Bachelorarbeit

Jannik Martin Iacobi

Smartphone-basiertes Bildmesssystem

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jannik Martin Iacobi

Smartphone-basiertes Bildmesssystem

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 23. März 2016

Jannik Martin Iacobi

Thema der Arbeit
Smartphone-basiertes Bildmesssystem

Stichworte
Bildmesssystem, Smartphone, Android, Eingabeoptimierung, Kameraoptik

Kurzzusammenfassung
Diese Arbeit befasst sich mit dem Entwurf eines Messsystems für die Smartphone-Plattform Android, welches Kamerabilder zur Vermessung nutzt. Es werden die wichtigsten Fehlerquellen eines Systems besonders in Bezug auf die suboptimale Anwendungsumgebung von mobilen Geräten analysiert und bewertet. Außerdem werden Techniken der Optimierung der Kameraoptik betrachtet und ihre Umsetzbarkeit im Rahmen von Androids Schnittstellen diskutiert. Auf Basis dieser Erkenntnisse wird ein prototypisches Messsystem für Android entwickelt. Dabei werden auch Aspekte der Bedienung und insbesondere der Eingabeoptimierung für Smartphone-Touchdisplays betrachtet und angewandt.

Jannik Martin Iacobi

Title of the paper
Smartphone-based Image Measuring System

Keywords
image measuring system, smartphone, android, input optimization, camera optics

Abstract
This thesis contains the design and development of a measuring system that uses camera pictures for measuring and is based on the smartphone platform android. It discusses the most important sources of errors, especially those related to its suboptimal application environment. In addition, techniques of optimizing the camera optics are being analyzed and their feasibility is being discussed in view of androids interfaces. Based on those conclusions a prototype of a measuring system for android is being developed, while also looking at aspects of usability and especially input optimization for smartphones touch displays.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation und Zielsetzung . . . 1
- 1.2 Struktur der Arbeit . . . 3

2 Grundlagen **4**
- 2.1 Anwendungsumgebung Android . . . 4
  - 2.1.1 Camera2 API . . . 4
  - 2.1.2 OpenCV4Android . . . 5
  - 2.1.3 Gewähltes Smartphone . . . 5
- 2.2 Bildmesstechnik . . . 6
  - 2.2.1 Brennweite, Gegenstandsweite und Bildweite . . . 7
  - 2.2.2 Linsenverzeichnung . . . 8
  - 2.2.3 Verkippung . . . 9

3 Mess-Szenarien **11**
- 3.1 Maßstabsfaktor . . . 12
  - 3.1.1 Messmarkenpapier . . . 13
  - 3.1.2 Kalibrierpapier . . . 13
  - 3.1.3 Münze . . . 14
  - 3.1.4 Beliebige Objekte . . . 14
  - 3.1.5 Beliebige Objekte mit Objekterkennung . . . 15
- 3.2 Fokus . . . 16
  - 3.2.1 Einstellbar . . . 17
  - 3.2.2 Feststellbar . . . 18
  - 3.2.3 Bekannt . . . 18
  - 3.2.4 Unbekannt . . . 19
  - 3.2.5 Unterstützung Android . . . 19
- 3.3 Linsenverzeichnung . . . 20
- 3.4 Verkippung . . . 21
- 3.5 Bedienungsaspekte . . . 23

4 Entwurf eines Messsystems **25**
- 4.1 Auswahl der Szenarien . . . 25
- 4.2 Realisierung . . . 29
  - 4.2.1 Architektur . . . 30

4.2.2 Design 32
4.2.3 Implementierung 35
4.2.4 Interface 39
4.3 Messergebnisse 43

5 Fazit **46**
5.1 Zusammenfassung 46
5.2 Bewertung 47
5.3 Ausblick 47

Tabellenverzeichnis

4.1 Zuordnung Buttons -> Funktionen 40
4.2 Messergebnisse 44

Abbildungsverzeichnis

1.1 Beispiel Smartphone-basiertes Bildmesssystem 2

2.1 Verwendetes Smartphone Moto G (2. Gen.) 6
2.2 Kameraaufbau schematisch dargestellt (angelehnt an Darstellung aus [1]) . . . 7
2.3 Schematische Darstellung der Linsenverzeichnungsarten 8

3.1 Stufen des Messvorgangs 11
3.2 Referenzmethoden für den Maßstabsfaktor 14
3.3 Optimierung eines beliebigen Objekts als Referenz 16
3.4 Schematische Darstellung der Fokussierungsgrößen 17
3.5 Messung mit einstellbarem Fokus 18
3.6 Messung mit Verkippungsanzeige 22

4.1 Launcher-Icon des Prototypen 25
4.2 Maßstabsmöglichkeiten 26
4.3 Korrektur von gezogenen Linien mit Winkel- und Kantenoptimierung 28
4.4 Korrektur von gezogenen Linien mit Kreiserkennung 29
4.5 Überblick der Systemkomponenten 31
4.6 Klassendesign der Kamera-Abstraktion 33
4.7 Gesamtes Java-Klassendesign der Android-Applikation 34
4.8 GUI der CameraActivitiy 39
4.9 GUI der MeasureActivitiy 39
4.10 Overlay Tutorial der MeasureActivity 41
4.11 Debug-Modus der Messung 42
4.12 Auswahl des Referenztyps 42
4.13 Messung IDs 1,2 44
4.14 Messung IDs 12,13 44

Listings

4.1 Touch-Interaktions-Methode der Messoberfläche ImageViewWithLine 36
4.2 Berechnung der Pixeldistanz in Millimetern 37

1 Einleitung

Die heutige Messtechnik ist besonders in der Industrie ein breites Feld, was sich stetig in feinere Spezialisierungen aufteilt. Dafür ist eine fortwährende Verbesserung von Sensoren und besonders deren digitale Verarbeitung essentiell. Oft werden dabei mit sehr verschiedenen Technologien sehr ähnliche Probleme gelöst. Beispielsweise kann die Vermessung von Produktionsobjekten auf einem Laufband per Laserabtastung erfolgen um die exakten dreidimensionalen Maße zu erhalten. Eine andere Technologie verwendet Kameras, um die Abmessungen zu überprüfen, wobei aus bestimmten Blickwinkeln gezielte Bilder aufgenommen und daraus einzelne Maße berechnet werden.

Solche Systeme, welche auf Basis von Bildern Objekte vermessen, werden Bildmesssysteme genannt. Sie werden außerdem in Bereichen wie der Landvermessung, oft mit Luftbildaufnahmen, oder der medizinischen Analyse verwendet. Die wichtigste Messung der Bildmesstechnik ist die Distanzmessung zwischen zwei Bildpunkten, die es beispielsweise in metrische Maße zu übersetzen gilt.

1.1 Motivation und Zielsetzung

Bildmesssysteme finden zurzeit überwiegend in der Produktion Verwendung. Dort überprüfen höchst präzise Systeme Werkstücke auf Genauigkeit im Nanometer-Bereich. Sie werden genauestens kalibriert und die Formen der zu überprüfenden Messobjekte sind vollständig bekannt.

Im alltäglichen Leben wird Präzision beim Vermessen von Objekten häufig mit einem Messschieber erreicht. Seine Bedienung ist einfach, bietet im Allgemeinen eine Messgenauigkeit von einem Zehntel Millimeter und er ist in fast jedem Haushalt vorhanden. Allerdings ist auch ein Messschieber nicht optimal für den alltäglichen Gebrauch, denn ein Werkzeug hat man nicht immer in der Tasche und die maximale Größe der zu vermessenden Objekte ist stark begrenzt.

Ein Smartphone ist nahezu jederzeit in Griffnähe und es hat fast immer eine Kamera. Die Idee dieser Arbeit ist es nun, diese Kamera für ein Bildmesssystem zu verwenden und eine Alternative für den Messschieber zu bieten. Zu untersuchen ist dabei insbesondere, wie weit sich die zusätzlichen Messfehler minimieren lassen. Diese Fehler können teilweise der Hardware des Smartphones geschuldet sein oder durch eine Einschränkung der Möglichkeiten des Betriebssystems entstehen (wenn beispielsweise eine Funktion der Kamera nicht per App ansteuerbar ist). Die vermutlich größte Fehlerquelle kann jedoch durch die Anwendung eines ungeschulten Nutzers entstehen. Bei der Interface-Bedienung kann etwas falsch verstanden werden und außerdem ist die freie Kamerahandhabung eine gravierende Fehlerquelle.

In dieser Arbeit soll es vor allem auch um die Bewertung dieser Messfehler-Ursachen gehen, sowie um Methoden, diese Fehler zu reduzieren. Daneben steht die Entwicklung eines prototypischen Bildmesssystems für die mobile Plattform Android im Fokus. Dieser Prototyp soll dem Nutzer auch die Möglichkeit geben, den Fokus der Messung speziell auf eine einfache Bedienung oder auf eine hohe Genauigkeit zu setzen.

Auch wenn die Umsetzung für Android entworfen wird, so sind die diskutierten Methoden auch auf andere mobile Systeme übertragbar, Android soll hier nur ein Anwendungsbeispiel sein.

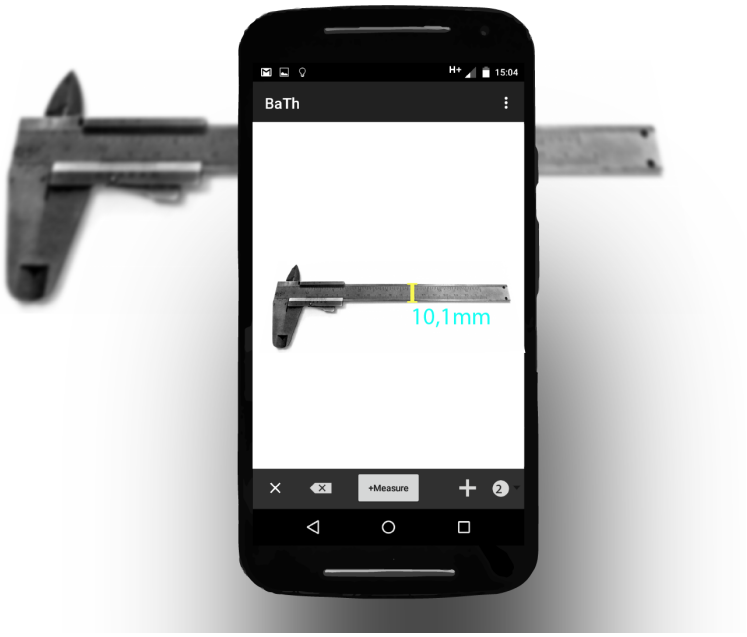

Abbildung 1.1: Beispiel Smartphone-basiertes Bildmesssystem

1.2 Struktur der Arbeit

Nach dieser Einführung werden in Kapitel 2 die Grundlagen des mobilen Betriebssystems Android und die verwendeten Bibliotheken erläutert. Außerdem werden ein gewisses Basiswissen aus den Bereichen der Messtechnik und der Bildverarbeitung vermittelt und die daraus resultierenden Messfehler erklärt.

Im Kapitel 3 werden verschiedene Mess-Szenarien entwickelt, die große Fehlerquellen minimieren und vergleichen. Dabei werden Fehler betrachtet, die durch Hardware, bildverarbeitende Algorithmen sowie freie Handhabung entstehen. In dem Zusammenhang werden auch Bedienungsaspekte eines mobilen Bildmesssystems diskutiert.

Das 4. Kapitel behandelt die Entwicklung eines prototypischen Bildmesssystems als Applikation für das Betriebssystem Android. Die konkrete Auswahl der Optimierungsmethoden für verschiedene Bedienungsmodi wird erläutert und die entwickelte Software inklusive des Entwicklungsprozesses wird vorgestellt. Des Weiteren werden einige Messungen durchgeführt und ihre Ergebnisse diskutiert.

Schlussendlich folgen eine Zusammenfassung und Bewertung der Arbeit sowie ein Ausblick, wie ausgehend von dieser Arbeit Bildmesssysteme in mobiler Anwendung weiter optimiert werden können.

2 Grundlagen

In diesem Abschnitt werden wichtige Grundlagen, die für das Verständnis dieser Arbeit wichtig sind, dargestellt. Dabei werden das verwendete mobile Betriebssystem Android sowie einzelne verwendete Technologien im Detail vorgestellt. Ferner wird ein Einblick in die wichtigsten Aspekte der Bildmesstechnik gegeben.

2.1 Anwendungsumgebung Android

Android ist zur Zeit eines der am Weitesten verbreiteten mobilen Betriebssysteme. "Mehr als eine Milliarde Smartphones und Tablets laufen mit dem Android-Betriebssystem"[2]. Es bietet umfangreiche Programmierschnittstellen (APIs) und eine große Anzahl bereits implementierter Anwendungen, womit sowohl Entwicklern wie auch Nutzern ein großer Markt geboten wird.

Die Entwicklung der Anwendungen (Applications/Apps) erfolgt mit dem Android SDK welches die Android-eigenen APIs umfasst, als Entwicklungsumgebung gibt es die Wahl zwischen einem Eclipse-Plugin oder der eigenen Entwicklungsumgebung Android Studio.

Das Android Studio bietet eine sehr übersichtliche Oberfläche mit vielen Hilfestellungen, die den Entwicklungsprozess von Apps sowie das Live-Debugging der Apps gezielt optimiert. Dadurch wurde ich nicht nur beim Einstieg in die Android-Programmierung unterstützt, sondern hatte auch später eine komfortable Test- und Entwicklungsumgebung zur Hilfe.

2.1.1 Camera2 API

Die wichtigste API für dieses Projekt ist die Camera2 API. Sie ist für die Hardware-Abstraktion der Kameras des Smartphones zuständig. Die Camera2 API ist direkter Bestandteil des Android SDKs und ersetzt die Camera API, wobei sie die Funktionalität stark erweitert.

Die Camera2 API verteilt die Aufgaben der Camera API von einem Interface auf mehrere Interfaces für verschiedene Systemkomponenten und bietet außerdem verschiedene „Komplexitätsebenen“, genannt Hardware support levels. Diese Ebenen vereinfachen dem Anwender die Nutzung bei niedriger Stufe, bieten aber auf hoher Stufe die Möglichkeit sehr direkt auf die Hardware einwirken zu können. Die einzige Einschränkung dabei: Der Hersteller muss die Einstellungsmöglichkeiten freigeben wollen, die für höhere Komplexitätsebenen notwendig sind.

Die niedrigste Stufe, *LEGACY*, ist letztendlich nur ein Wrapper der alten Camera API und unterstützt somit keine neuen Schnittstellen. Die Abfrage von Werten welche höheren Stufen zugeordnet sind, liefert bei einer solchen Kamera keine Ergebnisse und Einstellungs-Befehle der höheren Stufen haben mit LEGACY-Unterstützung auch keine Wirkung. Das höchste Hardware support level *FULL* bietet viele Schnittstellen, die für eine genaue Kontrolle der Kamera notwendig sind. Darunter gibt es Funktionen wie eine manuelle Kontrolle des Sensors und manuelle Nachbearbeitung des Bildes (vgl. [3]), welche für diese Arbeit bezüglich der erreichbaren Genauigkeit sehr vorteilhaft sind.

2.1.2 OpenCV4Android

Eine der bekanntesten Bildverarbeitungs-Bibliotheken ist OpenCV (Open Source Computer Vision), welche für viele wichtige Problemstellungen der Bildverarbeitung Implementierungen bereitstellt um besonders die kommerzielle Nutzung von Bildverarbeitung zu unterstützen (vgl. [4], Seite 18). Da sie kostenlos nutzbar und sehr umfangreich ist, kann die optimalste Problemlösung oft eine eigene Implementierung derselben Problemlösung sein - wenn allerdings die Performance nicht die oberste Priorität des Projektes ist, kann OpenCV eine große Verkürzung der Entwicklungszeit ermöglichen.

Für viele gängige Programmierumgebungen gibt es eine Implementierung von OpenCV, unter anderem gibt es auch für Android das OpenCV4Android.

2.1.3 Gewähltes Smartphone

Für die Anwendung des prototypischen Bildmesssystems wählte ich mein eigenes Smartphone, das Motorola Moto G (2. Generation). Es ist ein vergleichsweise preiswertes (160 Euro Kaufpreis) Smartphone mit einer 5 Megapixel Kamera und einem 5 Zoll großen Display. Zu Beginn der Entwicklung unterstützte es die Android API Level 21 (Lollipop 5.0). Mit diesen Werten ähnelt

es einem Großteil der Android-Smartphones im Umlauf, beispielsweise haben 56 Prozent Geräte im Oktober entweder die gleiche oder vorherige Systemversion [5] und viele Smartphone-Kameras haben eine Auflösung von 5-8 Megapixeln.

So kann es gut als Prototyp eines Smartphone-basierten Bildmesssystems dienen, das letztendlich von fast jedem Android-Nutzer nutzbar sein soll. Problematisch bei dem Smartphone ist nur, dass es das Hardware support level LEGACY hat, weil so viele Möglichkeiten der Camera2-API nicht verwendet werden können und in dieser Arbeit nur theoretisch betrachtet werden können.

