Kamera-basierte Detektion von Bedienelementen im Fahrstuhlinnenraum

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von:	Lorenz Fichte
geboren am:	05.12.1990
in:	Neubrandenburg
Gutachter(innen):	Prof. Dr. Peter Eisert
	Prof. Dr. Beate Meffert

eingereicht am: ..... verteidigt am: .....

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Kapitelübersicht . . . 2

2 Stand der Technik **3**
- 2.1 Gegenüberstellung . . . 4
  - 2.1.1 Hardware . . . 4
  - 2.1.2 Annahmen . . . 4
  - 2.1.3 Algorithmen . . . 5
- 2.2 Zusammenfassung . . . 6

3 Detektionsverfahren **7**
- 3.1 Vorüberlegungen . . . 7
  - 3.1.1 Gemeinsame Eigenschaften von Fahrstuhl-Buttons . . . 7
  - 3.1.2 Probleme . . . 8
- 3.2 Pipeline zur Buttondetektion . . . 9
- 3.3 Stufe 1: Entzerrung . . . 11
  - 3.3.1 Kantenbild . . . 11
  - 3.3.2 Hough-Transformation . . . 12
  - 3.3.3 Berechnung paralleler Linien im Hough-Raum . . . 15
  - 3.3.4 Berechnung der Homographie . . . 19
- 3.4 Stufe 2: Detektion der Bedienelemente . . . 22
  - 3.4.1 Initiale Detektion der Formen . . . 22
  - 3.4.2 Zusammenfassen in einen Graphen . . . 25
  - 3.4.3 Morphologische Operationen zur Kantenverbesserung . . . 26
- 3.5 Stufe 3: Template-Matching . . . 29
  - 3.5.1 Verfahren . . . 29
  - 3.5.2 Neuberechnung des Button-Graphen . . . 31
- 3.6 Implementierung . . . 32
  - 3.6.1 Kantenbild . . . 32
  - 3.6.2 Rektifizierung . . . 32
  - 3.6.3 Formendetektion und Graphenerstellung . . . 32
  - 3.6.4 Template Matching . . . 33
  - 3.6.5 Evaluation . . . 33

4 Auswertung **35**
4.1 Statistische Auswertung 35
4.1.1 Precision und Recall 36
4.1.2 Parameter 36
4.1.3 Ergebnisse 38
4.2 Beispielbilder 42
4.2.1 Probleme 45

5 Ausblick **46**
5.1 Zusammenfassung 46
5.2 Verbesserungsvorschläge und Erweiterungen 46

A Bilder **48**

B Quellen **51**
B.1 Bildquellen 51
Literatur 55

Abbildungsverzeichnis

1.1 Verdeckter Stuhl, Quadrat 2

3.1 Fahrstuhlbild mit störenden Strukturen 8
3.2 Fahrstuhlbuttons mit schlechten Kanten 9
3.3 Schematische Pipeline 10
3.4 Fahrstuhlbild mit dazugehörigem Kantenbild 11
3.5 Gerade nach Hesseschen NF 13
3.6 Prinzip der Hough-Transformation 14
3.7 Fahrstuhlbilder mit Geraden und dazugehörigen Hough-Räumen 16
3.8 Beste gefundene Kurve für horizontale bzw. vertikale Punkte im HS 18
3.9 Mapping für Eckpunkte 20
3.10 Im Fahrstuhlbild detektierte Ebene und rektifiziertes Fahrstuhlbild 21
3.11 Detektierte Beispielformen 24
3.12 Schematische Graphendarstellung 26
3.13 Anwendung des Dilatationsoperators auf Kantenbild 27
3.14 Anwendung des morphologischen Operators Schließen auf das Kantenbild 28
3.15 Prinzip des Template-Matchings mit den resultierenden Ergebnisräumen 30
3.16 Ergebnis der Detektion auf rektifiziertem Bild 31
3.17 Fahrstuhlbild mit dazugehöriger Buttonmaske 33

