig, da diese, falls sie zu nah am Schild sind, fälschlicherweise als Teil der Raumnummer erkannt werden können. Zu Problemen führt auch der Fall, dass sich ein Zeichen zu weit von den anderen entfernt befindet. Somit muss ein Kompromiss für die Suchreichweite nach möglichen zusammenhängenden Zeichen gefunden werden. Nach einigen Durchläufen wurde mit der dreifachen Höhe des Startzeichens der Kontur der optimale Wert gefunden.

Die kleinste Gruppe innerhalb der Fehlklassifizierungen ist die falsche Vorhersage des richtigen Zeichens durch die SVM. Dieses wurde hauptsächlich durch eine sehr starke perspektivische Verzerrung oder auch Rotation der Buchstaben und Ziffern verursacht.

Zusammenfassend lässt sich sagen, dass bei guter Sicht auf das Türschild die Raumnummer korrekt erkannt wird. Befindet sich die Raumnummer jedoch auf schwierigen Hintergrund, so ist eine richtige Klassifizierung womöglich erst mit mehreren Bildern oder eventuell auch gar nicht gewährleistet.

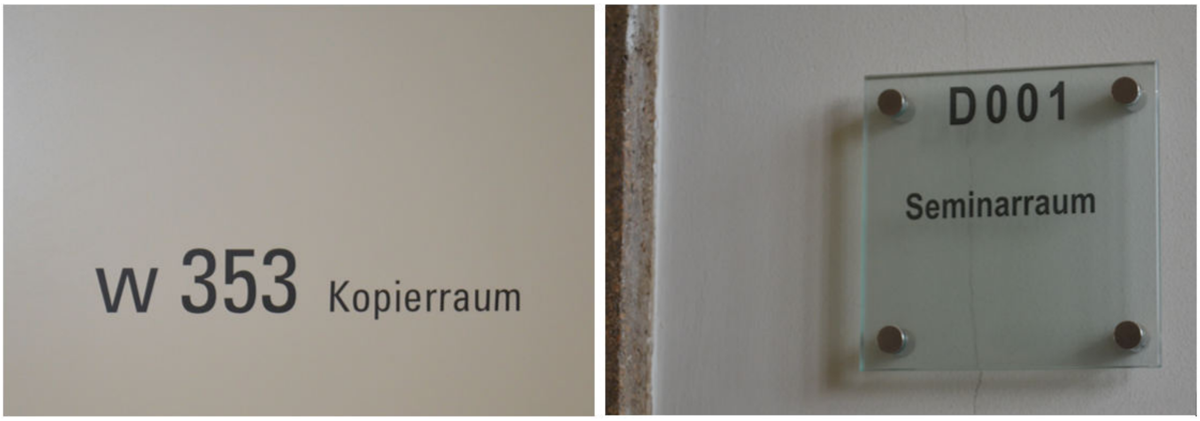


Abbildung 5.10.: Fehlerkennung der Raumnummer durch zusätzliche Zeichen; links: Beschriftung des Raumes zu nah an Raumnummer; rechts: Halterung des Glasschildes wird als Zeichen erkannt

Extraktion des Türschildes

Um eine Aussage über die richtige Erkennung des die Raumnummer umgebenden Türschildes machen zu können, wurden nur diejenige Bilder des Datensatzes verwendet, in welchen auch eine Raumnummer erkannt wurde. Dabei spielte es keine Rolle, ob sie richtig klassifiziert wurden, es musste lediglich als potentielles Türschild erkannt werden. Ebenfalls sind Bilder mit Glasschildern aussortiert worden, da eine Kantenextraktion aufgrund des minimalen Unterschieds zum Hintergrund fast unmöglich ist. Bei den verbliebenen 74 Einträgen wurde optisch überprüft, ob eine richtige Extraktion der Ecken des Türschildes stattgefunden hat. Hierfür wurden die gefundenen Rahmen des Schildes mithilfe von OpenCV rot nachgezeichnet. Stimmte dieser mit der vorhandenen Struktur überein, ist der Eintrag als positives Ergebnis eingegangen. Wurde mindestens eine Ecke nicht oder falsch erkannt, ist das Bild negativ gewertet worden. Auch eine Nichtdetektion zählte in diese Kategorie.

Nach dem Durchlaufen des Algorithmus mit gegebenen Testbildern wurden folgende Ergebnisse erreicht:

Bei etwas weniger als der Hälfte der Schilder konnte die geometrische Figur erfolgreich extrahiert werden. Dieser Wert erscheint wenig, jedoch wird ein in der Datenbank vorhandenes Objekt auf dem Bildschirm eingezeichnet. Bewegt sich der Benutzer nun vor das Schild, ist eine bessere Konturenerkennung wahrscheinlicher. Zwar ist besonders bei Schildern mit Glashintergrund die Formdetektion auch trotz direktem Sichtkontakt nicht

Tabelle 5.4.: Vergleich von richtig und falsch extrahierten Konturen des Türschildes