Abbildung 2.1: Verwendetes Smartphone Moto G (2. Gen.)

2.2 Bildmesstechnik

Die Bildmesstechnik umfasst im Allgemeinen das Vermessen von Objekten anhand einer optischen Aufnahme und deren Nachbearbeitung. Dadurch befasst sich die Messtechnik einerseits mit den Fehlerpotenzialen der physischen Optik als auch mit denen der digitalen Sensorik und der digitalen Verarbeitung der Daten. Um diese weite Thematik auf eine Bachelorarbeit einzugrenzen, werden nur die wesentlichen Fehlerquellen im Zusammenhang mit mobilen Systemen

behandelt. Für das weitere Verständnis der Arbeit werden die dazugehörigen technischen und physikalischen Grundlagen in diesem Abschnitt umrissen.

2.2.1 Brennweite, Gegenstandsweite und Bildweite

Eine digitale Kamera nimmt mit einem Sensor Licht auf, der die Intensität des Lichtes in elektrischen Widerstand und schließlich in digitale Werte umwandelt. Zwischen dem eigentlichen aufgenommenen Gegenstand und dem Sensor liegt die Optik, die einfach betrachtet als eine Linse gesehen werden kann. Zum einfacheren Verständnis sind dieser Aufbau und seine wichtigsten zugehörigen Größen in Grafik 2.2 dargestellt. Wichtig ist bei folgenden Betrachtungen zu bedenken, dass es sich um ideale Bedingungen handelt und die Zusammenhänge stark vereinfacht dargestellt sind.

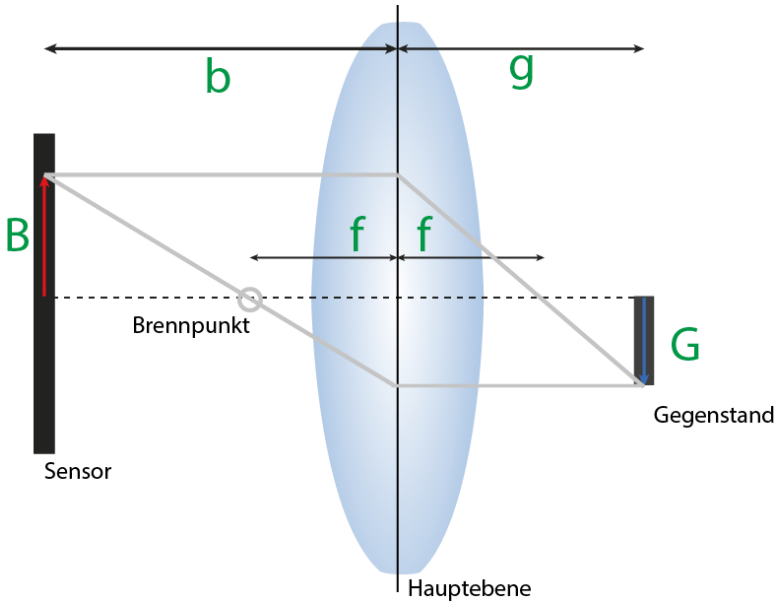

Abbildung 2.2: Kameraaufbau schematisch dargestellt (angelehnt an Darstellung aus [1])

Die Mitte der Linse wird Hauptebene genannt, der Abstand von ihr zu dem Sensor heißt Bildweite b und der Abstand zum Gegenstand wird Gegenstandsweite g genannt. Die tatsächliche Höhe des Gegenstands wird als Gegenstandsgröße G bezeichnet, während die Höhe des Gegenstands im Bild, also auf dem Sensor, die Bildgröße B ist. Die Brennweite f ist der Abstand zwischen Linse und Brennpunkt und ist eine Eigenschaft der Linse, die im Brennpunkt die einfallenden Strahlen bündelt.

Daraus lässt sich der Abbildungsmaßstab bilden, wobei die Gegenstandsgröße in Beziehung zu der Objektgröße gesetzt wird [6]:

$$A = \frac{B}{G} = \frac{b}{g}$$

Dies bedeutet, dass das Verhältnis von Bild- zu Gegenstandsgröße, also der Maßstab vom Bild zu dem reellen Gegenstand, abhängig von der Entfernung der Hauptebene zu dem Gegenstand und dem Bild ist.

Die Linsengleichung, auch Abbildungsgleichung genannt, beschreibt die Relation der Bild- und Gegenstandsweite ferner mit der Brennweite durch folgende Gleichung [6]:

$$\frac{1}{b} + \frac{1}{g} = \frac{1}{f}$$

2.2.2 Linsenverzeichnung

Linsenverzeichnungen sind Bildfehler, die durch fehlerhafte radiale Krümmungen der Linse entstehen und die reelle Position der Bildpunkte verfälschen. Es wird dabei zwischen zwei Verzeichnungsarten unterschieden, der radialen und der tangentialen Verzeichnung (vgl. [7]).

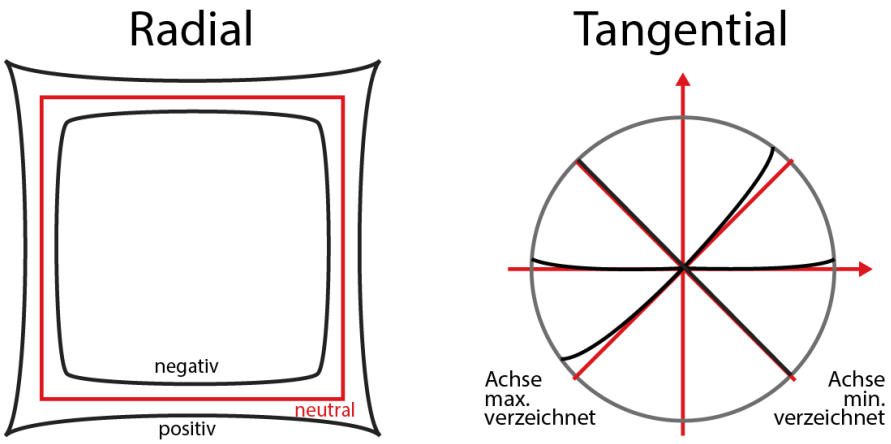

Abbildung 2.3: Schematische Darstellung der Linsenverzeichnungsarten

Radiale Verzeichnung (*radial distortion*) verschiebt die Position der Bildkoordinaten nach innen oder nach außen, wobei die Stärke der Verschiebung in der Bildmitte am schwächsten und am Bildrand am stärksten ist. Optisch wirkt das Bild dann bei nach innen verschobenen

Koordinaten, und somit negativer Verzeichnung, gewölbt während bei positiver Verzeichnung das Bild gestreckt wird (siehe Abbildung 2.3).

Die tangentiale Verzeichnung (*tangential distortion*) ist eine Verschiebung der Bildachsen, wobei auch hier die Stärke der Verschiebung zunimmt, je weiter der Bildpunkt vom Mittelpunkt entfernt ist.

2.2.3 Verkippung

Wenn die Kamera während der Aufnahme nicht im gleichen Winkel geneigt ist wie der zu vermessende Gegenstand, entstehen sogenannte Verkippungsfehler. Diese sind einerseits Perspektivenfehler und andererseits entstehen sie durch eine verfälschte Erkennung des Randes des Gegenstands.

Bei Betrachtung von mehreren Gegenständen ist außerdem zu bedenken, dass diese nun je nach Verkippung unterschiedliche Gegenstandsweiten haben und somit einen unterschiedlichen Abbildungsmaßstab besitzen.

Perspektivenfehler

Der Perspektivenfehler ist eine projizierte Abbildung, wodurch sich die scheinbare Länge des Gegenstandes ändert.

Bei flachen Objekten ist die Wirkung, dass die Objekte kleiner erscheinen - die scheinbare Längenänderung δ lässt sich aus der tatsächlichen Länge L und dem Kippwinkel α mit folgender Gleichung bilden [8]:

$$\delta = L * (1 - \cos \alpha)$$

Tiefe Gegenstände hingegen wirken durch Verkippung größer - die scheinbare Längenänderung ist [8]:

$$\delta = L * \sin \alpha$$

Ein weiteres Problem ist auch, dass die Kanten unsymmetrisch werden, wodurch beispielsweise parallele Kanten nicht mehr als solche erkannt werden.

Randproblem

Durch Verkippung kann der eigentliche Rand des Gegenstands verdeckt werden, wenn der Gegenstand beispielsweise eine abgerundete Kante hat und damit ab einem bestimmten Kippwinkel über dem Rand liegt. Dieses Problem kann verringert werden, indem manuell die richtige Kante geschätzt wird, jedoch kann es nicht vollständig korrigiert werden.

3 Mess-Szenarien

Ein jedes Messsystem hat eine gewisse Messunsicherheit *e*. Diese setzt sich meist aus mehreren Teilfehlern zusammen, welche einzeln betrachtet werden. Ein Bildmesssystem hat viele kritische Faktoren, welche die Messunsicherheit stark erhöhen können. Bei einem mobilen Bildmesssystem kommen noch weitere Teilfehlerquellen hinzu. Die ausschlaggebendsten dieser Teilprobleme werden in diesem Kapitel im Detail vorgestellt und diskutiert.

Diese Arbeit fokussiert sich zwar auf die Messung von Distanzen, aber auch viele andere Bildmessungen sind darauf aufbauend mit relativ geringem zusätzlichem Aufwand lösbar.

Messen ist das Vergleichen eines zu messenden Objektes mit einem bekannten Maßstab. In der Bildmesstechnik werden Gegenstände der realen Welt dafür zuerst anhand von Bildern elektronischer Kameras mit Hilfe der Optik auf den Sensor der Kamera projiziert. Der Sensor tastet die Projektion ab und reproduziert sie als digitales Pixelraster. So wird letztendlich das in Pixeln dargestellte Objekt mit einem bekannten Maßstab verglichen. Diese einzelnen Bereiche des Messvorgangs lassen sich in die auf Abbildung 3.1 dargestellten Stufen gruppieren.

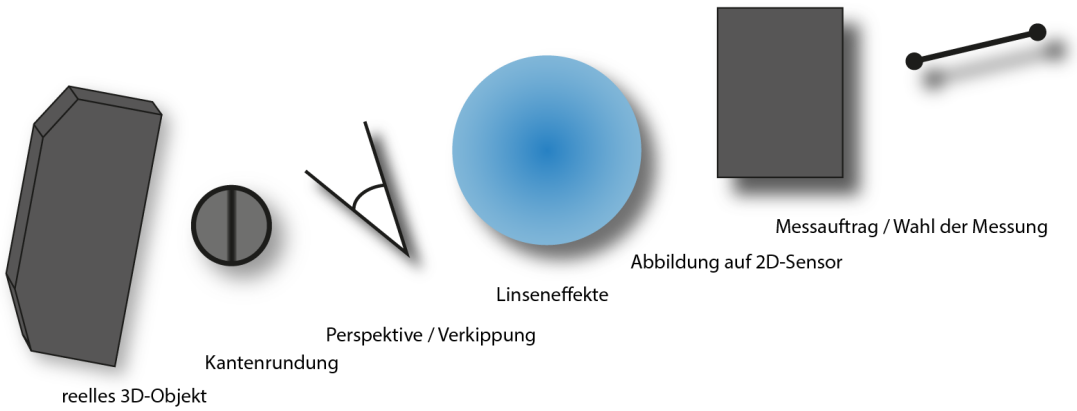

Abbildung 3.1: Stufen des Messvorgangs

Während dieses Prozesses treten systematische sowie unsystematische - und damit nicht korrigierbare - Einflüsse auf das Messergebnis auf. Selbst die systematischen Einflüsse sind nicht in jedem Fall korrigierbar, da dort Informationen fehlen welche für die Korrektur notwendig sind (beispielsweise die exakte Positionierung von Kamera und 3D-Objekt).

Der zu vermessende Gegenstand in der realen Welt hat eine dreidimensionale Form, was dazu führen kann, dass unterschiedliche Distanzen zu der Kamera betrachtet werden müssen, um einen systematischen Messfehler in beispielsweise der Distanzmessung zu vermeiden. Die zu vermessenden Kanten des Objektes sind bei einer gerundeten Form nicht mehr eindeutig definiert oder ermöglichen, falls dann die ganze Breite der Rundung gilt, nur eine sehr ungenaue Messung.

Eine perspektivische Projektion des Objektes hat die in Abschnitt 2.2.3 Verkippung erläuterten Verfälschungen des Messergebnisses zur Folge, wobei der Perspektivenfehler systematisch ist und sich somit korrigieren lässt. Das Randproblem ist dabei teilweise nicht korrigierbar wenn der Rand verdeckt wird und ist somit unsystematisch.

Zu den wichtigsten Effekte, welche Linsen auf die Projektion haben können, gehören sphärische Aberration und Astigmatismus (die allerdings die Schärfe betreffen und somit die Messunsicherheit nicht direkt verstärken) sowie die systematische Verzeichnung der Linse (siehe Kapitel 2.2.2).

Schließlich kann die Projektion auch noch durch die nichtideale Abtastung des Sensorchips verfälscht werden, wobei die einzelnen Sensorpunkte nicht in unendlich kurzer Zeit aktualisiert werden können und somit eine zufällige Anzahl an Bildpunkten verfälscht ist (vgl. [9]). Somit sind diese Abbildungsfehler nicht systematisch und nicht korrigierbar.

Im letzten Abschnitt des Messvorgangs, der Angabe des eigentlichen Messauftrags auf dem Bild, entstehen hauptsächlich systematische Bedienfehler. Dabei kann einerseits die gewünschte Messdistanz falsch gesetzt werden, was bei kleineren Anzeigen des Bildes wahrscheinlicher ist, und außerdem kann der Maßstabsfaktor ungenau angegeben werden.

3.1 Maßstabsfaktor

Der Maßstabsfaktor beschreibt die bekannte Größe, mit der in der Messung verglichen wird. In der Bildmessung ist der Bildmaßstab meistens in Pixeln angegeben, während der Weltmaßstab, also der Maßstab der realen Objekte, oft in Millimetern beschrieben wird.

Der Zusammenhang zwischen Weltmaßstab und Bildmaßstab wird entweder

- durch eine einmalige Kalibrierung der Kamera oder
- durch einen im Messbild sichtbaren Vergleichskörper hergestellt.

3.1.1 Messmarkenpapier

Eine sehr exakte Methode, den Maßstab von Bildeinheiten zu metrischen Einheiten zu erhalten, ist das Verwenden eines speziellen Messmarkenpapiers. Diese Papiere werden unter dem Messobjekt platziert und haben spezielle Markierungen, welche dem Messsystem bekannt sind und welche bestimmte metrische Abstände voneinander haben. Ein einfaches Beispiel wäre ein Papier mit Karomuster, bei dem das Messsystem die Größe der Karos kennt und die Karos automatisch erkennt. Anhand der einzelnen Abstände kann der Faktor von Pixeln zu Millimetern ausgerechnet werden. Ein solches Papier ist in Grafik 3.2 abgebildet.

Bei Papieren welche viele Messmarken haben, können sogar Koordinaten-abhängige Faktoren verwendet werden, um Messfehler zu minimieren, die etwa durch Linsenverzeichnung und Verkippung entstehen.

Eine große Einschränkung in der Verwendung von Messmarkenpapieren ist die, dass das Papier größer als das Messobjekt sein muss und das Papier extra für jede Messung mitgeführt werden muss.

3.1.2 Kalibrierpapier

Als Kalibrierpapier wird hier ein einfarbiges Blatt Papier im DIN-Format bezeichnet. Es ist sozusagen ein minimiertes Messmarkenpapier, denn es hat keine Messmarken aufgedruckt und nur seine Kanten werden für die Berechnung des Maßstabsfaktors verwendet - ihre metrischen Größen sind im DIN-Format definiert [10].

Dadurch, dass nur die Kanten den Maßstab bilden, können keine Koordinaten-abhängigen Faktoren verwendet werden und die Messfehler müssen mit anderen Mitteln kompensiert werden. Dafür sind besonders Blätter im DIN-A4 Format sehr einfach zu erhalten und müssen nicht extra gedruckt werden. Ein weiterer Vorteil im Vergleich zu Messmarkenpapieren ist, dass sich die Objekterkennung sehr viel einfacher gestaltet und sogar per Hand ausgeführt oder zumindest angedeutet werden kann.