A.1 Fahrstuhlbilder Teil 1.1 48
A.2 Fahrstuhlbilder Teil 1.2 49
A.3 Fahrstuhlbilder Teil 2 50

Tabellenverzeichnis

4.1 Wahrheitsmatrix 36
4.2 einstellbare Parameter 37
4.3 Vergleich der Ergebnisse bei variablen und fixen Canny-Schwellwerten 38
4.4 Dilatation im Vergleich zum Schließen 38
4.5 Auswirkungen der Rektifizierung 39
4.6 Ergebnisse auf ausgewählten Bildern 39
4.7 Auswirkungen des TM 40
4.8 Ergebnisse bei veränderten TM-Parametern 41

Kapitel 1

Einleitung

1.1 Motivation

Fahrstühle sind nützliche und in der Regel auch nicht mehr wegzudenkende Hilfsmittel, um Menschen und Gegenstände schnell von der einen in die andere Etage zu transportieren. Ihre äußere Form und Gestalt sind dabei meist sehr unterschiedlich, auch ihr Innenleben variiert sehr stark. Das einzige, was mit großer Sicherheit gesagt werden kann, ist, dass die meisten Fahrstühle Knöpfe (Buttons) besitzen, mit denen die Zieletage bestimmt, die Tür geöffnet oder geschlossen oder der Alarm betätigt werden kann, falls der Fahrstuhl stecken bleibt. Wie diese Bedienelemente aussehen, wo sie sich befinden und in welcher Art und Weise sie angeordnet sind, ob horizontal, vertikal oder ungeordnet, ist oft nicht standardisiert.

Der Mensch ist von Natur aus sehr gut in der Lage, Objekte optisch und semantisch voneinander zu unterscheiden. Er kann Objekte und Formen gut differenzieren, selbst wenn sich diese nur erahnen lassen, d.h. wenn sie teilweise oder zu großen Teilen verdeckt sind. So ist es beispielsweise ein Leichtes zu erkennen, dass sich in Abbildung 1.1 a) ein Stuhl befindet obwohl dieser sehr stark durch eine Mauer verdeckt ist. Es ist außerdem nicht schwer für einen Menschen, Formen zu finden, die als solche eigentlich nicht existieren und erst durch andere entstehen, wie in Abbildung 1.1 b) erkennbar. So einfach, wie das Identifizieren von verdeckten Objekten für den Menschen ist, so einfach ist es auch, in einen beliebigen Fahrstuhl den Bereich ausfindig zu machen, wo sich die Knöpfe bzw. die Bedienkonsole befindet, die einen zur gewünschte Etage bringen.

Andere Menschen jedoch, deren Sehfähigkeit eingeschränkt oder vollständig verloren gegangen ist, müssen sich anders im Leben zurechtfinden, insbesondere in einem Fahrstuhl. In einigen Fällen besitzen die jeweiligen Knöpfe auf dem Panel bereits Brailleschrift[1], um solchen Menschen mit starker Sehbehinderung eine Orientierung zu ermöglichen und die gewünschte Zieletage auszuwählen. Dafür muss allerdings vorausgesetzt sein, dass die Person bereits weiß, wo sich die Konsole befindet, worauf in der Regel keine Blindenschrift hinweist. Eine Möglichkeit zur Unterstützung stellt eine auf dem Mobiltelefon integrierte App dar, die durch eine Kamera die Umgebung analysieren und durch akustische Signale auf Ort und Bedeutung der Knöpfe hinweisen könnte.