Art der Detektion	**Anzahl**	**Anteil am Gesamtdatensatz** [%]
richtig erkannt	34	45,94
Falschdetektion der Ecken	20	27,03
Nichtdetektion des Schildes	20	27,03

gegeben, jedoch könnte man hier beispielsweise einen festen Positionswert im Bezug auf das Schild vorgeben, wenn aufeinanderfolgend keine Konturen extrahiert werden konnten.
Wie bereits angedeutet, wurden bei einer direkten Frontalaufnahme des Türschildes die besten Ergebnisse erzielt. Bei starker Verzerrung konnte, falls überhaupt eine mögliche Raumnummer erkannt wurde, nur in zwei Fällen eine Kontur erkannt werden. Auch war es unmöglich bei einem mehrfarbigen Türschild die äußere Form exakt zu bestimmen, da die Trennlinien zwischen den Farben meist auch als Kante detektiert werden. Dies ist zum Beispiel in Abbildung 5.11 zu sehen.
Abschließend lässt sich ein positives Fazit aus den Testwerten ziehen. Trotz der hohen Fehlerraten von circa 25% bei der Raumnummererkennung und rund 55% bei der Kantenextraktion des Türschildes können diese durch spätere Frames mindestens einmal erkannt werden. Diese Detektion wird optisch auf dem Bildschirm hervorgehoben und der Nutzer kann sich zu dem jeweiligen Objekt begeben. Bei frontaler Ansicht konnten beinahe alle Türschilder erfolgreich bestimmt werden, weswegen eine genaue Positionsbestimmung anhand dieser Landmarken möglich ist.

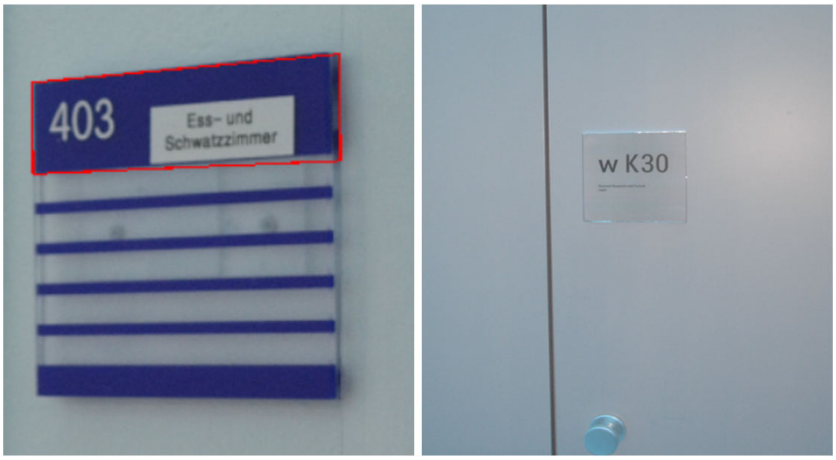


Abbildung 5.11.: links: falsche Extraktion des Türschildes durch Farbunterschide auf Selbigem; rechts: keine Erkennung des Schildes aufgrund zu großer Ähnlichkeit zu Hintergrund

5.4. Zeitkritische Betrachtung des Algorithmus

Zuletzt folgt eine genauere zeitliche Betrachtung des Positionierungsprogrammes. Um eine zeitliche Betrachtung aller Teilsysteme machen zu können, wurden sowohl für die Kinect, als auch für das Smartphone ein Video inklusive Aruco-Marker aufgenommen. Somit sind neben der optischen Detektion und Extraktion auch die Datenbankabfrage und das Auslesen der Tiefenwerte gemessen worden. Die Aufnahmen dauerten einige Minuten und für jedes Modul wurde der Mittelwert gebildet. Somit kann eine Aussage über mittlere Zeit eines Durchlaufs getroffen werden. In nachfolgender Tabelle 5.5 sind die Zeiten für die jeweiligen Module entsprechend nach verwendeter Hardware aufgeschlüsselt.

Unter Tabelleneintrag *Sonstiges* fallen zum Beispiel die benötigte Zeit zur Visualisierung oder zur Freigabe nicht mehr verwendeten Speichers. Eine geringer dennoch sollte

Tabelle 5.5.: Vergleich der Zeitdauer der jeweiligen Module von Kinect und Phab 2 Pro

Modul	**Dauer [ms]**	
	Kinect	**Phab 2 Pro**
Bildgewinnung	17,9	15,8
Vorverarbeitung	6,4	13,8
Aruco-Detektion	33,3	122,0
Raumnummererkennung	37,9	161,9
Datenbankabfrage	18,3	3,4
opt. Distanzberechnung	0,3	2,9
ToF Distanzberechnung	1,3	45,3
Sonstiges	4,6	4,3
Gesamtzeit	86,7	247,4
FPS[1]	11,53	4,06