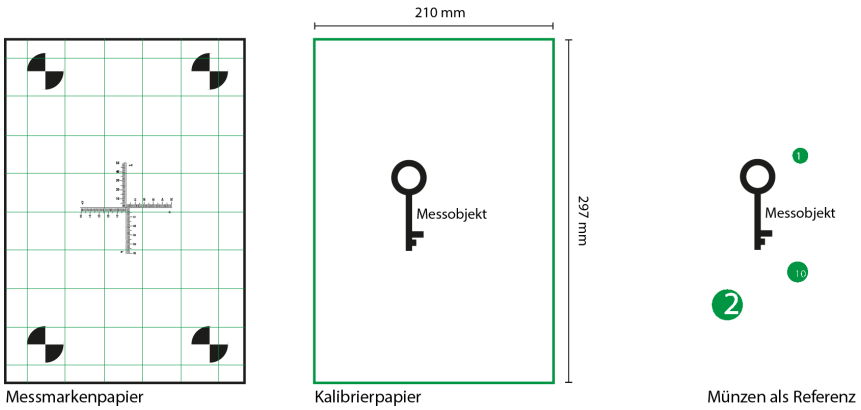

Abbildung 3.2: Referenzmethoden für den Maßstabsfaktor

3.1.3 Münze

In Bezug auf einfach erhältliche Objekte sind Münzen in unserer Gesellschaft wohl die besten Referenzobjekte, denn ein paar Münzen sind fast immer in Reichweite. Sie haben im Allgemeinen auch eine definierte, exakte Größe wie beispielsweise die Euromünzen, welche von der Europäischen Zentralbank verwaltet werden [11]. Mit Münzen bietet sich eine für den Anwender in vielen Situationen bequeme Methode, den Maßstab des Messobjektes darzustellen, indem die Münzen neben dem Messobjekt im Bild liegen (siehe Abbildung 3.2).

Das System kann die Münzen dann entweder vollautomatisch erkennen oder dem Benutzer ermöglichen, Stellen im Bild zu markieren in denen eine Münze liegt und um welche Münze es sich handelt. Daraufhin kann das System dann die erkannten Pixelmaße den metrischen Maßen der Münze zuordnen.

Ein großer Nachteil dieser Methode ist jedoch die Tatsache, dass die Münzen kleiner als viele der gewünschten Messobjekte sind. Dadurch muss der Maßstab vergrößert werden und es kommt zu einer verstärkenden Fortpflanzung des Kalibrierfehlers. Mehrere Münzen können den Messfehler dafür aber wieder verringern - durch geschickte Mittelung und Eliminierung der Extremwerte.

3.1.4 Beliebige Objekte

Wenn spezielle Objekte für die Berechnung des Maßstabs verwendet werden, hat das in der Optimierung der Messung Vorteile und in vielen Fällen kann die Messung für den Benutzer

beschleunigt und vereinfacht werden. Doch die Einschränkung auf solche speziellen Referenzobjekte ist in manchen Anwendungsfällen nicht gewünscht, weshalb ein Modus für ein allgemeines Referenzmaß für die meisten mobilen Bildmesssysteme sinnvoll ist.

Die einfachste und direkteste Methode, beliebige Referenzmaße zu definieren, ist durch Markierung von zwei Punkten A und B. Dabei muss die Distanz A -> B vom Benutzer in metrischer Einheit angegeben werden, die Distanz in Pixeln kann dann direkt mit der metrischen Distanz in Relation gesetzt werden und damit den Maßstab bilden.

Das beinhaltet jedoch mehrere Fehlerquellen, deren Größen stark von einer richtigen und exakten Benutzung abhängen. Einerseits muss die Markierung der Punkte A und B bestenfalls subpixelgenau erfolgen, was besonders auf kleinen Bildschirmen mobiler Systeme umständlich für den Anwender ist. Erfolgt die Markierung nicht genau an der richtigen Stelle, ist die Ungenauigkeit der Markierung ein zusätzlicher Messfehler, multipliziert mit der Skalierung zum Messobjekt. Außerdem muss die metrische Distanz korrekt sein, wobei eine Ungenauigkeit durch ungenaue Messinstrumente entstehen kann oder dadurch, dass die Markierung nicht im gleichen Winkel von Kante zu Kante erstellt wird wie die Messung durchgeführt wurde (vgl. Abbildung 3.3, Schritt 1).

3.1.5 Beliebige Objekte mit Objekterkennung

Eine detaillierte Objekterkennung wie etwa Flächenerkennung kann die Markierung von Referenzmaßen deutlich optimieren. Dabei wird das Bild in Flächen aufgeteilt, wobei aneinander liegende, ähnliche Pixel eine Fläche bilden. Wenn nun die Punkte A und B in der Nähe von Kanten solch einer Fläche liegen, kann die Auswahl automatisch auf die dichteste Kante der Fläche optimiert werden.

Da solch eine Verarbeitung für ein kleines Projekt zu arbeitsaufwändig sein mag, kann auch eine vereinfachte Variante ausreichende Genauigkeit bieten. Diese Methode hat sogar bei den meisten Objekten das gleiche Ergebnis und der Entwicklungs- und Performance-Aufwand ist ein deutlich geringerer. In diesem Fall wird auf der Strecke von Punkt A zu Punkt B (eventuell auch eine bestimmte Strecke darüber hinaus) nach Kanten gesucht und die Schnittpunkte anstelle der Punkte A und B verwendet (siehe Abbildung 3.3, Schritt 2).

Besonders wichtig ist bei dieser einfacheren Variante das Verhalten bei Erkennung von mehreren Kanten, wobei beispielsweise die Dichtesten oder die mit stärksten Steigungen gewählt werden können.

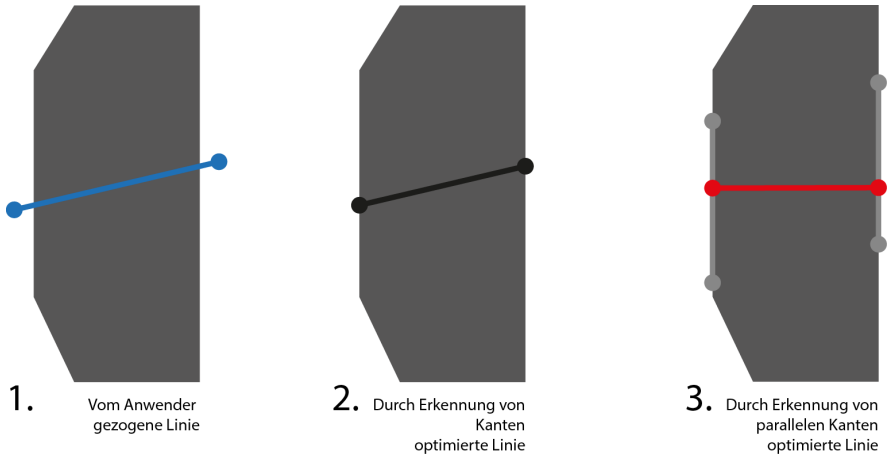

Abbildung 3.3: Optimierung eines beliebigen Objekts als Referenz

In beiden Fällen kann jedoch noch eine weitere Optimierung der Punkte A und B durchgeführt werden, indem die Kanten um die Punkte auf geometrische Eigenschaften wie Parallelität untersucht werden. Im Fall der Parallelität kann dem Benutzer angeboten werden, nicht die Distanz der (optimierten) Punkte A -> B zu verwenden, sondern die Distanz zwischen den parallelen Kanten zu verwenden (siehe Abbildung 3.3, Schritt 3). Dadurch wird in diesem Fall der Fehler vermieden, dass die Markierung der Messung nicht im rechten Winkel durchgeführt wurde während die ursprüngliche Messung für den rechten Winkel gilt.

3.2 Fokus

In der Fotografie wird die Fokussierung dazu genutzt, das Motiv eines Bildes scharf darzustellen. Hierfür wird die Hauptebene bewegt und somit die Bild- und die Gegenstandsweite angepasst (vgl. Abbildung 3.4). Für eine exakte Messung wird eine scharfe Motivdarstellung benötigt.

Der Abbildungsmaßstab (siehe 2.2.1) besagt:

$$\frac{\textbf{b}\text{ildweite}}{\textbf{g}\text{egenstandsweite}} = \frac{\textbf{B}\text{ildgröße}}{\textbf{G}\text{egenstandsgröße}}$$

Wenn sich also der Fokus verändert und somit die Gegenstands- und Bildweiten andere Werte haben, ändert sich auch die Bildgröße, da ja die Gegenstandsgröße gleich bleibt (vgl. auch Abbildung 3.4). Wird die Bildgröße im Messsystem als Maßstabsgröße verwendet, ergibt diese

Änderung folglich ein verändertes Messergebnis. Um dem entgegenzuwirken, darf entweder die Bildgröße nicht als Maßstab verwendet werden oder es muss die Auswirkung der Veränderung des Fokus bekannt sein. Wenn die Auswirkung zumindest ungefähr bekannt ist, kann sie in die Maßstabsrechnung mit eingebracht werden und der entstehende systematische Fehler berechnet und eliminiert werden.

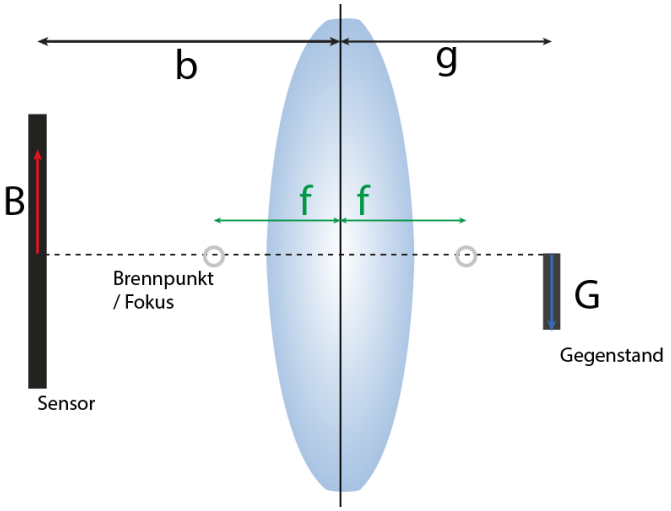

Abbildung 3.4: Schematische Darstellung der Fokussierungsgrößen

3.2.1 Einstellbar

Ein einstellbarer Fokus bietet die meisten Möglichkeiten der Fehlereliminierung. Zwei davon möchte ich kurz vorstellen: Einerseits kann man dem Anwender die Möglichkeit bieten, den Fokus auf eine beliebige Entfernung einzustellen. Diese Entfernung muss er dann für die Kalibrierung des Maßstabes verwenden, wobei er darauf achten muss, dass das Kalibrierobjekt scharfgestellt ist. Danach können mit dem gleichen Fokus Objekte vermessen werden, wobei wieder beachtet werden muss, dass diese Objekte scharfgestellt sind.

Die andere Option ist die, dass ein voreingestellter Fokus verwendet wird, welcher nicht vom Anwender geändert werden kann. Dieser Fokus muss einmal auf die gleiche Weise kalibriert werden, allerdings kann hier die Erfahrung der Entwickler in die Wahl der Fokusdistanz einbezogen werden und eine sinnvolle Distanz gewählt werden, während der Anwender eventuell oft eine unpraktische Distanz wählen würde. Des Weiteren bietet sich hier eine aufwendigere Kalibrierung eher an, da der Anwender die Kalibrierung nur einmal durchführen

muss und folglich bereit ist, mehr Zeit zu investieren. Drittens, und dies mag für manche Entwickler am Vorteilhaftesten sein, kann bei einem festen Fokus die Maßstabs-Kalibrierung mit anderen Geräten geteilt werden. So kann man die Kalibrierung zentral in einer Datenbank verwalten und somit eine Anwendung anbieten, die nur einmal pro Gerätetyp (oder pro eingebautem Kameratyp) kalibriert werden muss - viele Anwender müssen so niemals ihre Kamera kalibrieren.

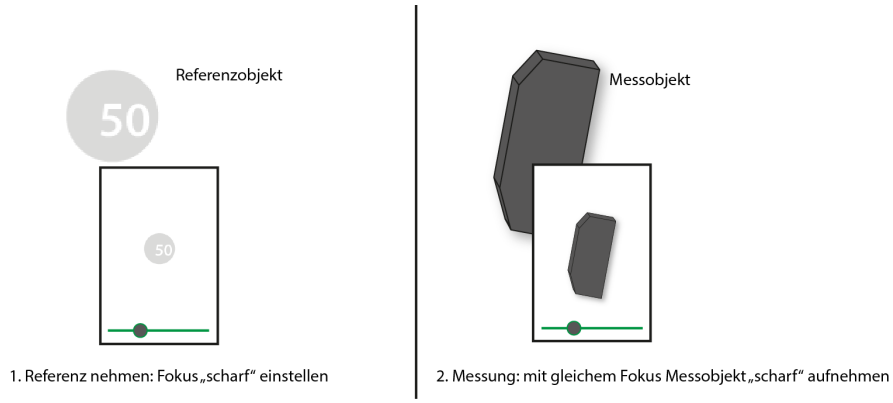

Abbildung 3.5: Messung mit einstellbarem Fokus

3.2.2 Feststellbar

Manche Kameras ermöglichen keine frei einstellbare Fokusdistanz, aber dafür kann der Fokus festgestellt, beziehungsweise der Autofokus temporär ausgeschaltet werden. Dadurch ergibt sich die Möglichkeit, dass der Nutzer den Fokus in einer Einstellung die er für sinnvoll hält feststellen kann (beispielsweise während das Messobjekt fokussiert ist). Dann kann er mit diesem Fokus eine Maßstabs-Kalibrierung durchführen und danach mit dem gleichen Fokus Messungen durchführen.

3.2.3 Bekannt

Ist die Fokussierung nicht frei wählbar aber bekannt, so ist die Auswahl der Methoden beschränkt. Ein gutes Vorgehen ist hier, eine einmalige Kalibrierung des gesamten Fokusbereichs vorzunehmen. Dafür muss nicht jede Fokuseinstellung genau kalibriert werden, sondern im Allgemeinen reichen Stichproben zwischen denen dann interpoliert werden kann.

3.2.4 Unbekannt

Ein unbekannter Fokussierungswert führt dazu, dass keine allgemeine Maßstabs-Kalibrierung allein von der Kamera vorgenommen werden kann. Das bedeutet, dass bei jeder Messung ein Referenzobjekt oder ein Referenzwert im Bild vorhanden sein muss, um den Maßstab anzugeben.

Eine Möglichkeit diese Einschränkung zu umgehen wäre das Hinzufügen einer Laser-Distanzmessung, welche die Fokussierungsdistanz ersetzt. Dafür ist in aktuell (im Jahr 2016) produzierten Smartphones jedoch zusätzliche Hardware notwendig, die im Allgemeinen ein eigenes System benötigt und beispielsweise per NFC (Near-Field-Communication) mit dem Smartphone kommuniziert.

Bei einem solchen System kommt die Fehlerquelle dazu, dass der Laser ein anderes Objekt fokussieren kann als das Smartphone. Eine synchronisierte Objekterkennung wäre also noch dazu notwendig um ein gutes Ergebnis zu erhalten. Eine Alternative zu der Objekterkennung wäre, falls die Kamera die fokussierte Bildkoordinate mitteilt, diese Bildkoordinate mit dem Laserfokus zu synchronisieren. Dafür wäre wieder eine Kalibrierung der Koordinatenräume der Kamera und des Lasers notwendig.

Im Allgemeinen kann also gesagt werden, dass es im Falle einer unbekannten Fokussierungsdistanz notwendig wird, ein Referenzobjekt in jeder Messung zu haben oder andernfalls ein sehr aufwändiges Ersatzsystem zu entwickeln.

3.2.5 Unterstützung Android

In Android ist, wie im Kapitel 2.1.1 beschrieben, die Camera2 API für den Zugriff auf die Kamera zuständig. In der API gibt es Zugriff auf den Wert *LENS_FOCUS_DISTANCE*, welcher die Fokussierungsdistanz zu dem Zeitpunkt der Aufnahme angibt. Dieser Wert kann auch explizit gesetzt werden. Die Genauigkeit des Wertes wird in drei Kategorien eingestuft, welche in den Werten *LENS_INFO_FOCUS_DISTANCE_CALIBRATION_[APPROXIMATE, CALIBRATED, UNCALIBRATED]* zu finden sind.

Die API bietet also die Möglichkeit, mit einem einstellbaren Fokus zu arbeiten, doch dafür muss die Kamera diese Funktionalität unterstützen, in der Dokumentation ist der Kommentar hinzugefügt, dass der Wert bei manchen Geräten *NULL* (also nicht verfügbar) sein kann. Aufgrund der Gerätevielfalt Androids gibt es keine Übersicht, wie viele Geräte diese Funktion

unterstützen, jedoch kann derzeit leider noch nicht davon ausgegangen werden, dass die Mehrheit der Geräte die 1 Jahr alte Camera2 API voll unterstützen.

Eine weitere Möglichkeit den Fokussierungswert zu beeinflussen, ist die, den Autofokus-Modus gezielt ein- und auszuschalten. Der zugehörige Wert in der API ist *CONTROL_AF_MODE*. Hiermit kann der Fokus zwar nicht gezielt gesetzt werden und ist auch nicht bekannt, aber er ist immerhin feststellbar, was die vorher, unter 3.2.2 Feststellbar, beschriebenen Möglichkeiten bietet. Leider ist dieser Wert nicht unbedingt verfügbar. Er ist von dem Hardware-Support-Level der Kamera abhängig.