In der Robotik ist oftmals die autonome Erkundung und selbstständige Orientierung eines Roboters von entscheidender Bedeutung. In der Medizin werden verstärkt Roboter eingesetzt, um dem Menschen sowohl bei komplexen Anwendungsszenarien wie Operationen, als auch bei der alltäglichen Pflege hilfebedürftiger Menschen zu assistieren. Dabei kommen sogenannte Service-/Medizin- bzw. Personal-Roboter immer öfter zum Einsatz. Da sich das Aufgabengebiet eines solchen Roboters beispielsweise im Krankenhaus im Allgemeinen nicht nur auf eine Etage erstreckt, sondern

[1] Blindenschrift, 1825 von Louis Braille entworfen [Wik14]

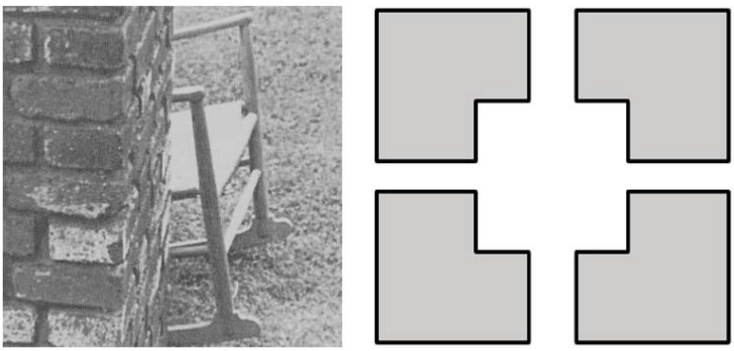


Abbildung 1.1: Abbildung a) zeigt einen verdecktes Objekt, das der Betrachter eindeutig als Stuhl einordnen kann, aus [ZG04], Abbildung b) zeigt vier beschnittene Quader, woraus der Betrachter einen weiterer Quader interpretiert, obwohl dieser als solcher nicht existiert

dieser womöglich im gesamten Gebäude eingesetzt werden soll, muss dieser auch in der Lage sein, sich selbstständig von einem Stockwerk in das Nächste zu bewegen und muss sich dadurch in einem Fahrstuhl orientieren können. Eine Teilaufgabe dieses komplexen Anwendungsfalls ist es daher, zunächst mögliche Buttons im Innenraum des Fahrstuhls zu detektieren, was der Hauptschwerpunkt dieser Bachelorarbeit sein soll.

In der vorliegenden Arbeit soll anhand von Methoden der Bildverarbeitung und Objekterkennung das Problem, Bedienelemente in einem Fahrstuhlinnenraum zu finden, gelöst werden, um den Lösungsansatz in Hinblick auf Menschen mit Sehbehinderungen, als auch Roboter einsetzen zu können.

1.2 Kapitelübersicht

Die nächsten Kapitel sind wie folgt aufgebaut: Im Kapitel 2 wird der aktuelle Stand der Technik beschrieben. Hier gibt es vor allem im Bereich der autonomen Orientierung von Robotern in Fahrstühlen viel betriebene Forschung. Das Kapitel 3 beschäftigt sich mit den existierenden Algorithmen und Methoden, die benutzt oder teils erweitert wurden, um die konkrete Buttonerkennung durchzuführen. Es wird dabei zunächst auf das Gesamtverfahren, also die Pipeline eingegangen und die drei Stufen der Detektion schrittweise erläutert. Dabei werden erste Ergebnisse präsentiert. Ein Abschnitt widmet sich hier der eigenen Implementierung. Im Kapitel 4 werden die Ergebnisse evaluiert und Erkenntnisse gezeigt. Das 5. und letzte Kapitel beinhaltet eine Zusammenfassung und es werden Vorschläge eingebracht, welche Erweiterungen zu dem entwickelten Verfahren denkbar sind. Im Anhang befinden sich alle verwendeten Bilder mit dazugehörigen Quellen.