Wie zu erwarten bedarf die Detektion potentieller Aruco-Marker und Raumschilder die meiste Zeit, wobei Zweiteres wenige Millisekunden länger benötigt als das Finden von ArUcos. Da in OpenCV die Algorithmen bezüglich ihrer Performance stark optimiert sind, ist diese Tatsache jedoch nicht verwunderlich. Durch die parallele Bearbeitung beider Module, konnte die Gesamtzeit wesentlich reduziert werden. Mit rund zwölf FPS ist der Positionierungslagorithmus in der Desktop-Umgebung circa dreimal schneller als auf dem Smartphone. Weil der verwendete Intel Core i5-6300HQ Prozessor wesentlich leistungsfähiger als die im Phab 2 Pro verbaute Snapdragon 652 CPU von Qualcomm ist, war diese Tatsache zu erwarten. Dennoch ist auch bei 4 Frames pro Sekunde eine Echtzeitfähigkeit gegeben, weil zur Positionierung lediglich zehn Bilder mit vorhandenem Marker benötigt werden. Somit wäre eine Lokalisierung innerhalb von 2,5 Sekunden möglich. Da Verzögerungen bei der Anzeige trotzdem erkennbar sind, sollte die Kameraauflösung bei Portierung des Programmes auf andere Android-Geräte reduziert werden. Zwar führt eine geringere Auflösung wahrscheinlich zu einer ungenaueren Positionierung, dennoch sollte der Geschwindigkeitsvorteil diese Tatsache überwiegen, da Bildvorverarbeitung und die Aruco-, sowie Raumnummerndetektion wesentlich weniger Zeit benötigen würde.

6. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Algorithmus zur kamerabasierten Lokalisation auf Basis von Landmarken entworfen und realisiert. Das Programm kann auf den Plattformen Windows und Android betrieben werden und benötigt zur Positionierung eine SQL kompatible Datenbank mit den eingetragenen Landmarken, sowie eine XML-Datei mit den Parametern der in dieser Arbeit verwendeten SVM. Diese Dateien werden neben den Source-Files, sowie dem gebauten Projekt und dessen Abhängigkeiten beigefügt. Zusätzlich wird eine XML-Datei mit intrinsischen Kameraparametern benötigt, sofern diese nicht von der Kameraumgebung bereitgestellt wird. Nach erfolgreicher Initialisierung werden die Bilder auf vorhandene ArUco-Marker und Türschilder untersucht, welche als Landmarken in dieser Arbeit fungieren. Werden potentielle Kandidaten entdeckt, so wird die Datenbank nach den jeweiligen IDs durchsucht. Wenn ein passender Eintrag gefunden wurde, erfolgt die Positionierung anhand der extrahierten Konturen der Marker beziehungsweise Türschilder. Dabei werden die realen geometrischen Maße, welche in der Datenbank vorliegen müssen, mit den gemessenen Daten verglichen. Dies geschieht zum einen durch Lösen des nichtlinearen Ausgleichsproblem der Projektion der jeweiligen Weltkoordinaten auf die Bildkoordinaten. Zum anderen werden die Distanzen mithilfe von Tiefensensoren, welche die Entfernung zum jeweilige Objekt durch beispielsweise die Time of Flight Methode messen, bestimmt. Wurde das gleiche Objekt in nacheinander folgenden Bildern erfolgreich detektiert, werden die jeweiligen Breiten- und Längengrade extrahiert und durch die ermittelte Distanz zur Landmarke ergänzt. Neben dieser Information werden zusätzlich die Orientierung der Kamera zum Objekt, sowie die Etagenangabe zurückgeliefert. Mithilfe einer Microsoft Kinect v2 und einem Lenovo Phab 2 Pro Smartphone wurden Testdaten für verschiedene Positionen und Orientierungen bezüglich der Landmarken aufgenommen und unter Einbeziehung der Ground Truth ausgewertet. Da in allen Fällen eine Position bestimmt werden konnte, welche je nach verwendetem Gerät und jeweiliger Pose meist eine Genauigkeit von unter einem halben Meter zu den Referenzwerten aufwies, ist die Aufgabenstellung zufriedenstellend gelöst wurden.