Sinnvoll zu betrachten ist auch das Feld *CONTROL_AF_REGIONS*, welches die fokussierten Bildkoordinaten beinhaltet (ebenfalls nur bei ausreichendem Hardware-Support-Level). Mit diesen kann gut überprüft werden, ob das zu vermessende Objekt fokussiert ist und falls dem nicht so ist, kann eine Wiederholung der Messung angefordert werden beziehungsweise eine Warnung angezeigt werden, dass das Messergebnis eventuell nicht exakt ist. Die Abweichung kann auch nur minimal sein, was aber nicht automatisch festgestellt werden kann. Daher ist eine Warnung mit Anzeige der fokussierten Koordinaten möglicherweise eine hilfreiche Ergänzung des Messsystems.

3.3 Linsenverzeichnung

Linsenverzeichnung (siehe 2.2.2) ist besonders bei mobilen Geräten eine große systematische Fehlerquelle, da hier meist billige und vor allem kompakte Linsensysteme verbaut werden. Es werden viele Weitwinkel-Linsen verwendet, welche grundsätzlich zu starker negativer Verzeichnung tendieren und außerdem werden sehr kleine Sensoren eingebaut um Platz zu sparen, wodurch der Verzeichnungseffekt wiederum verstärkt wird.

Um die Verzeichnung (bis zu einem gewissen Grad) zu eliminieren, wird von der Kamera ein Verzeichnungsprofil erstellt. Es besteht meist aus einer Anzahl von Koeffizienten, die in einer Formel mit den verzeichneten Koordinaten x bzw. y die korrigierten Koordinaten $x_{korrigiert}$ und $y_{korrigiert}$ ergeben. Um diese Koeffizienten zu erhalten, wird oft eine Art Kalibrierpapier verwendet, welches in verschiedenen Positionen und Winkeln aufgenommen wird. Die aufgenommenen Koordinaten des Kalibrierpapiers werden dann jeweils mit den bekannten Koordinatenverhältnissen verglichen und so das Kameraprofil Bereich für Bereich berechnet. Die resultierenden Koeffizienten können danach für jedes Bild der Kamera verwendet werden, solange die Linse in der Kamera fixiert ist.

Da dieses Vorgehen sehr aufwendig zu entwickeln ist, kann es sich lohnen, hierfür externe Software zu verwenden. Die OpenCV-Library (siehe Kapitel 2.1.2) implementiert ein Verfahren, welches Linsenverzeichnung korrigieren kann, wobei sowohl radiale als auch tangentiale Faktoren kalibriert werden[12]. Es unterstützt 3 verschiedene Muster, die zur Kalibrierung verwendet werden können und als Ergebnis werden 5 Koeffizienten errechnet, die in Funktionen der Library verwendet werden können um Koordinaten zu korrigieren.

$$\begin{aligned} x_{rk} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{rk} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ x_{tk} &= x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_{tk} &= y + [2p_2 xy + p_1(r^2 + 2y^2)] \end{aligned} \tag{3.1}$$

$$\Rightarrow \; Distortion_{Koeffizienten} = (k_1 \; k_2 \; p_1 \; p_2 \; k_3)$$

vgl. [12] | x_{rk} : radial korrigierte x-Koordinate

Mittlerweile haben viele in Smartphones eingebaute Kameras eine Megapixel-Anzahl, die für viele Bildmessungen eine mehr als ausreichende Bildpunkt-Genauigkeit bietet. Dadurch ergibt sich eine weitere Methode, Projektionsabweichungen durch Linsenverzeichnung zu verringern. Da diese im Zentrum der Linse (und damit des Bildes) am schwächsten sind, kann der Messbereich auf das Zentrum beschränkt werden. Dadurch ist die Linsenverzeichnung im verwendeten Messbereich schwächer, allerdings ist die Auflösung geringer weshalb der Bereich nicht zu stark eingeschränkt werden darf. Außerdem müssen größere Messobjekte aus einer weiteren Entfernung aufgenommen werden.

3.4 Verkippung

Der durch Verkippung entstehende Messfehler wird hauptsächlich vom Perspektivenfehler (siehe 2.2.3 Perspektivenfehler) verursacht, gründet sich jedoch teilweise auch auf dem Problem, dass die zu messenden Ränder nicht mehr genau zu erkennen sind wenn die Kamera keine senkrechte Aufnahme macht (vgl. 2.2.3 Randproblem).

Durch Verkippung entstehende Messfehler sind untypisch für Bildmesssysteme. Im Allgemeinen werden die Systeme fest installiert und genauestens kalibriert. Es gibt Anwendungsfälle, in denen mit Verkippung gearbeitet werden muss weil die Kamera nicht rechtwinklig auf das zu

vermessende Objekt blicken kann, allerdings ist dann fast immer der Winkel der Verkippung bekannt.

Mobile Geräte werden im Normalfall weder fest installiert noch speziell kalibriert, weshalb auch ein mobiles Bildmesssystem mit einem verkippten Messobjekt umgehen können sollte, wenn es eine einfache Benutzung anbieten möchte. Moderne Smartphones haben Gyroskope eingebaut, mit denen der Neigungswinkel des Gerätes festzustellen ist, im Beispiel von Android sind diese Werte einfach zu erhalten. Das Problem ist jedoch, dass der Neigungswinkel des Messobjektes noch unbekannt ist aber für die Eliminierung des Verkippungsfehlers notwendig ist. Denn dafür sind die Bildkoordinaten um einen Winkel x zu kippen, wobei gilt:

x = Neigungswinkel des Gerätes - Neigungswinkel des Messobjektes

Eine gute Möglichkeit den Neigungswinkel des Messobjektes zu erhalten, ist eine kurze Kalibrierung dessen vor der Messung. Dafür wird das Smartphone in den gleichen Winkel gebracht, was am exaktesten mit einer geraden Oberfläche sowohl des Messobjektes als auch des Smartphones zu erreichen ist. Genau genommen muss der Winkel der Kamera senkrecht zu dem Winkel der Oberfläche des zu vermessenden Objektes sein. Da die Kameras von Smartphones senkrecht vom Gerät weg gerichtet sind, ist es also ausreichend, das Smartphone in den gleichen Winkel zu bringen wie das Messobjekt (dargestellt in Abbildung 3.6, 1.).

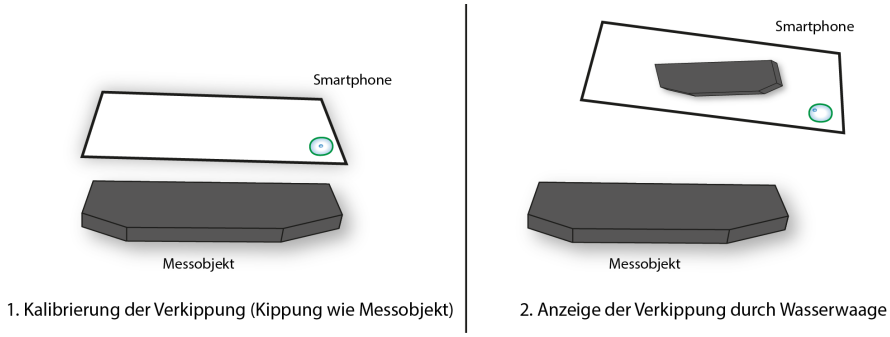

Abbildung 3.6: Messung mit Verkippungsanzeige

Ist nun dieser Referenzwinkel des Messobjektes aufgenommen, so kann der Messfehler verkleinert werden, indem während der eigentlichen Messung der momentane Winkel des Gerätes mit aufgenommen wird und auf seiner Basis das aufgenommene Bild transformiert wird. Dabei reicht es aus, die für die Messung verwendeten Bildkoordinaten zu transformieren und so den Prozess zu beschleunigen. Für die Transformation ist jedoch der Winkel nicht die einzige

notwendige Information, sondern es müssen gewisse Informationen über das Messobjekt vorhanden sein um die perspektivischen Eigenschaften zu erhalten. Bei einem flachen Objekt reichen dafür zwar vier Punkte, jedoch sind das schon doppelt so viele Punkte wie eigentlich für die Messung allein notwendig sind. Außerdem ist das Randproblem weiterhin vorhanden, was diese Methode noch aufwändiger sowohl in der Entwicklung als auch in der Bedienung macht.

Um ein genaueres Ergebnis mit dem Referenzwinkel zu erhalten kann dem Benutzer mitgeteilt werden, wenn er während des Fotografierens eine Verkippung hat. Beispielsweise die Anzeige einer Kugel, die während des Fotografierens in einem Kreis sein muss, kann dem Anwender einfach signalisieren ob die er mit Verkippung aufnimmt und wie stark die Verkippung ist (siehe Abbildung 3.6). Des Weiteren kann ein Vibrieren oder ein Audiosignal den korrekten Winkel signalisieren.

Denn wenn das Foto ohne Verkippung aufgenommen wird, ist keine Nachbearbeitung des Bildes, die wieder Folgefehler hervorruft, notwendig und auch die Ränder des Messobjektes sind auf diese Weise am besten zu erkennen.

3.5 Bedienungsaspekte

Messfehler können auch unabhängig von der Hardware oder von physischen Gesetzen entstehen, beispielsweise muss ein Messinstrument wie ein mobiles Bildmesssystem korrekt bedient werden. Dabei sind die wichtigsten Aspekte welche Fehler erzeugen die Ergonomie der Software sowie fehlerhafte Dateneingabe.

Da die Optimierung der Bedienung, oder auch der Usability, ein sehr umfangreiches Thema ist, folgen hier nur die wichtigsten Aspekte, für genauere aber trotzdem kompakte Informationen empfehle ich Literatur wie „Schnelleinstieg App Usability“von Gralak und Stark [13].

Die Ergonomie, sozusagen die Bedienbarkeit der Software für den Menschen, besteht zentral aus dem Mensch-Maschine-Interface und hat bei mobilen Anwendungen eine besonders wichtige Rolle. Dies basiert darauf, dass die Eingabe- und die Anzeigefläche einerseits klein und andererseits meist dieselbe Fläche sind, was besonders die Anzeige erschwert. Der Entwurf der Oberfläche darf daher nicht zu voll mit Informationen und Eingabemöglichkeiten sein, sollte aber die wichtigsten Funktionen beinhalten oder zumindest auf diese hinweisen. Da sich hier schwer Regeln festlegen lassen, ist es sinnvoll, besonders die Oberfläche der Anwendung von verschiedenen Benutzergruppen testen zu lassen.

Fehlerhafte Dateneingabe kann durch falsche Bedienung entstehen, wenn dem Anwender wichtige Messregeln nicht bekannt sind. Übliche Bildmesssysteme haben geschulte Anwender, mobile Bildmesssysteme werden jedoch meist bei der ersten Benutzung ohne Vorwissen geöffnet und ausprobiert. Da dieses Problem auf viele mobile Anwendungen zutrifft, ist es üblich geworden dem Benutzer bei der ersten Anwendung eine Anleitung für die Software zu zeigen. Diese besteht nur aus einer über die Oberfläche gezogenen, halbtransparenten Grafik welche weitere Informationen über die Elemente gibt oder die wichtigsten Anwendungsschritte in Stichworten auflistet.

Eine umfangreiche Dokumentation, auf die in der Anwendung hingewiesen wird, ist für ein mobiles Bildmesssystem meist trotzdem sinnvoll, da dort genauer erklärt werden kann, wie das Messergebnis optimiert werden kann.

Vom Android-System gibt es noch ein besonders nützliches Werkzeug mit dem die Anwendung erklärt werden kann, und zwar die sogenannte Snackbar. Dies ist eine Leiste, die temporär über dem Rand der Anwendung eingeblendet wird und einen Text sowie bis zu zwei Buttons beinhaltet. So kann beispielsweise der nächste Anwendungsschritt angezeigt werden und der Benutzer kann auf Drücken eines Buttons die Leiste wieder ausblenden.

4 Entwurf eines Messsystems

Dieses Kapitel behandelt den Entwurf eines Bildmesssystems nach Betrachtung und Auswahl der im vorherigen Kapitel gewonnenen Erkenntnisse.

Die Anwendung ist eine App für Android und stellt vorerst einen Prototypen und eine Grundlage für ein Bildmesssystem dar. Daher trägt sie auch einen Entwicklungsnamen: BaTh (kurz für Bachelor Thesis).

Android-Apps haben sogenannte Launcher-Icons, also ein Anwendungssymbol, auf das zum Starten der Anwendung geklickt wird. Da das Launcher-Icon ein wichtiger Indikator für die Aufgabe der App ist, wurde auch für den Prototypen ein Icon entwickelt, welches in Abbildung 4.1 dargestellt ist. Das Grunddesign ist vom Aufbau her an das Launcher-Icon der Systemkamera Androids angelehnt, weil dadurch das Symbol für viele Benutzer direkt mit der Kamera assoziiert wird. Dazu ist der Ausschnitt einer Maßstabsskala auf der Linse abgebildet, welcher die Aufgabe der Anwendung, das Messen, symbolisiert.

Abbildung 4.1: Launcher-Icon des Prototypen

4.1 Auswahl der Szenarien

Für den Entwurf des mobilen Bildmesssystems im Rahmen der Bachelorarbeit war nach Betrachten der großen Anzahl von Fehlerquellen und Möglichkeiten diese zu optimieren (vgl. Kapitel 3) klar, dass nicht alle Probleme mit diesem System optimiert werden können. Die Entwicklungszeit eines vollwertigen Systems übersteigt diesen Rahmen, weshalb eine Auswahl der für diesen Zweck wichtigsten Szenarien getroffen wurde.

Das entwickelte System bildet daher einen Prototypen, welcher die erreichbare Genauigkeit unter gegebenen Umständen umreißt und einen Ausblick auf verschiedene Optimierungsfaktoren für exakte Bildmessung mit mobilen Geräten gibt.

Besonders wichtig war mir bei der Optimierung stets darauf zu achten, dass eine gute Usability weitestgehend ermöglicht wurde. Denn ich sehe es als sehr wichtig, auf der mobilen Plattform auch die Anwendergruppe zu bedienen, welche eine App ohne umfangreiches Vorwissen nutzen will und der dabei eine mittelmäßige Genauigkeit reicht.

Der Maßstabsfaktor sollte in dem System für gute Usability besonders per Fokus direkt automatisch erkennbar sein, nachdem die Smartphone-Kamera einmalig kalibriert wurde (vgl. Kapitel 3.2.1 und 3.2.5). Diese Werte sollten dann zentral in einer Datenbank hinterlegt sein, wodurch nur wenige Anwender diese Kalibrierung durchführen müssen.

Mein Smartphone, das Moto G2, unterstützt jedoch nur das Hardware-Support-Level Legacy (das Niedrigste) und ich fand zu dem Zeitpunkt (Oktober 2015) nur wenige Android-Smartphones mit höherem Support-Level. Da mit diesem Support-Level nur ein unbekannter Fokus (vgl. Kapitel 3.2.4) möglich ist, entschied ich mich bei dem Prototypen nur mit Referenzen zu arbeiten. Dadurch ist in diesem Prototypen einerseits zu erwarten, dass ein zusätzlicher Fehler durch ungenaue Referenzwert-Erkennung entsteht und außerdem die Bedienbarkeit gravierend aufwändiger ist.

Als Referenzobjekte sind Münzen meiner Meinung nach besonders interessant, da sie fast immer in greifbarer Nähe sind. Um ihre Ungenauigkeit besonders bei größeren Objekten auszugleichen, wähle ich außerdem die Unterstützung von einfachen A4-Kalibrierpapieren. Sowohl ihre Länge als auch ihre Breite kann als Referenzwert gewählt werden, unterstützte Münzen sind die Euro-Münzen 1,2,5,10,20,50 Cent sowie 1 und 2 Euro.

Abbildung 4.2: Maßstabsmöglichkeiten

Da der Fehler durch Verkippung auch relativ gut gering gehalten werden kann, indem das Bild ungefähr parallel aufgenommen wird, ist die Unterstützung dessen per Software für den Prototypen hintenangestellt. Für mich sind andere Fehlerursachen wichtiger und vor allem

deren Implementierung interessanter zu betrachten, da die Lösung der Verkippung relativ direkt und einfach ist (siehe Kapitel 3.4).

Die Korrektur der Linsenverzeichnung lässt sich gut mit OpenCV lösen (siehe 3.3), wobei der Code nur noch an die OpenCV4Android-Library angepasst werden muss. Da es genau wie bei der Verkippung eine sinnvolle und ohne großen Aufwand zu implementierende Methode gibt, ist auch die Linsenverzeichnungskorrektur für den Prototypen niedrig priorisiert. Allerdings ist davon auszugehen, dass hier der resultierende Teilfehler deutlich größer ist und er vor allem nicht wirklich durch gute Bedienung verkleinert werden kann. Ein wenig kann er aber dadurch verkleinert werden, dass die Objekte in einem möglichst kleinen Bereich in der Mitte des Sensors sind, da dort die radiale Verzeichnung am geringsten ist.