Kapitel 2

Stand der Technik

Besonders im Bereich der Robotik wird viel Forschung betrieben, um die Fähigkeit von Robotern zur Navigation und autonomen Erkundung noch weiter zu verbessern, da dies eines der wichtigsten Felder solcher mobilen Systeme darstellt [LTJ+09, KAyO07]. So zählt hierzu gerade bei Service-Robotern, die auf verschiedenen Etagen eingesetzt werden sollen, als Teilproblem die Detektion von Bedienelementen innerhalb von Fahrstühlen.
Im Folgenden soll ein Überblick über die Arbeiten aus aktueller Forschung und Entwicklung gegeben und diese in ihren Methoden untereinander verglichen werden. Im letzten Abschnitt wird der Forschungsstand kurz zusammengefasst.

- In der Arbeit von [TSG+13] wird als Zielstellung ein experimentelles, semi-humanuides System mit autonomer Mehr-Etagen-Navigation angestrebt, das mit Laser-Scanner und Stereokamera arbeitet.
- In dem Paper [KCRN10] geht es besonders um die autonome Interaktion von Robotern mit unbekannten Fahrstuhlbedienelementen und die auftretenden Schwierigkeiten durch beispielsweise Buttonform und Beleuchtung. Dabei wird für die Detektion auf eine einfache Monokamera sowie Laser-Scanner zurückgegriffen.
- Eine weitere Arbeit von [MIS05] beschäftigt sich dagegen mit der Mensch-Roboter Interaktion und der komplexen Aufgabe einen Roboter zu lehren, wie dieser beispielsweise in einem mehrstöckigen Haus auf verschiedenen Etagen autonom agieren kann.
- Die Arbeit [LTJ+09] stellt sich die Frage, wie die Detektion von Bedienelementen bei stark oder vollständig spiegelnden Oberflächen durchgeführt werden kann, wie es oft der Fall ist, wenn die Buttons aus Metall sind, wozu auch hier eine Stereokamera eingesetzt wird.
- [KAyO07] ist ein Paper, das unter anderem ein künstliches neuronales Netzwerk zur Detektion der Fahrstuhlbedienelemente benutzt und einen Laser-Scanner für die Abstandsbestimmung verwendet.
- Die Autoren aus [YDLH09] unterteilen die Detektion der Buttons in zwei Teilaufgaben, wobei zunächst das gesamte Panel und anschließend die individuellen Buttons gesucht werden.
- Die Abschlussarbeit [Lum09] bietet einen guten Überblick über verschiedene Verfahren zur Objekterkennung und benutzt viele etablierte Verfahren zur Bestimmung der Buttons über eine Monokamera.

- In der Arbeit [DKK+08] werden allgemein die Erkenntnisse im Bereich Forschung mit Robotern und Erfahrungen im RoboCup[1] dargelegt, wobei in einem Teilabschnitt speziell auf die Buttondetektion im Fahrstuhl eingegangen wird, allerdings nicht so ausführlich wie in den zuvor genannten Arbeiten.

2.1 Gegenüberstellung

Alle der vorgestellten Arbeiten beschreiben die Detektion von Bedienelementen im Fahrstuhlinnenraum als eine nicht triviale Herausforderung, da unter anderem die Variationen der Knöpfe enorm ist. Die einzelnen Verfahren unterscheiden sich hinsichtlich der benutzten Hardware, der Grundannahmen und den verwendeten Algorithmen voneinander. Die Gemeinsamkeiten sowie Unterschiede sollen im Folgenden herausgearbeitet und die gängisten Verfahren genannt werden.