Darüber hinaus existieren einige Ansätze um diese Arbeit fortzuführen, da bisher zwar eine geografische Position geliefert wird, jedoch keine Positionierung innerhalb des Gebäudes stattfindet. Aus diesem Grund muss der Algorithmus in das bereits bestehende Indoor-Positionierungsframework an der Professur Schaltkreis- und Systementwurf der TU Chemnitz integriert werden. Besonders die Auswertung der Messdaten des Lenovo Phablets zeigte, dass eine genaue Positionierung im Bereich von wenigen Zentimetern möglich ist. Deswegen würde sich dieser Algorithmus hervorragend zur genauen Initialisierung der Position eignen. Da die Indoor-Navigation jedoch vorwiegend mithilfe von portablen Geräten wie Tablets oder Smartphones erfolgt, sollte der vorhandene Algorithmus so in die Android Umgebung integriert werden, dass eine Benutzung ohne Verwendung des Google Tango Frameworks möglich ist. Schließlich ist das Lenovo Phab 2 Pro das bisher einzigw verfügbare Consumer-Smartphone, welche diese Technologie einsetzen kann. Deswegen ist der bisherige Nutzen des Positionierungsprogramms auf der Android Plattform sehr eingeschränkt. Weiterhin werden die Tiefendaten nur unterstützend verwendet und sind somit zur Lokalisierung nicht erforderlich. Ein weiterer Verbesserungsvorschlag ist die unterstützende Anzeige von Augmented-Reality Objekten auf dem Bildschirm. Dies können zum Beispiel Informationen zu einem Raum bei erkanntem Türschild sein. So wäre es möglich Sprechzeiten oder auch Informationen zu den Personen anzuzeigen, welche in dem Raum ihren Sitz haben. Zum anderen könnte bei einer Navigation im Gebäude der Weg direkt auf dem Bildschirm angezeigt werden, so dass der Nutzer optisch bei seiner Wegfindung unterstützt werden kann. Zuletzt sollte auch der Fall betrachtet werden, wenn eine Raumnummer kein Türschild besitzt oder dieses aufgrund von zu starker Ähnlichkeit mit der Umgebung nicht erkannt wird. Da dennoch die geografische Position der Landmarke bestimmt werden kann, sollte ein fester Wert vorgegeben werden, welcher die Distanz der Kamera zum Objekt angibt. Eine Ermittlung der Pose über die Größe der Schrift erscheint als nicht praktikabel, da sonst für jeden Raumnummer aufgrund von verschiedenen Buchstaben und Zahlen auch die unterschiedlichen geometrischen Maße gemessen und gespeichert werden müssten.

Somit zeigt sich, dass im Bereich der optischen Positionierung Potential zur Optimierung gegeben ist. Besonders die Einbindung der Augmented Reality ist interessant, da dem Nutzer viele zusätzliche Informationen aus seiner näheren Umgebung geliefert werden können. Dieses führt neben einer verbesserten Nutzerfreundlichkeit auch zu einer besseren Orientierung im Gebäude.

Literaturverzeichnis

[AASAe16] ALARIFI, A. ; AL-SALMAN a. ; ALSALEH, M. ; ET al: Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances. 16(5) (2016), May, S. 751

[AGD07] AZAD, P. ; GOCKEL, T. ; DILLMANN, R.: *Computer Vision - das Praxisbuch.* Elektor-Verlag, 2007. – ISBN 9783895761652

[All03] ALLEN, G. D.: *The GPS.* `http://www.math.tamu.edu/~dallen/physics/gps/gps.htm`. Version: 2003. – [Online; besucht am 18.04.2017]

[Ama17] AMAZON: *Amazon Go.* `https://www.amazon.com/b?node=16008589011`. Version: 2017. – [Online; besucht am 01.03.2017]

[And10] ANDREAS: *Orthogonale Matrix.* `http://www.mathebibel.de/orthogonale-matrix`. Version: 2010. – [Online; besucht am 18.03.2017]

[Ash14] ASHLEY, James: *Skeletal Tracking.* `http://www.imaginativeuniversal.com/blog/2014/03/05/Quick-Reference-Kinect-1-vs-Kinect-2/`. Version: 2014. – [Online; besucht am 02.04.2017]

[Bha12] BHARADI, Vinayak A.: *Canny Edge Detection in C#.* `https://www.codeproject.com/Articles/93642/Canny-Edge-Detection-in-C`. Version: 2012. – [Online; besucht am 29.03.2017]

[Bil14] BILDA, Sebastian: *Untersuchung von Methoden der kamerabasierten Sturzdetektion im häuslichen Umfeld*, TU Chemnitz, Bachelorarbeit, 2014

[Bou16] BOUNDLESS: *Color Vision.* `https://www.boundless.com/physics/textbooks/boundless-physics-textbook/vision-and-optical-instruments-25/the-human-eye-172/color-vision-620-11256/`. Version: 2016. – [Online; besucht am 19.03.2017]

[CY11] CAMPBELL, ICG ; YIMING, Ying: *Learning with Support Vector Machines.* Morgan and Claypool, 2011. – ISBN 9781608456161

[eta16] ETAILMENT.DE: *Wie man im Crosschannel Location Based Services mit dem Marketing verbindet.* `http://etailment.de/news/stories/Wie-man-im-Crosschannel-Location-Based-Services-mit-dem-Marketing-verbindet-4114`, 2016. – [Online; besucht am 01.03.2017]

[GJMSe14] GARRIDO-JURADO, S. ; MUNOZ-SALINAS, R. ; ET al: Automatic Generation and Detection of Highly Reliable Fiducial Markers Under Occlusion. In: *Pattern Recogn.* 47 (2014), Juni, Nr. 6, S. 2280–2292. – ISSN 0031–3203

[GM14] Gao, Wen ; Ma, Siwei: *Advanced Video Coding Systems.* Springer International Publishing, 2014. – ISBN 978–3–319–14243–2, 978–3–319–14242–5