Für die Implementierung der Verkippungs- und Linsenverzeichnungskorrektur war im Prototypen letztendlich auch keine Zeit mehr, es ist aber ein relativ geringer Aufwand, das Ergebnis noch etwas zu optimieren, sollte daher für ein vollständiges Projekt beachtet werden.

Ein mir sehr wichtiger Aspekt des Prototypen ist der Bedienungsaspekt, besonders bezogen auf die Ergonomie. Bei der Entwicklung wurden grundsätzliche Regeln wie eine überschaubare Oberfläche mit wenigen und selbsterklärenden Elementen beachtet (genauere Erläuterung siehe Kapitel 4.2.4). Da die Optimierung von Ergonomie schwer zu definieren ist, ließ ich den Prototyp außerdem während der Entwicklung oft von verschiedenen Nutzern ausprobieren, um seine Benutzbarkeit zu verbessern.

Viel direkter kann die Dateneingabe betrachtet und optimiert werden. Konkret ist das die Wahl der Referenzobjekte und der Messdistanz. Touchgesten sind auf mobilen Plattformen weitverbreitet und mittlerweile zur intuitiven Interaktion geworden, weshalb besonders die Kennzeichnung der Messdistanz sinnvollerweise als Ziehen einer Linie von Anfang bis Ende der Messung umgesetzt ist. Auch für das Markieren der Referenzobjekte gilt diese Geste, unabhängig von der Form des Objektes. So gibt es nur eine Geste zu lernen und die Bedienung ist konsistent. Der einzige Nachteil ist dabei eine etwas aufwendigere Aufarbeitung der Eingabe bei Münzen oder allgemein bei Referenzobjekten, welche nicht nur eine Distanz sondern beispielsweise einen Kreis als Referenzwert haben.

> ⇒ Die Eingabe ist also stets eine Linie zwischen den Punkten A und B. Die resultierenden, zu erkennenden Formen sind Linien und Kreise. Eine weitere Form ist außerdem sinnvoll, nämlich eine lange Linie, bei der ein großer Bereich in der Mitte der Punkte nicht betrachtet wird. Diese Form ist beispielsweise bei langen Distanzen

wie beim Kalibrierblatt sinnvoll, da das darauf liegende Messobjekt ignoriert werden soll und so außerdem Performance gespart werden kann. Bei anderen Linien ist die Betrachtung der gesamten Linie sinnvoll, da das Ziehen auf einem kleinen Touchdisplay sehr ungenau sein kann.

Die Ungenauigkeit der Geste führt dazu, dass die Punkte A und B in jedem Fall überprüft werden sollten. Alternativ wäre es denkbar, dem Benutzer eine Vergrößerungsoption zu bieten um die Punkte genau zu markieren, allerdings verlangsamt das die Messung sehr stark und da die Punkte A und B auf einer Kante liegen, ist ihre automatisch Erkennung gut per Software zu lösen. Dazu kommt, dass bei der exakten Markierung der Punkte durch den Nutzer davon auszugehen ist, dass er die Messung nicht im exakten rechten Winkel markiert und so trotzdem die Punkte A und B optimiert werden müssen.

⇒ Für die Optimierung der Punkte gibt es also zwei Probleme zu lösen: Einerseits gilt es, die gewünschte Kante des Objektes auszumachen, wobei Kantenerkennung ein oft auftretendes Problem der Bildmesstechnik ist und es deshalb eine große Auswahl an Lösungen gibt. Für den Prototypen reicht mir die einfache Gewichtung nach dem Steigungswert, wobei der höchste Wert in einem Bereich die stärkste Kante ist. Andererseits müssen die Punkte darauf optimiert werden, dass der Winkel von ihrer Linie zu dem eigentlichen Objekt so ist, wie der Benutzer ihn haben will. Bei der Messung von einem Objekt mit parallelen Kanten ist das im Allgemeinen der rechte Winkel (vgl. Abbildung 4.3). Wenn das Objekt jedoch keine parallelen Kanten hat, kann nicht automatisch bewertet werden, welchen Winkel der Anwender wählen wollte, weshalb ich mich entschied, in dem Fall den gewählten Winkel beizubehalten.

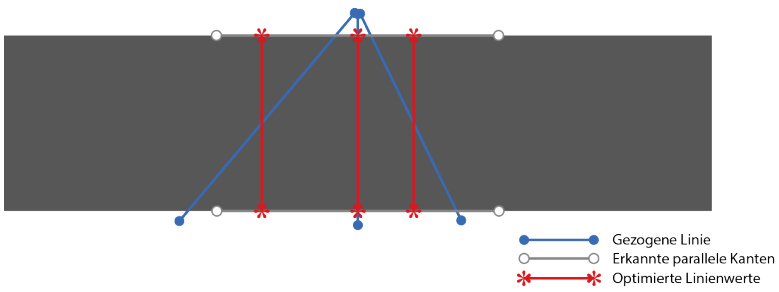

Abbildung 4.3: Korrektur von gezogenen Linien mit Winkel- und Kantenoptimierung

Die Optimierung von gezogenen Linien, die einen Kreis markieren, verwendet die gleiche Kantenerkennung, um die genauen Punkte der Kante festzulegen. Dazu kommt aber, dass zu der gezogenen Linie noch weitere Linien betrachtet werden, die dadurch erzeugt werden, dass die gezogene Linie um ihre eigene Achse gedreht wird. In bestimmten Abständen werden Linien gewählt und ihre Kanten erkannt. Die so erhaltenen Kreispunkte werden per Kreisausgleichung optimiert, wobei praktischerweise auch die Verkippung zu einem gewissen Grad eliminiert wird. Schlussendlich resultieren der Mittelpunkt und der Radius des Kreises.

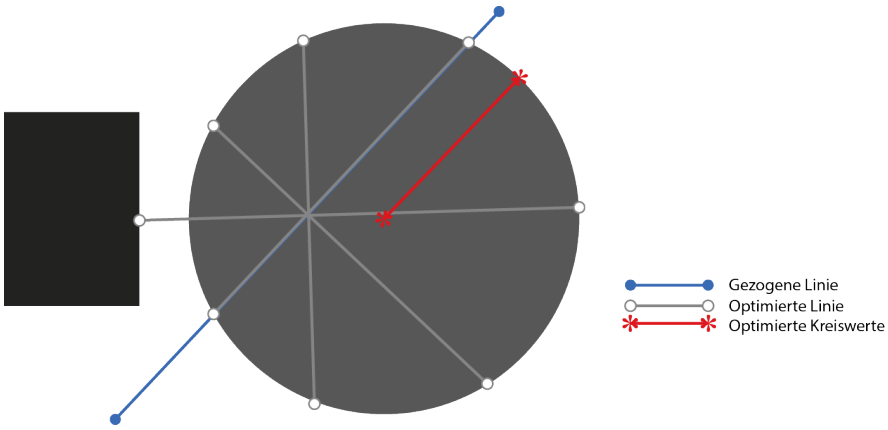

Abbildung 4.4: Korrektur von gezogenen Linien mit Kreiserkennung

Ein Nachteil der Kreisausgleichung tritt auf, wenn Objekte in der Nähe des Kreises sind und deren Kante versehentlich anstelle des Kreises gewählt wird (vgl. Abbildung 4.4, linker Rand). In diesem Fall würde die Kreisausgleichung den Kreis verfälschen, um den falschen Punkt mit zu beachten. Um das zu verhindern, wird die Kreisausgleichung mehrfach durchgeführt und nach jedem Durchlauf die Punkte entfernt, die am stärksten vom erkannten Kreis abweichen.

4.2 Realisierung

Dieser Abschnitt behandelt die Umsetzung des Bildmesssystems als Prototypen. Da die Applikation sehr umfangreich ist und einige Bereiche für diese Arbeit nicht so wichtig sind, werde ich hier besonders die wichtigen Aspekte erläutern.

4.2.1 Architektur

Die grobe Struktur einer Android Applikation ist im Allgemeinen schon recht stark durch das Framework vorgegeben. Die Entwicklung der Oberfläche wird von der Kontroll- und Datenstruktur getrennt, da die Oberfläche in XML notiert wird während die restliche Anwendung im Allgemeinen in Java geschrieben wird. Die Kontrollstruktur des Frameworks ist Event-Driven aufgebaut, das heißt sie wartet auf Eingaben wie Button-Klicks, woraufhin eine vorher zugeordnete Callback-Funktion aufgerufen wird.

Die Trennung von Oberfläche und Kontrollstruktur bietet auch bei einem kleinen Projekt wie diesem einen großen Vorteil, da die Entwicklung der grafischen Oberfläche (GUI) nach neuen Usability-Erkenntnissen mit sehr geringem technischem Aufwand angepasst werden kann.

Der große Nachteil eines Event-Driven-Systems ist der, dass die einzelnen Event-Reaktionen besonders kurz sein müssen, um das System schnell wieder in Reaktionsbereitschaft zu versetzen. In Android werden die Events und die Anzeige der GUI von einem Thread übernommen, weshalb eine lange Reaktion hier sogar dazu führt, dass die Anwendung still steht. Deshalb bot sich für diese Anwendung ein zusätzlicher Thread an, welcher im Hintergrund größere Aufgaben wie die Input-Optimierung durchführt. Da diese größeren Aufgaben jedoch auch keine lange Laufzeit haben und ihre Ausführung von mindestens zwei Klicks abhängig sind (hierzu später mehr, siehe 4.2.4), ist keine weitere Parallelität notwendig.

Activities

Die Aufteilung der Anwendung erfolgt auf oberster Ebene in drei Activities[1] . Die Erste, *Main*, ist die startende Anwendung deren Aufgabe lediglich das Starten der zweiten Activity *CameraActivity* ist. In der CameraActivity kann ein Bild für die folgende Messung aufgenommen werden oder das vorherige Bild gewählt werden. Es wird die Vorschau der Kamera angezeigt und die Einstellungen der Aufnahme verwaltet. Die dritte Activity *MeasurementActivity* ist für die eigentliche Messung zuständig. Hier können Referenzmaße auf dem Bild gekennzeichnet und Messungen markiert werden, sowie wieder einzeln entfernt werden. Falls das Bild nicht wie gewünscht aussieht, kann auch wieder zur CameraActivity gewechselt werden. Des Weiteren kann eine Kurzanleitung angezeigt werden und „Debug-Linien“eingeblendet werden, die weitere Informationen über die Input-Optimierung bieten.

[1] Activities sind sozusagen einzelne Seiten der Anwendung zwischen denen je nach Aufgabenbereich des Programms gewechselt wird.

Komponenten

Die Android-Applikation lässt sich ferner in acht Systemkomponenten unterteilen, welche in Abbildung 4.5 dargestellt werden. Die Darstellung ist nicht in allen Aspekten exakt konform zu dem genaueren Entwurf, um hier einen besseren Überblick und einfacheres Verständnis zu bieten.

Zu der CameraActivity gehört *Camera*, die Android-Komponente der Kameras des Smartphones, welche über die Interfaces der Camera2-API (siehe 2.1.1) erreichbar ist. Über Camera kann das Kamerabild erhalten werden, welches in der CameraActivity dauerhaft über die Komponente *PicturePreview* angezeigt wird. Die *CameraAL* (kurz für Camera Abstraction Layer) ist die zentrale Komponente der CameraActivity, wobei sie die Reaktion auf Eingaben der GUI übernimmt und Aktionen zuteilt indem sie beispielsweise die Camera2-Interfaces aufruft. Außerdem führt sie den Wechsel zur MeasurementActivity durch, wobei sie davor das erhaltene Kamerabild abspeichert und den Speicherort sowie Metadaten der Kameraaufnahme als Android-Extra[2] übergibt.

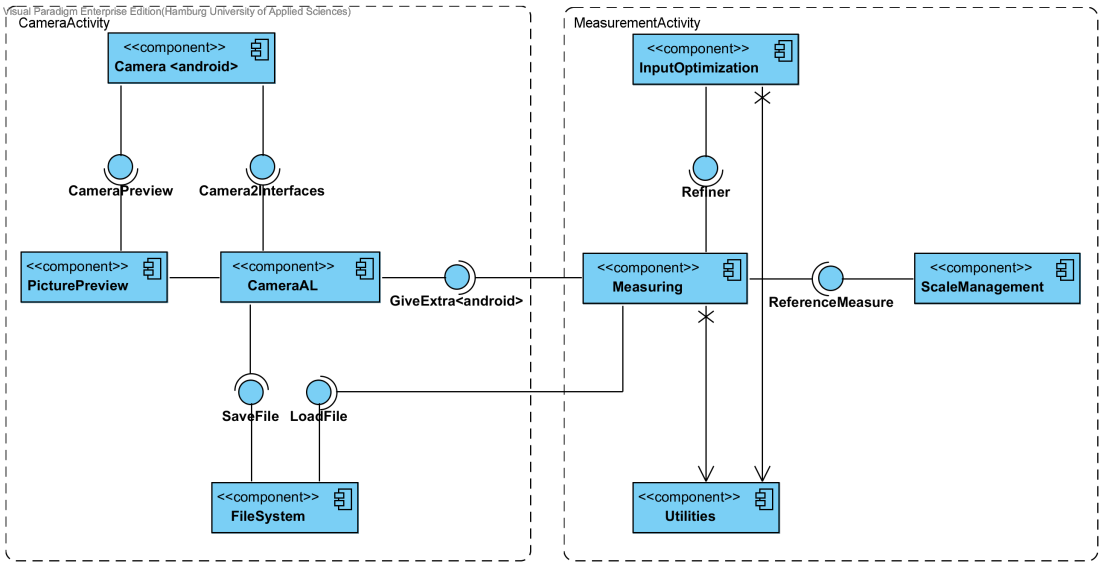

Abbildung 4.5: Überblick der Systemkomponenten

[2]Extras sind eine Möglichkeit, mit der unter anderem beim Start einer Activity zusätzliche Daten übergeben werden.

Die zentrale Komponente der MeasurementActivity ist *Measuring*. Sie ist neben der Reaktion auf die Eingaben auch für die Ausgabe zuständig, welche das Bild darstellt und darauf übergebene Messungen und Referenzen markiert sowie diese mit den zugehörigen Werten beschriftet. Eine neue Referenz sowie eine neue Messung, dargestellt durch eine Linie von Punkt A nach B, wird der *InputOptimization* übergeben, welche ein geometrisches Objekt zurückgibt, dessen Koordinaten dem Objekt auf dem Bild im Idealfall gleichen. Im Falle einer Referenz wird das erhaltene Objekt im *ScaleManagement* registriert, wobei die Größe in Millimeter des Objektes referenziert wird, welches per GUI gewählt wurde. Die Größen der Objekte werden mithilfe der *Utilities* aus einer XML-Datei geladen. Die Utilities sind außerdem für verschiedene allgemeine Berechnungen und Definitionen sowie Funktionsadapter zuständig.

4.2.2 Design

Nach dem Überblick über die Systemkomponenten weist das genauere Klassendesign der Java-Komponenten viele Klassen auf, deren Aufgaben jedoch begrenzt und klar verteilt sind. Die zur CameraActivity gehörigen Klassen sind in Abbildung 4.6 dargestellt. Zentral platziert ist die Activity-Klasse *CameraFragment* (Fragment ist eine Art Activity). Ihre on..-Methoden sind Callback-Methoden, die das Betriebssystem in einem bestimmten App-Stadium aufruft, beispielsweise wird die onClick-Methode aufgerufen, wenn ein GUI-Element der Activity geklickt wurde. Die onClick-Methode ist damit für die Zuordnung von Aktionen auf die verschiedenen Click-Events zuständig und ruft dabei beispielsweise die Methode takePicture auf, welche mit *CameraStateCaptureCallback* den Prozess startet, ein Bild aufzunehmen.

CameraStateCaptureCallback ist für diesen letzten Schritt der Camera2-API (siehe Kapitel 2.1.1) zuständig, das Aufnehmen (Capture) eines Frames. Dafür beinhaltet die Klasse eine Statemachine, die mithilfe des Zustandes der *Camera* und verschiedener Eingabesignale entscheidet, ob die Capture-Sequenz durchgeführt werden kann.

Die Camera-Klasse übernimmt die Abstraktion der CameraDevice-Klasse der Camera2-API und repräsentiert als solche nach ihrer Initialisierung durch das CameraFragment die Hauptkamera des Gerätes. Während der Initialisierung wird die Klasse *CameraConfiguredCallback* eingesetzt, die mit ihren Callback-Funktionen vom Betriebssystem die Information bekommt, ob die Konfiguration der Kamera erfolgreich war und falls nicht, welche Einstellungen nicht übernommen werden konnten.