2.1.1 Hardware

Da Roboter mit der unterschiedlichsten Hardware ausgestattet sein können, wurden verschiedene Voraussetzungen an die verwendete Hardware gestellt: In einigen Verfahren wurden Stereokameras eingesetzt, um Tiefeninformationen zu messen und eine räumliche Lokalisierung zu ermöglichen [TSG+13, MIS05, LTJ+09, KAyO07]. Dabei wurden bei [MIS05] sogar omnidirektionale Kameras[2] benutzt.
3D-Laser-Scanner wurden in fast allen Systemen verwendet [TSG+13, KCRN10, MIS05, KAyO07, DKK+08]. In [TSG+13, MIS05, KAyO07, DKK+08] wurden LRF[3] verwendet, wobei dies bei [TSG+13] im Speziellen ein planarer LiDAR[4]-Scanner ist. In [KCRN10] wurde ein rotierender Linien-Laser-Scanner eingesetzt. Die Abstandsberechnung erfolgt bei solchen Systemen über die ausgesendeten Laserstrahlen.
Das Verfahren von [KCRN10, YDLH09, Lum09] benutzt dagegen nur eine einfache Monokamera, wobei die Tiefeninformation in [YDLH09] durch Ultraschall gemessen werden kann. [Lum09] nimmt dagegen keine Abstandsberechnungen vor, sondern setzt einen verbindlichen Abstand von weniger als 50 cm zu den Bedienelementen voraus.
Es scheint allgemein sinnvoll, dass bei Robotersystemen neben den Kameras auch weitere Sensoren benutzt werden, um die genaue räumliche Lokalisierung zu ermöglichen. Die ausschließliche Benutzung von Monokameras in solchen Robotersystemen kann hier nicht erkannt werden.
Im Gegensatz zu der Vielzahl der genannten Sensoren in den vorgestellten Referenzsystemen, wurde in der vorliegenden Arbeit wie in [Lum09] nur eine einfache Monokamera verwendet, die keine weiteren Tiefeninformationen messen kann. Der Grund liegt in der im Kapitel 1 genannten Motivation das System bestenfalls auch für Menschen mit Sehbehinderungen nutzbar zu machen, was idealerweise dann auch mit Hilfe von geringen Ressourcen möglich sein sollte (beispielsweise über eine Smartphone-App).

2.1.2 Annahmen

Zur Detektion der einzelnen Bedienelemente wurde in den verschiedenen Arbeiten unterschiedliche Annahmen über Form und Größe der jeweiligen Buttons gemacht.
Besonders interessant ist hierbei die Arbeit von [KCRN10], da dort die durchschnittliche Buttongröße anhand von Fahrstuhl-Richtlinien für *Amerikaner mit Behinderungen*[5] berechnet wurde.

[1] RoboCup: Weltmeisterschaftsturnier in denen Roboterteams, ähnlich wie im Fußball gegeneinander antreten
[2] 360° Kamera
[3] Laser Range Finder
[4] Light detection and ranging
[5] `http://www.ada.gov/2010ADAstandards_index.htm`

Demnach werden nur Bedienelemente erkannt, die einen größeren Durchmesser als 1.9 cm haben und dessen entsprechendes Symbol der Etage oder sonstige andere Bedeutung direkt links neben dem Button liegt.
In den Arbeiten [LTJ+09, Lum09] werden auch bestimmte Größen der Buttons angenommen. Dabei sollen in [LTJ+09] einfach nur die Ausmaße vor der Detektion bekannt sein, wohingegen in [Lum09] eine Durchschnittsgröße von $2 - 4$ cm angenommen wird. [KAyO07] beschreibt als Teilschritt auch eine Größen-Filterung der möglichen Buttons, durch Prüfen der Seitenverhältnisse und Flächen, gibt hierbei allerdings keine exakten Werte an. Um die vorgeschriebenen Größen einhalten zu können, wird in den jeweiligen Fällen eine metrische 3D-Rekonstruktion der Szene durchgeführt, um die genauen Abmaße der Objekte zu bestimmen.
Eine weitere klassische Einschränkung zur Vereinfachung des Detektionsverfahrens ist die Form der Buttons. In den Arbeiten [LTJ+09, KAyO07] werden hier rechteckförmige Buttons vorausgesetzt, wohingegen in [MIS05, Lum09] auch zusätzlich kreisförmige Bedienelemente erlaubt sind. Bei den Verfahren [TSG+13, KCRN10, YDLH09, DKK+08] werden dagegen keine expliziten Einschränkungen zur Form explizit genannt, wobei sich im Fall von [KCRN10] wahrscheinlich wieder auf die genannten Richtlinien bezogen wird. Im Abschnitt 3.1.1 wird für das vorgestellte Verfahren ähnliche Annahmen über Form und Verzerrung gemacht, die auf gemeinsamen Beobachtungen in Fahrstühlen beruhen und die Detektion vereinfachen soll.
Es ist auffällig, dass wenige der Arbeiten genauer auf die perspektivische Verzerrung eingehen, die möglicherweise im Bild enthalten ist, wenn die Aufnahme nicht direkt orthogonal zum Bedienfeld erfolgt. In [Lum09] wird zwar von einer Verzerrung ausgegangen, diese wird aber nicht weiter beachtet, da das beschriebene Verfahren bis zu einen gewissen Grad relativ robust gegenüber perspektivischer Verzerrung ist und ein Winkel kleiner als 30° vorausgesetzt wird. In [TSG+13] wird dagegen ein Verfahren vorgestellt, das von vornherein invariant gegenüber dem Betrachtungswinkel und damit auch der Verzerrung ist.
In der vorliegenden Arbeit findet, sofern möglich, eine Verzerrungskorrektur (Rektifizierung) statt und wird im Abschnitt 3.3 erläutert.