[Goo17a] Google: *GLSurfaceView.Renderer.* `https://developer.android.com/reference/android/opengl/GLSurfaceView.Renderer.html`. Version: 2017. – [Online; besucht am 22.04.2017]

[Goo17b] Google: *Tango Developer Overview.* `https://developers.google.com/tango/developer-overview`. Version: 2017. – [Online; besucht am 15.03.2017]

[Her11] Hermann, M.: *Numerische Mathematik.* Oldenbourg Wissenschaftsverlag, 2011. – ISBN 9783486708202

[Hof09] Hofmann, Tim: *Geometrische Kamerakalibrierung.* `http://www.mi.hs-rm.de/~schwan/Projects/CG/CarreraCV/doku/intrinsisch/intrinsisch.htm`. Version: 2009. – [Online; besucht am 19.03.2017]

[ind17a] indoora: *Indoora – Indoor Positioning Systems – Technology.* `http://www.indoora.com/technology/`. Version: 2017. – [Online; besucht am 03.03.2017]

[Ind17b] IndoorAtlas: *How it works – IndoorAtlas.* `http://www.indooratlas.com/how-it-works/`. Version: 2017. – [Online; besucht am 06.03.2017]

[Inf16] Infineon: *Partner in Google's Project Tango.* `http://www.infineon.com/cms/de/product/sensor/3d-image-sensor-real3/3d-image-sensor-for-consumer/partner-in-googles-project-tango/channel.html?channel=5546d4614937379a0149382fa03c007b`. Version: 2016. – [Online; besucht am 12.03.2017]

[inf17a] infsoft: *Indoor Positioning, Tracking and Indoor Navigation with WiFi.* `https://www.infsoft.com/technology/sensors/wifi`. Version: 2017. – [Online; besucht am 03.03.2017]

[inf17b] infsoft: *infsoft blog - indoor positioning & more.* `https://www.infsoft.com/blog-en/articleid/41/indoor-navigation-indoor-positioning-using-bluetooth`. Version: 2017. – [Online; besucht am 01.03.2017]

[Kow12] Kowalk, Prof. Dr. W.: *Hammingbedingung.* `http://einstein.informatik.uni-oldenburg.de/rechnernetze/seite11.htm`. Version: 2012. – [Online; besucht am 15.03.2017]

[Lau13] Lau, Daniel: *The Science Behind Kinects or Kinect 1.0 versus 2.0.* `http://www.gamasutra.com/blogs/DanielLau/20131127/205820/The_Science_Behind_Kinects_or_Kinect_10_versus_20.php`. Version: 2013. – [Online; besucht am 11.03.2017]

[LDMC+16] La Delfa, Gaetano C. ; Monteleone, Salvatore ; Catania, Vincenzo ; De Paz, Juan F. ; Bajo, Javier: Performance analysis of visualmarkers for indoor navigation systems. In: *Frontiers of Information Technology & Electronic Engineering* 17 (2016), Nr. 8, S. 730–740. – ISSN 2095–9230

[Len17] Lenovo: *Lenovo Phab 2 Pro*. `http://shop.lenovo.com/de/de/tango/`. Version: 2017. – [Online; besucht am 05.04.2017]

[Lor16] LorenaGdL: *SVM predict multiclass*. `http://answers.opencv.org/question/98714/svm-predict-multiclass/`. Version: 2016. – [Online; besucht am 25.03.2017]

[LSVW11] Link, J. Á. B. ; Smith, P. ; Viol, N. ; Wehrle, K.: FootPath: Accurate map-based indoor navigation using smartphones. In: *International Conference on Indoor Positioning and Indoor Navigation*, 2011, S. 1–8

[Lyn08] Lynch, Kevin: *The image of the city*. M.I.T. Press, 2008. – ISBN 0262120046

[Mö16] Möllenhoff, Stefan: *Time of Flight: Real3-Kamera von Infineon und PMB erklärt*. `http://www.techstage.de/ratgeber/Time-of-Flight-Real3-Kamera-von-Infineon-und-PMB-erklaert-3234565.html`. Version: 2016. – [Online; besucht am 13.03.2017]

[MBL+09] Mazuelas, S. ; Bahillo, A. ; Lorenzo, R. M. ; Fernandez, P. ; Lago, F. A. ; Garcia, E. ; Blas, J. ; Abril, E. J.: Robust Indoor Positioning Provided by Real-Time RSSI Values in Unmodified WLAN Networks. In: *IEEE Journal of Selected Topics in Signal Processing* 3 (2009), Oct, Nr. 5, S. 821–831. – ISSN 1932–4553

[MGHX15] Ma, Rui ; Guo, Qiang ; Hu, Changzhen ; Xue, Jingfeng: An Improved WiFi Indoor Positioning Algorithm by Weighted Fusion. In: *Sensors 2015* 15 (2015), Aug, S. 21824–21843. – ISSN 1424–8220

[MHST09] Mete, Mutlu ; Hennings, Leah ; Spencer, Horace J. ; Topaloglu, Umit: Automatic identification of angiogenesis in double stained images of liver tissue. In: *BMC Bioinformatics* 10 (2009), Nr. 11, S. S13. – ISSN 1471–2105