AutoAdjustTextureView schließlich ist für die laufende Anzeige des Vorschaubildes zuständig, wobei die Klasse die Auflösung des Bildes an die Größe des zugeordneten GUI-Elements anpasst.

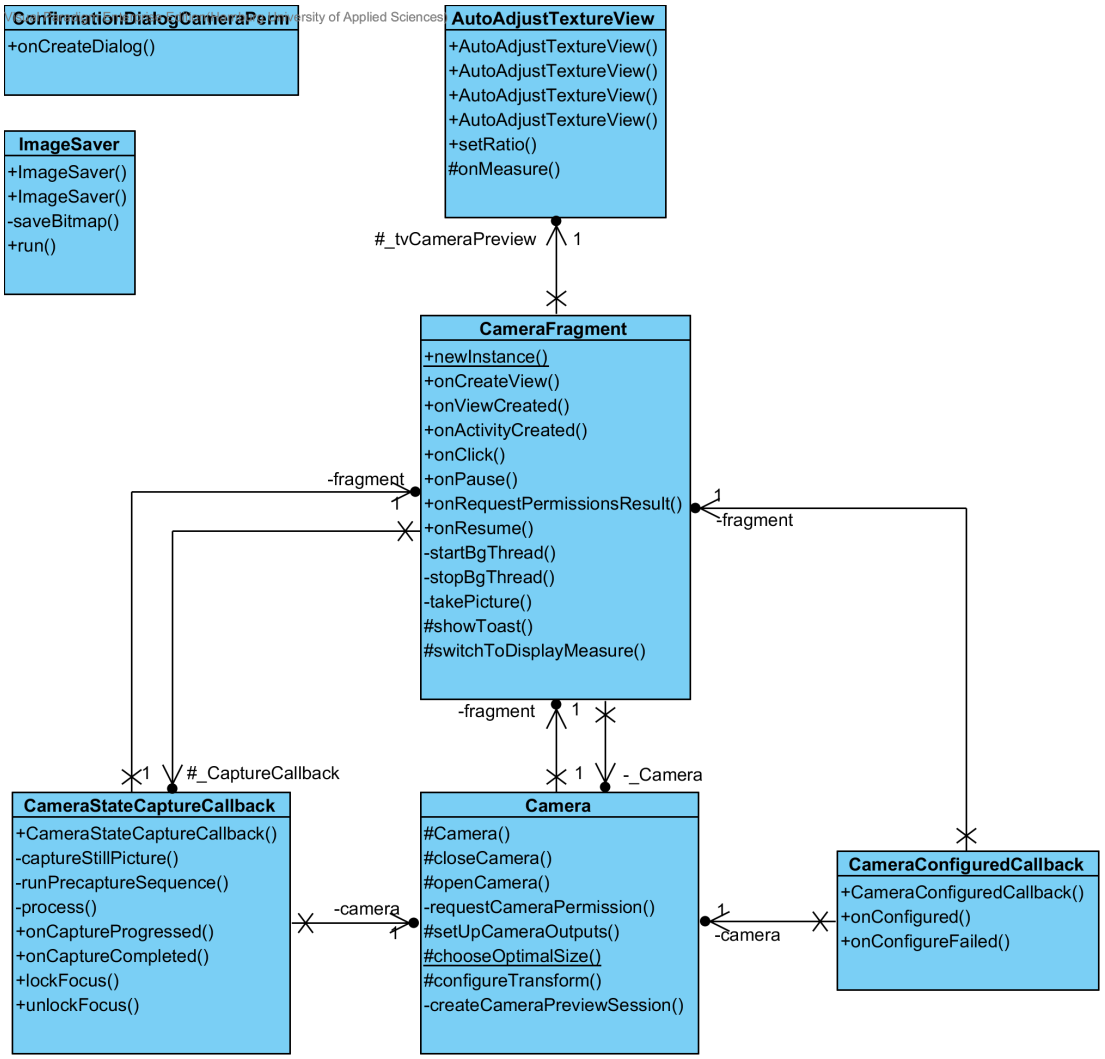

Abbildung 4.6: Klassendesign der Kamera-Abstraktion

ConfirmationDialogCameraPermission ist ein Popup-Dialog, welcher vom CameraFragment geöffnet wird, wenn die App keine Erlaubnis hat, die Kamera zu nutzen und die Klasse *ImageSaver* speichert das aufgenommene Bild im Dateisystem ab.

Die anderen Java-Klassen der Applikation, welche den Hauptteil des Prototypen ergeben, sind in Abbildung 4.7 dargestellt. Das dort abgebildete Package *cameraAL* umfasst die eben erklärten Klassen von Abbildung 4.6. Es wird von der *Main*-Klasse ausgeführt, wofür die Activity ein

GUI-Element über die ganze Oberfläche besitzt und das CameraFragment diesem zuordnet und so startet.

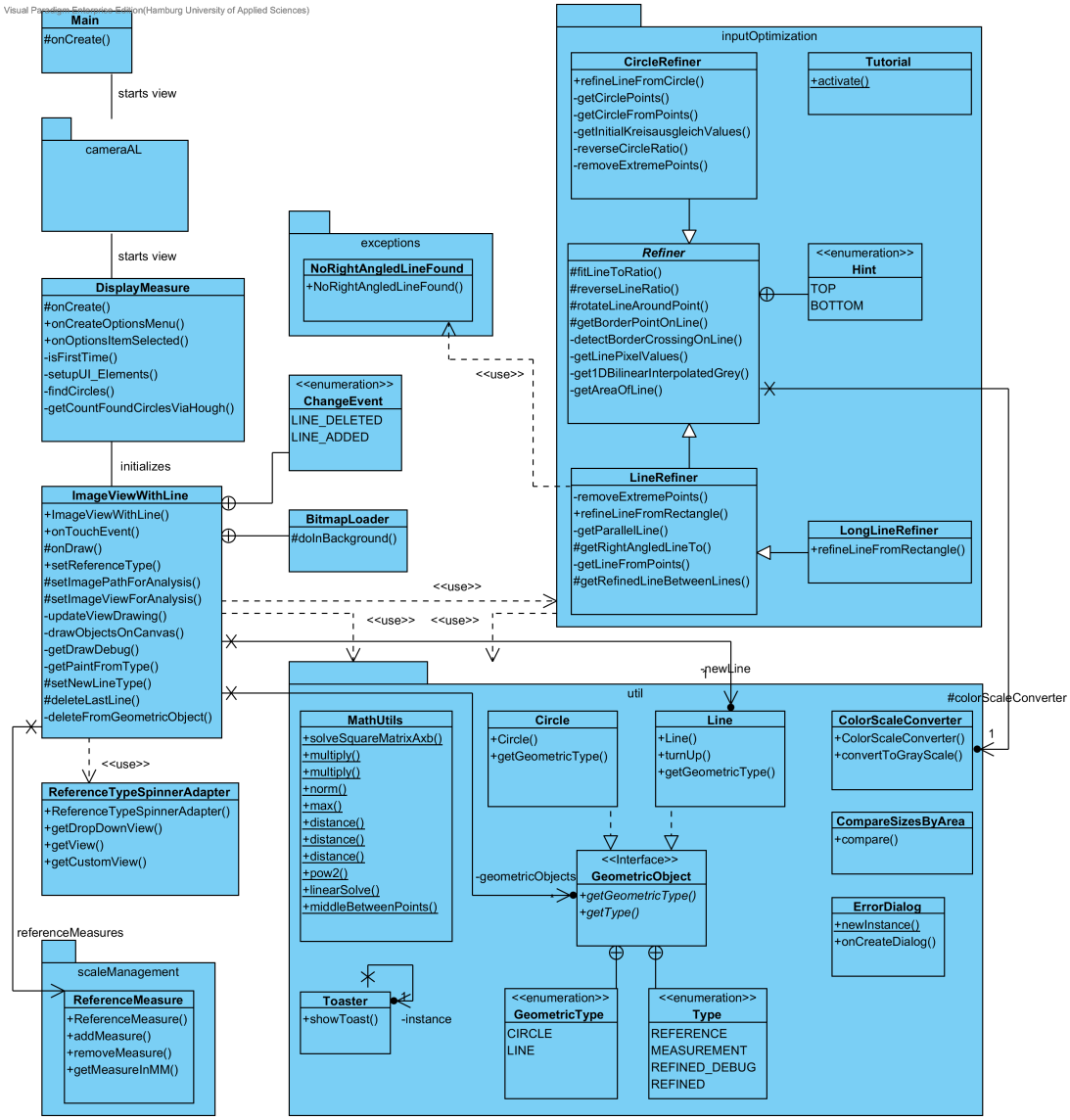

Abbildung 4.7: Gesamtes Java-Klassendesign der Android-Applikation

Das CameraFragment wiederum wechselt nach Auswahl oder Aufnahme des zu vermessenden Bildes zu der *DisplayMeasure*-Activity. Sie lädt das gewählte Bild von dem Dateisystem (vom per Extra übergebenen Pfad) und initialisiert seine GUI-Komponenten und die Klasse

ImageViewWithLine. ImageViewWithLine repräsentiert dabei die Haupt-Interaktionsfläche des Messsystems, hinter der das geladene Bild angezeigt wird und auf dem die Referenzen und Messungen markiert werden können. Außerdem zeigt es die wesentliche Ausgabe des Messsystems an.

Der wesentliche Controller der MeasurementActivity ist auch hier die Activity selbst, ImageViewWithLine. Sie reagiert auf die restlichen Eingaben der Buttons über einzelne ActionListener und auf Eingabe des Auswahldialogs *ReferenceTypeSpinnerAdapter*, mit dem der Referenztyp gewählt wird. Des Weiteren aktiviert ImageViewWithLine beim ersten Aufruf der Anwendung das *Tutorial*, um durch die Erklärung der Anwendung gleich die erste Messung zu verbessern, indem Fehler durch Bedienungsaspekte verringert werden. Das von Android empfohlene Einstellungsmenü wird ebenfalls von der Activity verwaltet, hier reagieren direkte Callback-Methoden.

Die Inputoptimierung wird durch die abstrakte Klasse *Refiner* repräsentiert. Sie beinhaltet allgemeine Funktionalität wie die Koordinatentranslation vom Eingabekoordinatenraum zum Bildkoordinatenraum sowie die eigentliche Kantenerkennung auf einer Linie. Dies sind Funktionen die unabhängig von der geometrischen Form benötigt werden, weshalb sie in dieser Oberklasse vereint sind. Spezielle Refiner wie der *CircleRefiner* für Kreisformen und der *LineRefiner* für Linien haben ihre eigenen Funktionen und Implementationen der restlichen Optimierung der Form, beispielsweise in der Unterscheidung zwischen Kreisausgleich und Geradenausgleich oder dem Erhalten von Kreispunkten im Vergleich zu rechtwinkligen Linien (vgl. Seite 28). Die spezielle Linienoptimierung für lange Linien wie Kanten von A4-Blättern ist implementiert in der von LineRefiner erbenden Klasse *LongLineRefiner*, weil sie eine eigene Implementierung der Optimierungsmethode benötigt, welche nicht die gesamte Linie betrachtet sondern nur die Bereiche um Anfang und Ende der Linie.

4.2.3 Implementierung

Da die Implementierung der Java-Klassen sehr umfangreich ist und in den meisten Bereichen eine direkte Umsetzung des Designs ist, werde ich in diesem Kapitel nur auf einzelne Besonderheiten und komplexe Aufgaben eingehen, die ich in dieser Arbeit besonders betrachtenswert halte.

Touch-Interaktion

Die Klasse ImageViewWithLine, die von der Android-Klasse View abstammt, hat eine Oberfläche für welche die Touch-Interaktion (TouchEvent) abgefangen werden kann, die Callback-Methode onTouchEvent wird dabei vom Betriebssystem aufgerufen.

Das übergebene MotionEvent beinhaltet dabei die Metadaten der Aktion. Einerseits enthält es einen Action-Code, wobei in diesem Fall drei wichtig sind: ACTION_DOWN als das startende Event der Geste, ACTION_MOVE während noch gedrückt wird aber die Position sich verändert hat und ACTION_UP für das Beenden der Geste. Neben dem Action-Code beinhalten MotionEvents die Koordinaten der View-Oberfläche, wo das Event registriert wurde. Das sind dann bei diesen drei Action-Codes im ersten Fall die Koordinaten des Anfangspunktes der Linie und im letzten Fall die des Endpunktes der Linie. Die dazwischen liegenden ACTION_MOVE-Events werden nicht verwertet, falls der Benutzer während der Ausführung der Geste zu einem neuen Endpunkt entscheidet.

Diese Methode ist im Code-Listing 4.1 abgebildet, wobei noch zu beachten ist, dass bei Registrierung des Anfangspunktes überprüft wird, ob vorher festgelegt wurde, ob die Geste eine Messung oder eine Referenzmarkierung angibt. Außerdem wird mit dem Anfangspunkt zuerst eine temporäre Linienvariable erstellt und erst die fertige Linie nach der Action ACTION_UP der Sammlung an GeometricObjects hinzugefügt, welche die zentrale Verwaltung der Referenzen und Messungen darstellt.

Der Aufruf der Funktion updateViewDrawing in Zeile 20 verursacht, dass die hinzugefügte Linie hinsichtlich ihres Typs optimiert wird (jeweiliger Line- oder CircleRefiner) und danach erst die Anzeige aktualisiert wird.

```
1  @Override
2  public boolean onTouchEvent(MotionEvent motionEvent) {
3    switch (motionEvent.getAction()) {
4      case MotionEvent.ACTION_DOWN: // started gesture
5        if (newLineType != null) { // newLineType needs to be set to define
6                                   //   whether the line is a reference or a measure
7          // -> save the point as the starting point in newLine
8          newLine = new Line(new PointF(motionEvent.getX(), motionEvent.getY()), null, newLineType);
9          Log.d(_TAG, "Line created");
10         return true;
11       }
12     case MotionEvent.ACTION_MOVE: // still doing gesture
13       return true;
14     case MotionEvent.ACTION_UP: // ended gesture
15       if (newLine != null) {
16         // -> save the point as the end point, together with newLines starting point
17         Line newLine_ = new Line(newLine.getA()
18                           , new PointF(motionEvent.getX(), motionEvent.getY()), newLine.getType());
19         geometricObjects.add(newLine_); // the new line is being saved to the set of
20                                         //   geometric objects displayed on the interface
21         Log.d(_TAG, "Line added");
```

```
22            newLine = null;
23            newLineType = null;
24            updateViewDrawing(ChangeEvent.LINE_ADDED); // update the view, but handle event LINE_ADDED first
25            return true;
26          }
27      }
28      return true;
29  }
```

Listing 4.1: Touch-Interaktions-Methode der Messoberfläche ImageViewWithLine

Referenzwerte

Die erhaltenen Referenzdistanzen werden in der Klasse ReferenceMeasure gesichert. Dabei wird einerseits die Distanz in Pixeln und in Millimetern angegeben, außerdem wird aber auch eine Gewichtung der Referenz mit angegeben. Mit dieser Gewichtung gibt es die Möglichkeit, beispielsweise Distanzen von parallelen Kanten stärker zu gewichten als Distanzen, bei denen der Distanzwinkel des Nutzers übernommen wurde weil keine parallelen Kanten erkannt werden konnten. Ferne kann eine Gewichtung von Kreisen wertvoller sein als eine von Geraden, da Kreise einerseits auf mehr Messpunkten basieren und andererseits durch die Kreisausgleichung einen verringerten Verkippungsfehler besitzen (siehe Seite 28).

Die so gewichteten Messungen werden in LinkedHashMaps hinterlegt, wobei der Schlüssel der Pixelwert ist und der Wert im einen Fall die Millimeter-Anzahl und im andern Fall die Gewichtung ist. Wenn jetzt eine Messanfrage kommt, also eine Pixeldistanz in Millimetern erfragt wird, kann einfach über beide Maps iteriert werden und mit in Listing 4.2 abgebildeter Methode berechnet werden. Da den Maps nur Werte hinzugefügt und entfernt werden müssen sowie über die kompletten Maps iteriert wird, bieten sich hier LinkedHashMaps an, weil das Einfügen und Entfernen von Werten keine weitere Aktion erfordert und das Iterieren über die Werte keine vorherige Betrachtung der ganzen Map erfordert.

```
1   public double getMeasureInMM(double valueInPX) {
2     double calculated, weight;
3     double sum = 0, count = 0;
4     // iterate over all reference measures
5     for (Double d : measures.keySet()) {
6       calculated = (valueInPX * measures.get(d)) / d; // calculate the mm value for this reference measure
7       weight = weights.get(d);
8
9       // save the weighted single mm values and their count
10      sum += calculated * weight;
11      count += weight;
12    }
13    return (count > 0) ? sum / count : -1;           // if there are no references,
14  }                                                  //   return -1 since it is no valid distance
```

Listing 4.2: Berechnung der Pixeldistanz in Millimetern

Input-Optimierung

Da der Vorgang der Optimierung der eingegebenen Linien einer der wichtigsten Prozesse der Applikation ist, erläutere ich ihn hier grob anhand der Optimierung zu einer Linie, also der Klasse LineRefiner.