2.1.3 Algorithmen

Es gibt eine Vielzahl an verwendeten Algorithmen, wobei bei den verschiedenen Arbeiten teilweise die gleichen Verfahren genutzt wurden.
So wurde beispielsweise in [MIS05, YDLH09] die Hough-Transformation (HT), genauer erläutert in Abschnitt 3.3.2, für Kreise angewendet, um speziell runde Buttons zu detektieren. In [LTJ+09] wurde zwar auch die HT für Linien erwähnt, für die eigentliche Detektion allerdings nicht umgesetzt. Die Begründung dafür ist, dass die Buttonbegrenzungslinien meist nur kurz auftreten und daher andere Verfahren besser geeignet seien, wie beispielsweise ein einfacher Algorithmus zum Verfolgen von Bildkanten. [Lum09] benutzt ebenso die HT, allerdings für Linien. Auch hier wird der Algorithmus nicht direkt für die Suche nach den Bedienelementen eingesetzt, sondern nur als Hilfsmittel verwendet, um die Kanten möglicher Buttons im Bild miteinander zu verbinden.
Für die eigentliche Detektion der Buttons werden teils sehr unterschiedliche Herangehensweisen verwendet. In [TSG+13] wird unter anderem die Verwendung eines speziellen Template-Matching-Verfahrens (siehe Abschnitt 3.5) diskutiert. Dies erweist sich als robust gegenüber kleinen Änderungen von Lichtbedingungen, allerdings wird dadurch auch eine initiale Detektion von Buttons vorausgesetzt, da ein allgemeines Template ohne Information über Form oder Größe nicht erstellt werden kann, da die Form in verschiedenen Fahrstühlen zu stark variiert. Dazu werden bestimmte charakteristische Bildpunkte mittels SIFT-Algorithmus[6] [Low99] extrahiert. Auch in [LTJ+09, Lum09] werden diese rotations- und skalierungsunabhängigen Merkmale verwendet, in

[6] *Scale-invariant feature transform*

[DKK+08] dagegen werden SURF-Merkmale[7] [BETG08] eingesetzt, um die Buttons zu extrahieren. [LTJ+09] benutzt außerdem Harris Corners [HS88] als Merkmalspunkte.
In [MIS05] wird wie in [TSG+13] eine Template-Matching vorgeschlagen, wobei allerdings die Button-Templates nicht autonom, sondern durch menschliche Interaktion gelernt werden müssen.
In den meisten Fällen werden nicht nur die Buttons erkannt, sondern auch deren Bedeutung bestimmt. Dazu wurde interessanterweise in [KCRN10] sowie in [Lum09] falsch erkannte Zahlen über die logische Anordnung korrigiert, d.h. es wird die Eigenschaft ausgenutzt, dass die Etagennummern der Reihenfolge nach geordnet auftreten und beispielsweise Button 9 gefolgt von 10 und vor 8 auftaucht. Dies wurde teilweise über das HMM[8] realisiert.
In [Lum09] wird wie in der vorliegenden Arbeit ein konturbasiertes Verfahren verwendet (siehe Abschnitt 3.4.1), um die speziellen Buttonformen ausfindig zu machen.