[Mic13] Microsoft: *Skeletal Tracking*. `https://msdn.microsoft.com/en-us/library/hh973074.aspx`. Version: 2013. – [Online; besucht am 04.04.2017]

[Mic16] Microsoft: *Microsoft Indoor Localization Competition – IPSN 2016*. `https://www.microsoft.com/en-us/research/event/microsoft-indoor-localization-competition-ipsn-2016/`. Version: 2016. – [Online; besucht am 04.03.2017]

[Mik06] Mikhalenko, Peter V.: *So funktioniert das Java Native Interface*. `http://www.zdnet.de/39147520/so-funktioniert-das-java-native-interface/`. Version: 2006. – [Online; besucht am 12.04.2017]

[Moo17] MOORHEAD, Patrick: *Lenovo Shows Off PHAB2 Pro, First Google Tango Commercial Device, At Lenovo Tech World.* `https://www.forbes.com/sites/patrickmoorhead/2016/06/09/lenovo-shows-phab2-pro-first-tango-commercial-device-at-lenovo-tech-world/#6fc2921d33aa`. Version: 2017. – [Online; besucht am 10.03.2017]

[Nov12] NOVRIANTO, Reky: *Methoden zur Lokalisierung von drahtlosen Endgeräten*, TU Ilmenau, Hauptseminararbeit, 2012

[Ope14a] OPENCV: *Introduction to SIFT (Scale-Invariant Feature Transform).* `http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html`. Version: 2014. – [Online; besucht am 09.03.2017]

[Ope14b] OPENCV: *Introduction to SURF (Speeded-Up Robust Features).* `http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html`. Version: 2014. – [Online; besucht am 09.03.2017]

[Ope14c] OPENCV: *Real Time pose estimation of a textured object.* `http://docs.opencv.org/3.0-beta/doc/tutorials/calib3d/real_time_pose/real_time_pose.html`. Version: 2014. – [Online; besucht am 16.03.2017]

[Ope15a] OPENCV: *Color conversions.* `http://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html`. Version: 2015. – [Online; besucht am 17.03.2017]

[Ope15b] OPENCV: *Feature Detection.* `http://docs.opencv.org/3.1.0/dd/d1a/group__imgproc__feature.html#ga354e0d7c86d0d9da75de9b9701a9a87e`. Version: 2015. – [Online; besucht am 15.04.2017]

[Ope15c] OPENCV: *Image Thresholding.* `http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html`. Version: 2015. – [Online; besucht am 15.04.2017]

[Ope16] OPENCV: *Quelltext der Konvertierungen in verschiedene Farbsysteme.* `https://github.com/opencv/opencv/blob/master/modules/imgproc/src/color.cpp`. Version: 2016. – [Online; besucht am 30.03.2017]

[Ope17] OPENCV: *OpenCV contrib.* `http://www.shopkick.com/shopbeacon`. Version: 2017. – [Online; besucht am 01.03.2017]

[Pro14] PROPHETSAGENCY: *Prophets innovates with iBeacon technology - Bring Rubens back to life.* `http://www.prophets.be/#/work/ibeacon/`. Version: 2014. – [Online; besucht am 14.03.2017]

[RH07] Ringbeck, Thorsten ; Hagebeuker, Bianca: A 3D time of flight camera of object detection. In: *Optical 3-D measurement techniques VIII : applications in GIS, mapping, manufacturing, quality control, robotics, navigation, mobile mapping, medical imaging, cultural heritage, VR generation and animation.* ETH Zurich, July 2007, S. 1 – 16

[SBPNT17] Space-Based Positioning, National Coordination O. ; Navigation ; Timing: *GPS Accuracy.* `http://www.gps.gov/systems/gps/performance/accuracy/`. Version: 2017. – [Online; besucht am 28.02.2017]

[Sch07] Schreer, O.: *Stereoanalyse Und Bildsynthese.* Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2007. – ISBN 354023439X

[SH99] Sorrows, Molly E. ; Hirtle, Stephen C.: The Nature of Landmarks for Real and Electronic Spaces. (1999), S. 37–50. ISBN 978–3–540–48384–7

[sho17] shopkick: *Shopkick shopBeacon™.* `http://www.shopkick.com/shopbeacon`. Version: 2017. – [Online; besucht am 06.03.2017]

[Sim13] Simek, Kyle: *Dissecting the Camera Matrix, Part 2: The Extrinsic Matrix.* `http://ksimek.github.io/2012/08/22/extrinsic/`. Version: 2013. – [Online; besucht am 12.04.2017]

[Ste14] Sterling, Greg: Magnetic Positioning - The Arrival of 'Indoor GPS'. In: *opusresearch Report* (2014)

[wik04] wikipedia: *Lochkamera Prinzip.* `https://upload.wikimedia.org/wikipedia/commons/9/9d/Lochkamera_prinzip.jpg`. Version: 2004. – [Online; besucht am 09.03.2017]