Die Linie wird in der Methode refineLineFromRectangle übergeben, zusammen mit dem zu vermessenden Bild als Bitmap sowie dem Größenverhältnis der GUI-Koordinaten der Linie zu den Koordinaten des Bildes. Da es sich um unterschiedliche Koordinatenräume handelt, wird die Linie zu Anfang direkt zu Bildkoordinaten konvertiert und die resultierenden Objekte werden am Ende in GUI-Koordinaten zurück konvertiert. So ist eine größere Genauigkeit in den Koordinatenwerten während der Berechnung möglich, da der Bildkoordinatenraum bei den meisten Geräten der Größere ist.

Als zweiter Schritt wird die Linie darauf überprüft, ob Punkt A auf der Y-Achse vor Punkt B liegt und falls dies nicht gilt, werden die Punkte ausgetauscht. Dies hilft später bei der Umsetzung von manchen Algorithmen, weil so die Ordnung auf der Y-Achse immer gleich ist. Daraufhin wird die Linie in der Mitte geteilt, um jede Seite einzeln und auf gleiche Weise zu bearbeiten.

Zu jeder Hälfte wird nun ein Kantenpunkt gesucht. Dieser soll aber nicht der Schnittpunkt mit der Eingabe-Linie sein, sondern mit dem anderen Punkt eine rechtwinklige Gerade über das Objekt bilden (siehe vorherige Erläuterung, Seite 28). Deshalb wird zuvor zu jeder Hälfte eine Kante gesucht, welche die Eingabe-Linie schneidet. Dabei wird ein übergebener Hint, also Hinweis, verwendet der auf das jeweilige äußere Ende der Eingabe-Linie weist (für die obere Hälfte also TOP und für die untere BOTTOM). So kann auch hier die gleiche Methode verwendet, aber trotzdem optimiert gesucht werden.

Schließlich wird mit Hilfe der Kanten und der Eingabe-Linie die optimierte Linie ermittelt. Dafür wird die Gerade berechnet, die rechtwinklig zwischen den Kanten steht und dabei den Mittelpunkt der Eingabe-Linie zwischen den Kanten schneidet. Diese Gerade wird als REFINED markiert, wodurch sie in der Ausgabe anders behandelt werden kann als beispielsweise Eingabe-Linien.

Neben dieser optimierten Gerade werden außerdem die beiden Kanten als DEBUG-Linien zurückgegeben, damit der Anwender im Erweiterten Anzeigemodus (Debug-Modus) teilweise nachvollziehen kann, welche Optimierung durchgeführt wurde.

4.2.4 Interface

Das Interface der Anwendung ist übersichtlich und schlicht gehalten, um eine möglichst gute Bedienbarkeit zu ermöglichen (vgl. Kapitel 3.5). Beide Activities haben eine sehr große Anzeigefläche mit schmalen Menüleisten. Die Höhe dieser Menüleisten ist die von Android empfohlene Höhe, die je nach Bildschirmform, Ausrichtung und Auflösung variiert. So wirken die Bedienungselemente bekannt und passen in die Gesamtoptik von Android. Außerdem wird ihre Größe immer so angepasst, dass sie groß genug für Interaktionselemente ist, wodurch dieses Problem der Ergonomie für die Buttons gelöst ist.

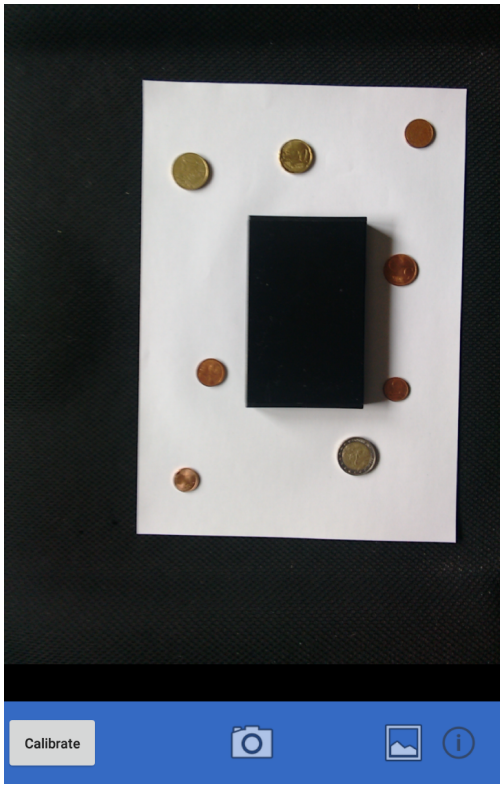

Abbildung 4.8: GUI der CameraActivitiy

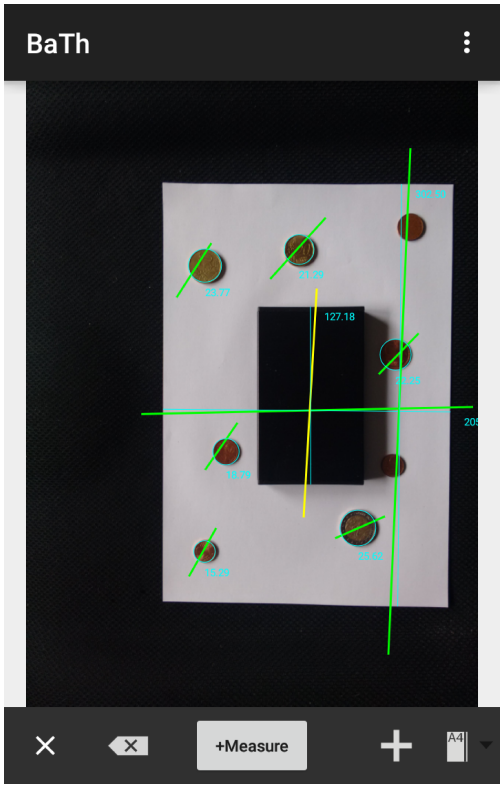

Abbildung 4.9: GUI der MeasureActivitiy

Für die Optik der einzelnen Buttons habe ich an den möglichen Stellen auch die Optik des Betriebssystems verwendet, damit diese auch einen Wiedererkennungswert haben und nicht jedes Element dem Benutzer erklärt werden muss sondern ihm dessen grundsätzliche Aufgabe intuitiv klar ist. So sind die verwendeten Icons der CameraActivity vollständig dem Betriebssys-

tem entnommen, bei der MeasureActivity sind es die beiden linken Icons. Nur Buttons, deren Funktion nicht an einem Bild einfach erklärbar ist, sind mit einem Schriftzug versehen.

Dies hat nun einen kleinen Nachteil wenn die Symbole nicht bekannt sind, aber bietet auch mehr nutzbare Fläche. Und möglichst viel freie Fläche ist nicht nur für mich als Designer vorteilhaft, sondern bietet auch den Vorteil, dass die Reaktionsfläche der Buttons bis weit über das Bild hinausragt, was wieder die Ergonomie der Anwendung verbessert.

Kamera	Aufnehmen eines Bildes, weiter zur MeasurementActivity *CameraFragment.takePicture*
Bild	Wählen des letzten Bildes, weiter zur MeasurementActivity *CameraFragment.switchToDisplayMeasure*
Info	Informationen zur Optimierung der Messungsaufnahme *Popup-Dialog*
Calibrate	Kalibrierung der Linsenverzeichnung *aufgrund fehlender Entwicklungszeit noch ohne Funktion*
Abbrechen	Abbruch der Messung, zurück zur CameraActivity *DisplayMeasure.finish*
Backspace	Löschen der letzten Eingabe und zugehöriger Ausgaben *ImageViewWithLine.deleteLastLine*
+Measure	Messung hinzufügen/markieren *ImageViewWithLine.setNewLineType*
Plus	Referenz hinzufügen/markieren *ImageViewWithLine.setNewLineType*
A4-Blatt	Referenztyp wechseln *ImageViewWithLine.setReferenceType*
Drei Punkte	Settings- bzw. Optionsmenü *DisplayMeasure.onOptionsItemSelected*

Tabelle 4.1: Zuordnung Buttons -> Funktionen

Die Buttons der Activities sind den Funktionen der Anwendung wie in Tabelle 4.1 dargestellt zugeordnet.

Die Hauptfläche der MeasureActivity reagiert auf Touch-Eingaben nach der Wahl von dem +Measure- oder dem Plus-Button, welche festlegen ob es sich bei der Eingabe um eine Referenzdistanz oder eine Messdistanz handelt. Diese werden nach der Eingabe grafisch auf der Fläche dargestellt, wobei die Referenzen Hellgrün und die Messungen Hochgelb dargestellt

sind. Die Beschriftungen sind in Cyan gefärbt. Diese drei Farbtöne gehören zu den Neonfarben und haben eine hohe Leuchtkraft mit hoher Farbsättigung. So sind die Linien auf fast jedem Hintergrund auch dünn gut sichtbar und müssen nicht viel Platz des Bildes in Anspruch nehmen.

Um die Oberfläche der MeasureActivity dem Anwender bei der ersten Benutzung der App zu erklären, wird eine, wie in Kapitel 3.5 bereits erläuterte, halbtransparente Grafik über der Oberfläche eingeblendet. Sie zeigt, wie in Abbildung 4.10 zu sehen, die Erklärungen für die GUI-Elemente und stellt in Kurzform den Ablauf einer Messung vor.

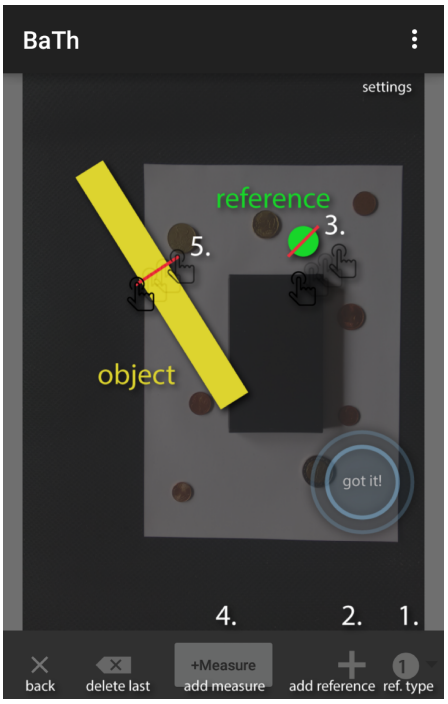

Abbildung 4.10: Overlay Tutorial der MeasureActivity

Zu der normalen Ausgabe der MeasureActivity gibt es auch die Möglichkeit, Debug-Informationen über die Settings anzeigen zu lassen. In Abbildung 4.11 ist dieser Debug-Modus aktiviert, bei der gleichen Messung wie in Abbildung 4.9 dargestellt.

Die Debug-Informationen bestehen aus cyan-farbenen Linien, die wichtige Entscheidungen des Optimierungsalgorithmus anzeigen. Bei der Messungslinie (gelb) beispielsweise sind die erkannten Kanten eingezeichnet, die zur optimierten, rechtwinklig dazwischen liegenden Messungslinie geführt haben. Bei der Referenzmünze sind die in der ersten Iteration erkannten

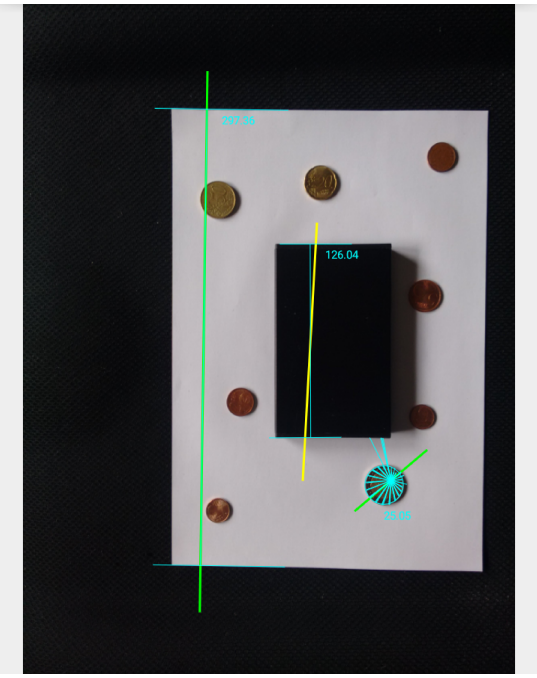

Abbildung 4.11: Debug-Modus der Messung

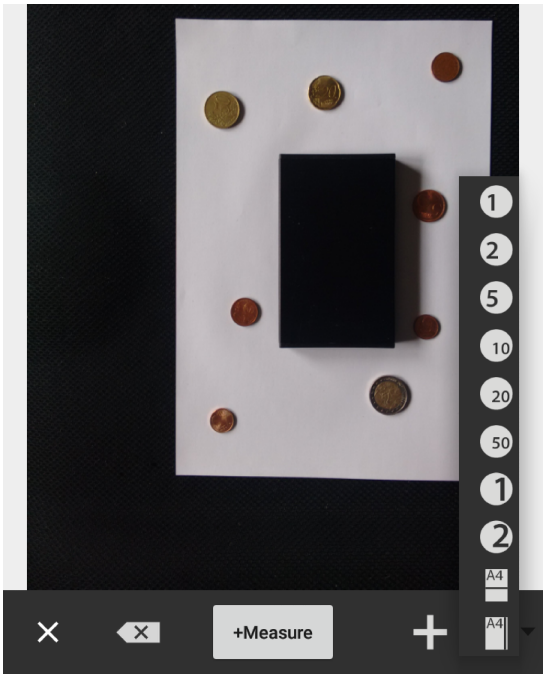

Abbildung 4.12: Auswahl des Referenztyps

Kantenpunkte des Kreises dargestellt. Besonders diese Kreispunkte geben viele Informationen, falls ein Kreis falsch erkannt wurde. Dann kann man an ihnen sehen, ob nahe liegende Objekte oder Schatten die Ursache sind und diese Fehlerursachen in einer neuen Messung vermeiden.

Abbildung 4.12 zeigt die Auswahl des Referenztyps. Diese Auswahl basiert auf einem Spinner-Objekt, welches nach dem Anklicken des rechten Buttons eingeblendet wird. Die Euromünzen und Kalibrierblätter sind in abstrakter Form dargestellt, wodurch sie zu den anderen Bildbuttons passen und auch in kleiner Anzeige leicht zu erkennen sind. Die Bilder sind, wie auch Schriftzüge und andere Bilder der Anwendung, per XML verknüpft, wodurch sie ausgetauscht werden können ohne dass der Programmcode geändert werden muss. Dies ist ein übliches und empfohlenes Verhalten bei der Verwendung von externen Ressourcen in Android-Apps, was daher auch vom Android Studio unterstützt wird, indem eine Vorschau der Ressource auch bei der Entwicklung im Programmcode angezeigt wird.

4.3 Messergebnisse

Das Messergebnis eines Messwerkzeugs ist nicht der wahre Wert der Messgröße, sondern hat stets als Näherungswert des wahren Wertes eine zusätzliche Angabe, die Messunsicherheit. Die Analyse der Unsicherheit des entwickelten Bildmesssystems ist daher notwendig, um die Qualität des Systems zu bewerten. Da es sich jedoch um einen Prototypen eines Smartphone-basierten Bildmesssystems handelt welcher viel Optimierungspotenzial bietet, kann diese Unsicherheit nicht als Machbarkeitsresultat der These dieser Arbeit gelten. Sie ist vielmehr ein Hinweis, ob sich die weitere Entwicklung und Vermarktung eines solchen Systems lohnt.

Deshalb wurde keine detaillierte Analyse der Teilsysteme vorgenommen, sondern die Unsicherheit des Gesamtsystems in einigen gewählten Testfällen betrachtet.

Im vorherigen Kapitel 4.2.4 Interface ist einer dieser Fälle grafisch zu betrachten in Grafik 4.9. Hier wird ein Festplattengehäuse vermessen (schwarzes Rechteck in der Mitte) und als Referenzmaße gelten die Längs- und Querseite eines A4-Blattes sowie sechs Euro-Münzen von verschiedenen Werttypen.

Die tatsächliche Länge des Messobjektes beträgt 122,1 mm - die Anwendung misst 127,18 mm. Das ergibt eine absolute Messabweichung F von 5,08 mm beziehungsweise eine relative Messabweichung f von 4,16%.

Zu beachten ist dabei auch, dass in der Aufnahme weitere zwei Münzen als mögliche Referenzwerte hätten gewählt werden können. Diese wurden allerdings vom System zu ungenau erkannt (aufgrund nahe liegender Schattenkanten) und deshalb nicht mit in die Messung einbezogen um ein besseres Messergebnis zu erhalten. Solch eine differenzierte Betrachtung des Messergebnisses ist jedoch nicht von jedem Anwender zu erwarten (sowie weitere Optimierungen, siehe 3.5 Bedienungsaspekte), weshalb eine genauere Analyse der Messunsicherheit auch unerfahrene Anwender als Tester beinhalten muss.