2.2 Zusammenfassung

Es konnte gezeigt werden, dass das Interesse und die Forschung bei der Buttonerkennung besonders in unbekannten Fahrstühlen gerade im Zusammenhang mit autonomer Navigation von Robotern sehr groß ist.
Dabei werden oftmals ähnliche Hardwarekomponenten verwendet. Die Benutzung von Stereokameras in Kombination mit Laser-Scannern scheint allgemein sehr üblich zu sein, wodurch Probleme mit Verzerrungen und die Entfernungsbestimmung zum Button-Panel gut gelöst werden können. Die Benutzung von einfachen Kameras, die nur ein Bild liefern, wird aber nicht grundsätzlich ausgeschlossen.
Der Vergleich hat gezeigt, dass die Verfahren viele Gemeinsamkeiten besitzen und sich hier viele bereits etablierte Methoden anbieten, um die Detektion durchzuführen. In einigen Arbeiten wurden, so wie in dem vorliegenden System, Verfahren wie HT und TM angewendet. Merkmalsbasierte Verfahren wie SIFT oder SURF sind hier auch häufig anzutreffen, werden allerdings wie auch gelernte Modelle in dieser Arbeit nicht benutzt.
Weiterhin ist eine Einschränkung der zu suchenden Buttons auf Form, Größe oder sonstige Grundannahmen in allen Arbeiten gemacht worden und scheint dadurch essentiell, um das komplexe Problem zu bewerkstelligen.
Im nächste Kapitel wird die eigene Vorgehensweise beschrieben und gezeigt welche teilweise hier schon genannten Techniken benutzt wurden, um die Fahrstuhlbedienelemente zu lokalisieren.

[7] *Speeded Up Robust Features*
[8] *Hidden Marcov Model*

Kapitel 3

Detektionsverfahren

Dieses Kapitel widmet sich hauptsächlich dem eigenen Verfahren. Dabei sind die nächsten Abschnitte wie folgt gegliedert: Im Abschnitt 3.1 gibt es allgemeine Vorüberlegungen, die dabei helfen können, die komplexe Aufgabe der Objekterkennung zu vereinfachen (3.1.1). Es werden außerdem auf Probleme und Herausforderungen hingewiesen, die bei dieser Art der Bilderkennungsaufgabe auftreten können (3.1.2). Im darauffolgenden Abschnitt 3.2 wird das eigene Verfahren mit der entsprechenden Pipeline vorgestellt und geklärt welche Algorithmen und Methoden verwendet wurden. Dazu beschreiben dann die Abschnitte 3.3, 3.4 und 3.5 die drei Stufen des Verfahrens. Im letzten Teil dieses Kapitels, dem Abschnitt 3.6, wird ein Überblick über die Implementierung der Algorithmen gegeben.

3.1 Vorüberlegungen

3.1.1 Gemeinsame Eigenschaften von Fahrstuhl-Buttons

Der Anwendungsbereich „Fahrstuhl“ ist sehr vielseitig und für eine Button-Objekterkennung relativ komplex, da die Formen und Anordnung der Elemente nicht einheitlich sein müssen. Trotzdem ist es möglich ein paar Verallgemeinerungen aufzustellen, welche die Aufgabe der Detektion vereinfachen können. Zwar ist das Innenleben von Fahrstühlen verschiedener Betreiber untereinander oft nicht standardisiert aber trotzdem gibt es oft Gemeinsamkeiten.