[wik07] wikipedia: *Sobel Operator.* `https://de.wikipedia.org/wiki/Datei:TOF-Kamera-Prinzip.jpg`. Version: 2007. – [Online; besucht am 15.03.2017]

[wik08] wikipedia: *TOF-Kamera-Prinzip.* `https://en.wikipedia.org/wiki/Sobel_operator#Example`. Version: 2008. – [Online; besucht am 21.03.2017]

[wik09a] wikipedia: *Pineapple and cross section.* `https://en.wikipedia.org/wiki/File:Pineapple_and_cross_section.jpg`. Version: 2009. – [Online; besucht am 14.03.2017]

[wik09b] wikipedia: *Verzeichnung.* `https://de.wikipedia.org/wiki/Verzeichnung#/media/File:Verzeichnung3.png`. Version: 2009. – [Online; besucht am 21.03.2017]

[wik13] wikipedia: *Single frame YUV420.* `https://en.wikipedia.org/wiki/YUV#/media/File:Yuv420.svg`. Version: 2013. – [Online; besucht am 14.03.2017]

[wik14] WIKIPEDIA: *Kinect for Xbox One Sensorleiste.* `https://de.wikipedia.org/wiki/Kinect#/media/File:Xbox-One-Kinect.jpg`. Version: 2014. – [Online; besucht am 05.04.2017]

[wik17a] WIKIPEDIA: *Erdradius.* `https://de.wikipedia.org/wiki/Erdradius#Erdumfang`. Version: 2017. – [Online; besucht am 23.03.2017]

[wik17b] WIKIPEDIA: *Lat-Long picture.* `https://en.wikipedia.org/wiki/Geographic_coordinate_system#/media/File:FedStats_Lat_long.png`. Version: 2017. – [Online; besucht am 22.03.2017]

[WKM11] WERNER, M. ; KESSEL, M. ; MAROUANE, C.: Indoor positioning using smartphone camera. In: *2011 International Conference on Indoor Positioning and Indoor Navigation*, 2011, S. 1–6

[Zha00] ZHANG, Zhengyou: A Flexible New Technique for Camera Calibration. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22 (2000), November, Nr. 11, S. 1330–1334. – ISSN 0162–8828

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, den 24. April 2017

Sebastian Bilda

Anhang

A. Tabellen

Auf den folgenden Seiten werden alle aufgenommenen Messwerttabellen aufgelistet. Diese beinhalten sowohl die Position, als auch die Orientierung von der Microsoft Kinect v2, als auch dem Lenovo Phab 2 Pro bezüglich des verwendeten ArUco-Marker. Für die spezifischen Werte wie die Time of Flight Messung wird auf die beiliegende CD verwiesen.

Tabelle A.1.: Differenzen der Positionsbestimmung in x-Richtung zum Referenzpunkt mittels Kinect

Mess-reihe	D_x **Ref[mm]**	D_x **opt Diff[mm]**	D_x **mid Diff[mm]**	D_x **corner Diff[mm]**
1	0	80,756	78	-23,6223
2	35	-242,245	-224	-712,957
3	30	-797,283	-649	-1380,54
4	15	-1221,88	-972	-1296,19
5	-15	-93,8378	-88	806,721
6	-30	-871,701	-735	1182,83
7	-35	1709,97	1.710	N/A
8	0	549,973	530	67,1811
9	0	162,349	156	-196,676
10	0	160,78	158	-49,3867
11	0	47,131	46	-1,155
12	-1500	233,19	-17	-1526,5845
13	-1000	-32,868	-131	-939,8944
14	1000	1799,971	1.719	979,99
15	1500	-31,08	156	1348,406

Tabelle A.2.: Differenzen der Positionsbestimmung in z-Richtung zum Referenzpunkt mittels Kinect

Mess-reihe	D_z **Ref[mm]**	D_z **opt Diff[mm]**	D_z **mid Diff[mm]**	D_z **corner Diff[mm]**
1	3000	-90,3	14,39	-35,53
2	3024	-102,17	100,11	156,24
3	3036	-86,21	467,57	795,56
4	3042	52,93	657,92	826,04
5	2980	-95,99	134,39	227,03
6	2970	-3,97	480,27	674,05
7	2960	361,88	361,88	N/A
8	5000	-172,53	21,52	70,77
9	4000	-133,69	35,05	-11,6
10	2000	-37,64	2,75	-14,76
11	1000	-32,69	-9,55	-10,96
12	3000	-55,24	388,72	-11,5
13	3000	-190,65	138,83	4,51
14	3000	-233,86	101,1	9,83
15	3000	-18,93	353,6	42,88

Tabelle A.3.: Differenzen der Positionsbestimmung in x-Richtung zum Referenzpunkt mittels Lenovo Phablet