In Tabelle 4.2 sind weitere Messungen aufgelistet, wobei zu jedem Gegenstand die reale Länge $rLen$ des Gegenstandes (gemessen mit Messschieber bzw. ab 150mm mit Zollstock) und die mit dem Prototyp gemessene Länge $mLen$ angegeben sind. Außerdem sind die absolute Messabweichung sowie die relative Messabweichung dazu berechnet. Dazu sind wichtigste Informationen zu der Messung angegeben, wie beispielsweise welche Objekte als Referenz dienten.

Die getätigten Messungen sind verschiedene mögliche Anwendungsfälle, wobei unterschiedliche Belichtungen und Entfernungen bei ähnlichen und teils sogar gleichen Objekten gewählt

ID	Gegenstand	rLen (mm)	mLen (mm)	F (mm)	f (%)	Referenzobjekte,..
1	Nagel	1,3	1,95	0,65	50,00	20C,10C,5C,2C
2	Haken	3,9	4,97	1,07	27,44	20C,10C,5C,2C
3	Haken	3,9	3,91	0,01	0,26	20C,10C
4	Haken	3,9	3,86	0,04	1,03	20C,10C,5C,2C
5	Dübel	9,8	10,13	0,33	3,37	10C
6	Dübel	9,8	10,29	0,49	5,00	20C,10C,5C,2C
7	HAW-Schlüsselanh.	10,6	10,71	0,11	1,04	20C,10C,5C,2C
8	USB-Stick	12,2	13,90	1,70	13,93	20C,10C,5C,2C
9	Holz-Untersetzer	88,7	88,50	0,20	0,23	10C,5C,2C,1C
10	Mousepad	179	180,74	1,74	0,97	A4-Länge,20C,10C,5C,2C
11	A4-Länge	297	297,39	0,39	0,13	1E,20C,10C,5C,2C
12	Türbreite	812	811,20	0,80	0,10	A4-Breite
13	Türbreite	812	811,75	0,25	0,03	A4-Breite

Tabelle 4.2: Messergebnisse

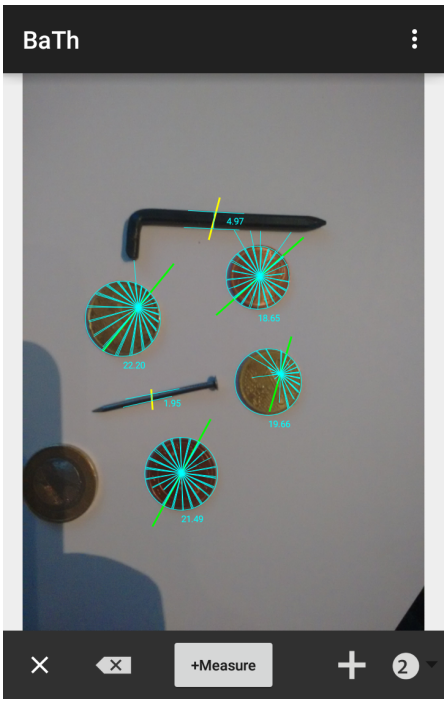

Abbildung 4.13: Messung IDs 1,2

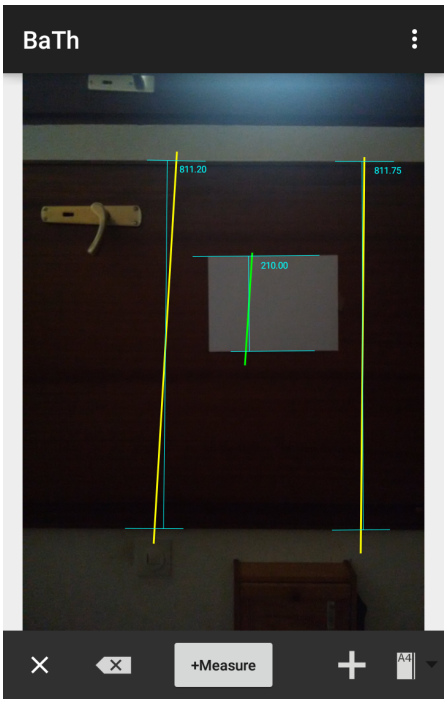

Abbildung 4.14: Messung IDs 12,13

wurden. Anhand der Testfälle müsste eine Gesamtunsicherheit von 50% für den Prototyp angesetzt werden, weil Messung 1 diese relative Messabweichung hat. Zu beachten ist dafür aber auch, dass Messung 1 und 2, per Screenshot in Abbildung 4.13 zu sehen, Extremfälle sind mit denen der Prototyp nicht gut umgehen kann.

Einerseits ist der Nagel aus Messung 1 so schmal, dass seine Kanten von der Objekterkennung nicht ordentlich differenziert werden können. Dies beeinträchtigt den Messfehler des eigentlichen Messauftrages. Dazu kommen starke Schatten, welche die Form der Münzen erweitern und so falsche Referenzmaße hervorrufen. In beiden Fällen wird deutlich, dass die Kantenerkennung für diese Aufgaben noch nicht optimiert ist. Schließlich sind durch eine sehr kleine Gegenstandsweite (die Kamera ist ca. 15cm vom Gegenstand entfernt) die Effekte der Linsenverzeichnung sehr stark und vor allem die Abbildung der in der Nähe des Bildrandes liegenden Münzen verfälscht.

Die Messung mit der nächstkleineren relativen Messabweichung, Messung 8, ist ebenfalls bei stärkeren Schatten und vor allem mit ähnlich kleiner Gegenstandsweite durchgeführt worden, genau wie Messung 6.

Bemerkenswert ist Messung 5 im Vergleich zu Messung 6, da eine niedrigere Abweichung bei gleichem Gegenstand und weniger Referenzmaßen auftritt. Beide Messungen basieren jedoch auf derselben Aufnahme. Dies zeigt, dass bei manchen Messungen ein besseres Ergebnis erzielt werden kann wenn Referenzmaße ignoriert werden, die vom System verfälscht erkannt werden (in diesem Fall waren starke Abweichungen durch Schatten zu sehen).

Allgemein kann trotzdem oft beim Anwenden bemerkt werden, dass die Messabweichung beim Hinzufügen von mehr Referenzmaßen geringer wird (was mathematisch nachvollziehbar ist, da die Ungenauigkeit von größeren Referenzmaßen nach Beobachtungen meist geringer ist und diese bei der Mittelung stärker bewertet wird).

Die Messungen 12 und 13 (Messung im Detail in Abbildung 4.14 dargestellt) zeigen, dass auch die Messung von großen Objekten wie einer Tür oder einer Wand möglich ist, indem ein Kalibrierblatt als Referenz verwendet wird und an das Objekt geheftet wird. In diesem Fall sind die Messungen sogar unter denen mit niedrigster Messabweichung, was auch zeigt, dass der Prototyp größere Objekte besser vermessen kann als kleine Objekte wie Nägel.

Insgesamt kann also keine wirkliche Unsicherheit für das entwickelte System angegeben werden. Es kann aber gesagt werden, dass selbst der Prototyp bereits ein in vielen Situationen nutzbares Bildmesssystem ist und in manchen Situationen den Messschieber übertrifft.

5 Fazit

In diesem Teil wird die Arbeit abschließend zusammengefasst und bewertet. Danach folgt ein Ausblick über mögliche Erweiterungen des entwickelten Bildmesssystems.

5.1 Zusammenfassung

Ziel dieser Arbeit war die Untersuchung und Beurteilung eines Bildmesssystems auf Basis eines Smartphones, sowie die Entwicklung eines solchen Systems, um dessen möglichen Nutzen zu überprüfen. Zu diesem Zweck wurden die wichtigsten Fehlerpotenziale betrachtet, wobei nicht nur die Genauigkeit des Ergebnisses an erster Stelle stand. Auch die Unterstützung des oft unerfahrenen, aber stets Fehler-verursachenden Anwenders spielte besonders bei der Entwicklung des eigentlichen Systems eine wichtige Rolle.

Verschiedene Lösungen für die Fehlerkorrektur, teils der Bildmesstechnik entnommen und teils neu entworfen, wurden verglichen und für ihre Nutzbarkeit in einem Smartphone-basierten Messsystem eingestuft. Aus den gewonnenen Erkenntnissen wurde ein Prototyp eines solchen Bildmesssystems für die Smartphone-Plattform Android entwickelt. Bei seinem Entwurf war besonders die Optimierung und Verhinderung von Anwendungsfehlern ein wichtiges Kriterium, denn die Behandlung von bildmesstechnischen Fehlern konnte in Fällen wie der Linsenverzeichnung analog zu der Behandlung in der allgemeinen Bildmesstechnik durchgeführt werden.

Der entwickelte Prototyp ermöglicht nach Aufnahme des zu vermessenden Bildes die Markierung von Referenzobjekten und Messdistanzen. Dabei werden die Benutzereingaben vom System optimiert, indem das Bild durch einfache Objekterkennung analysiert wird. Auf Basis der Referenzmaße werden die markierten Messungen berechnet und auf dem Bild in Millimeter-Einheit angezeigt. Die Messergebnisse des prototypischen Systems weisen in vielen Situationen bei erfahrener Benutzung eine niedrigere Messunsicherheit als Messschieber und ferner kann das System größere Objekte vermessen als ein Messschieber.

5.2 Bewertung

Der entwickelte Prototyp ist ein Smartphone-basiertes Bildmesssystem, welches bereits nützlich ist und verwendbare Ergebnisse liefert. Er kommt zwar nicht an die ursprünglich gewünschte Messunsicherheit eines Messschiebers heran, jedoch ist seine Präzision im alltäglichen Leben meist vollkommen ausreichend und es gibt viele Situationen in denen das entwickelte System die erwünschten Vorteile bietet.

Im Prototyp konnte nicht die diskutierte Wirkung der manuellen Steuerung des Fokus betrachtet werden, da das Smartphone dies nicht unterstützte. In diesem Punkt, und allgemein im Bereich der manuellen Kontrolle von Kamerasensor und Linse, ist noch großes Optimierungspotenzial der Messgenauigkeit vorhanden. Auch die untersuchte Korrektur von Linsenverzeichnungen stellt eine weitere Verbesserung der erreichbaren Präzision dar, die in der letztendlichen Entwicklung des Prototypen anderen Programm-Modulen weichen musste.

Der Grund dafür bestand vor allem darin, dass mir die Entwicklung der Eingabeoptimierung für den Prototypen wesentlich wichtiger war, weil sein Entwurf die Herausforderung der Mensch-Maschine-Interaktion darstellte.

Daraus wurde jedoch auch die größte Herausforderung der Arbeit deutlich, nämlich die, im Rahmen der Bachelorarbeit sowohl in der Softwareentwicklung des Prototypen als auch in der technischen Betrachtung der Messproblematik die wichtigsten Bereiche vertieft zu behandeln, weil beide Bereich sehr umfangreich und schwer zu begrenzen sind.

5.3 Ausblick

Im zuvor dargestellten Sinne können für den Entwurf einer auf dieser Arbeit aufbauenden Software einige Bereiche, besonders der Software vertieft werden.

Die Kalibrierung der Kameraeigenschaften wie Linsenverzeichnung sollte vorgenommen werden, wobei ein großer Vorteil nicht nur die gewonnene Genauigkeit wäre. Sie würde die Möglichkeit bieten, diese Kalibrierung auch für andere Smartphones mit dem gleichen Kameratyp zu verwenden. Dadurch würde die zusätzliche Beeinträchtigung der Benutzung (durch das Hinzufügen der Kalibrierung) nur für wenige Anwender, und auch nur einmalig auftreten.

Besonders hilfreich ist dafür eine manuelle Steuerung vieler Komponenten der Kamera, wobei Androids Camera2-API diese Möglichkeiten bereits anbietet. Hier ist ein wichtiger Schritt der Industrie notwendig, diese manuelle Steuerung auch zu ermöglichen weil so die Kamerasysteme von Smartphones auf Anwendungsebene enorm verbessert werden können. Neben den in der Arbeit erwähnten Vorzügen des FULL-Hardwaresupports gibt es auch die Möglichkeit der API, Mehrpunkt-Fokussysteme zu unterstützen. Diese von Anwendungsebene aus zu kontrollieren, ist nicht nur für Bildmesssysteme vorteilhaft, sondern auch für Smartphones, die eine vollwertige Kamera ersetzen wollen.

Eine andere Erweiterung des entwickelten Prototypen wäre die Erweiterung der Objekterkennung, eventuell sogar um die Erkennung von 3D-Modellen. Besonders in industriellen Anwendungsgebieten wäre eine Ausgabe des Messergebnisses als 3D-Modell für die Nutzung in anderen technischen Programmen wie CAD-Software eine sehr gute Option.

Insofern bleibt noch viel an dem Prototypen zu entwickeln, um ein vollwertiges Smartphone-basiertes Bildmesssystem zu erhalten. Aber selbst der entwickelte Prototyp weist eine hohe Nutzbarkeit auf und unterstützt damit die These, dass Smartphone-basierte Bildmesssysteme in vielen Situationen den Alltag erleichtern können.

Glossar

API Application programming interface, bietet Schnittstellen zu einem bestehenden Software-System zur externen Nutzung an

App Application Software oder Anwendung, besonders auf mobilen Plattformen übliche Abkürzung

GUI Graphical User Interface oder auch graphische Oberfläche

SDK Software Development Kit, Softwaresammlung welche die Erstellung von neuer Software für einen speziellen Bereich ermöglicht

XML Extensible Markup Language, Notationssprache für hierarchisch strukturierte Daten

Activity In sich geschlossene Anwendungsseiten, meist mit eigener Oberfläche

Hardware support level Klassifizierung des unterstützten Funktionsumfangs von Kameras in der Androids Camera2 API

Usability Bedienbarkeit und Benutzerfreundlichkeit von Software

Literaturverzeichnis

[1] M. M. Volgger, “Gegenstandsweite - Bildweite.” [Online]. Available: https://www.univie.ac.at/mikroskopie/1_grundlagen/optik/opt_linsen/3c_bildweite.htm

[2] Open Handset Alliance, “Android - Geschichte,” 2016. [Online]. Available: https://www.android.com/intl/de_de/history/

[3] A. F. Simond, “Camera2 API on MWC 2015 devices: Galaxy S6, HTC One M9 and more Lollipop devices,” Mar. 2015. [Online]. Available: https://spectrastudy.com/camera2-api-on-mwc-2015-devices/

[4] M. S. Nixon and A. S. Aguado, *Feature extraction & image processing for computer vision*, 3rd ed. Amsterdam [u.a.]: Acad. Press, 2012.

[5] Android, “Android version market share 2015 | Statistic,” Oct. 2015. [Online]. Available: http://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/

[6] M. M. Volgger, “Linsengleichung.” [Online]. Available: https://www.univie.ac.at/mikroskopie/1_grundlagen/optik/opt_linsen/4a_linsengleichung2.htm

[7] J. Zhang, “Angewandte Sensorik,” Hamburg, Oct. 2004. [Online]. Available: https://tams.informatik.uni-hamburg.de/lehre/2004ws/vorlesung/angewandte_sensorik/vorlesung_08.pdf

[8] V. Academy, “Lexikon der Bildverarbeitung - Verkippung,” 2006. [Online]. Available: http://geffe.de/mv_wbuch/np/6D6CB660CDBE4F01C1256D9500526C01.htm

[9] A. Mertins, *Signaltheorie: Grundlagen der Signalbeschreibung, Filterbänke, Wavelets, Zeit-Frequenz-Analyse, Parameter- und Signalschätzung*. Springer-Verlag, Dec. 2012.

[10] Deutsches Institut für Normung e.V., *DIN EN ISO 216 2007-12*. Berlin: Beuth-Verlag, 2007.

[11] European Central Bank, "Gemeinsame Seiten," 2016. [Online]. Available: http://www.ecb.europa.eu/euro/coins/common/html/index.de.html

[12] OpenCV Library, "Camera calibration With OpenCV," 2015. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[13] M. Gralak and T. Stark, *Schnelleinstieg App Usability [plattformübergreifendes Design: Android, Apple IOS und Windows Phone ; 35 Checklisten für die Praxis].* München: Franzis, 2015.

[14] B. Murphy and R. D. Morrison, Eds., *Introduction to environmental forensics*, 2nd ed. Amsterdam: Academic Press, 2007.

[15] P. P. L. Regtien and L. Finkelstein, *Measurement science for engineers.* London [u.a.]: Kogan Page Science, 2004.

[16] N. Menn, *Practical optics.* Amsterdam [u.a.]: Elsevier Academic Press, 2004.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 23. März 2016 Jannik Martin Iacobi