Mess-reihe	D_x **Ref[mm]**	D_x **opt Diff[mm]**	D_x **mid Diff[mm]**	D_x **corner Diff[mm]**
1	0	-355,662	-6,94	-42,92
2	0	335,254	-782,86	-67,40
3	0	-383,713	779,68	-33,14
4	-1000	48,46	-98,15	-548,50
5	1000	79,906	-46,19	169,84
6	0	119,586	-72,49	-127,12
7	0	47,6777	-14,15	-63,31
8	0	-130,065	23,39	111,13
9	0	-298,975	N/A	N/A

Tabelle A.4.: Differenzen der Positionsbestimmung in z-Richtung zum Referenzpunkt mittels Lenovo Phablet

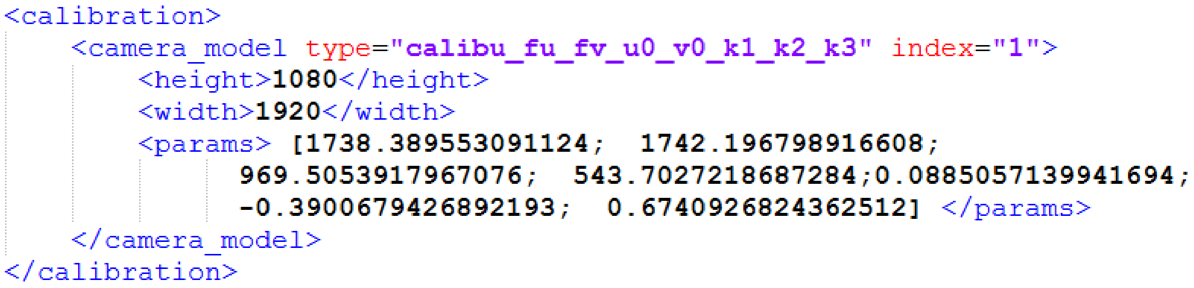
Messreihe reihe	D_z **Ref[mm]**	D_z **opt Diff[mm]**	D_z **mid Diff[mm]**	D_z **corner**
1	3000	-98,97	-23,97	35,61
2	2985	-152,93	57,79	-24,21
3	3015	-8,97	138,02	49,5
4	3000	-136,16	-10,82	-1,95
5	3000	-68,68	9,33	-19,02
6	2000	-57,57	-8,33	7,95
7	1000	-14,19	2,675	10,319
8	4000	-97,45	2,78	43,3
9	5000	-100,56	N/A	N/A

Tabelle A.5.: Orientierungsbestimmung zum Referenzpunkt

Referenz-orientierung [°]	**Orientierung**	
	Kinect [°]	**Phab2Pro** [°]
0	1,84	6,83
15	9,01	8,58
30	15,68	
35	22,0	
-15	-1,53	-7,65
-30	-18,55	
-35	-42,23	
0	7,36	4,31
0	1,28	-1,53
0	3,72	0,18
0	0,26	0,01
0	2,18	
0	-1,69	1,19
0	28,85	1,48
0	-0,495	

B. Kalibrierungsdatei

In der nachfolgenden Abbildung B.1 ist die XML-Datei zu sehen, welche die Kalibrierwerte der in dieser Arbeit verwendeten Microsoft Kinect v2 enthält. Neben der Breite und Höhe des Bildes, enthält die Datei Informationen zu den Brennweiten f_u, f_v in x-, beziehungsweise y-Richtung, dem Bildmittelpunkt $(u_0,\, v_0)$ und den radialen Verzerrungsparametern k_1, k_2, k_3. Die gezeigten internen Kameraparameter wurden mittels der Kalibrierung nach Zhang (siehe [Zha00]) bestimmt.



```xml
<calibration>
    <camera_model type="calibu_fu_fv_u0_v0_k1_k2_k3" index="1">
        <height>1080</height>
        <width>1920</width>
        <params> [1738.389553091124;  1742.196798916608;
              969.5053917967076;  543.7027218687284;0.0885057139941694;
              -0.3900679426892193;  0.6740926824362512] </params>
    </camera_model>
</calibration>
```

Abbildung B.1.: Datei mit Kalibrierwerten der in dieser Arbeit verwendeten Microsoft Kinect v2

C. SVM-Parameter

Alle in Tabelle C.1 gezeigten Parameter der von OpenCV bereitgestellten SVM wurden empirisch bestimmt. Durch den Wert "C_SVC" wird festgelegt, dass bei Ermittlung der linearen Trennebene einzelne Ausreißer für die jeweiligen Klassen erlaubt sind. Die falsch klassifizierten Ausreißer ziehen jedoch eine Bestrafung mit Wert C nach sich. Für die Ermittlung der optimalen Trennebene versucht der Algorithmus den Wert der Bestrafung möglichst klein zu halten. Um die Zeit für das Training zu begrenzen wird die Anzahl der maximalen Iterationen, sowie ein Toleranzwert ϵ vorgegeben.

Tabelle C.1.: SVM-Parameter

Parameter	**Wert**
SVM-Typ	C_SVC
Kernel	linear
C	100
Abbruchbedingung bei Training:	
ϵ	1^{-6}
max Iterationen	100000