Hier ist eine kurze Auflistung häufig beobachteter Eigenschaften in verschiedenen Fahrstühlen:

1: Button-Panel ist immer vorhanden

2: Button-Panel immer links oder rechts vom Türbereich, selten bis nie gegenüber der Tür (zwar in dieser Arbeit nicht benutzt, kann für Roboter aber nützlich sein, siehe Kapitel 2)

3: Buttons haben oft einen Rand der sich vom Untergrund, auf denen sie sich befinden, abhebt

4: Buttons innerhalb eines Panels haben untereinander immer dieselbe Form und meist auch dieselbe Farbe

5: die Form der Buttons sind meist kreisförmig oder quadratisch, selten bis nie dagegen nur rechteckig

6: Buttons sind in der Regel direkt mit einer Zahl oder Symbol markiert oder haben eine Kennzeichnung in der Nähe, die auf dessen Bedeutung hinweist

7: Button-Panel befindet sich in einer Ebene[1]

3.1.2 Probleme

Anders als bei verwandten Arbeiten (siehe Kapitel 2), die komplexe Bilder von Stereo-Kameras oder Laser-Scanner als Eingabe benutzen, genügen in dieser Arbeit die Aufnahmen von einer einzelnen Kamera, wie sie bereits mit einem Mittelklasse-Smartphone gemacht werden können.
Bilder in geschlossenen Räumen sind, unabhängig vom Kameramodell, immer eine Herausforderung, wenn das Bild nicht zu stark unterbelichtet oder verwackelt sein soll. Aufnahmen im Fahrstuhlinnenraum verstärken diese Herausforderung, da die Lichtverältnisse nicht selten ungünstig bis schlecht sind. Einige der verwendeten Bilder wurden daher auch mit Blitz aufgenommen, wodurch aber teilweise neue Probleme entstehen. Oft ist die Verkleidung des Fahrstuhlinnenraums metallisch oder glatt, sodass die Oberflächen stark spiegelt, wenn zusätzlich der Blitz bei der Aufnahme verwendet wird. Außerdem sind dann häufig die verschiedenen Bereiche im Fahrstuhl unterschiedlich stark ausgeleuchtet, was die Detektion weiter erschwert. Teilweise ist der Fahrstuhl auch mit einem Spielgel ausgestattet, was die Reflexionen noch weiter verstärken. Wird ohne Blitz gearbeitet, kann es passieren, dass das Bild zu dunkel ist und die eigentlichen Bedienelemente nicht mehr erkannt werden können. Oder es entsteht wegen der Benutzung einer höheren Sensorempfindlichkeit (oft bezeichnet als ISO) ein starkes Rauschen, was dazu führt, dass im Kantenbild Strukturen entstehen, die eigentlich nicht vorhanden sind. Verwackelungen und verwaschene Bilder sind dann die Folge.

Abbildung 3.1: Es ist deutlich zu sehen, dass durch die geriffelte Metalloberfläche im Originalbild a) entsprechende störende Strukturen im Kantenbild b) entstehen

[1] es wird davon ausgegangen, dass das Panel selbst weder gebogen oder in sonstiger Art verformt, sondern planar ist

Das Kantenbild wird im Vorverarbeitungsschritt mit Hilfe des Canny Edge Algorithmus [Can86] erstellt (siehe Abschnitt 3.3.1). Eine weitere Herausforderung ist hier das Einstellen der entsprechenden Parameter. In manchen Fällen können wegen der geriffelten Oberfläche im betreffenden Fahrstuhl die Kantenbilder schlecht ausfallen, sodass Strukturen entstehen, die die Detektion sehr erschweren (siehe Abbildung 3.1). In manchen Fahrstühlen heben sich die Buttons von ihrer Umgebung relativ gut voneinander ab, wenn sie eine farbige Umrandung haben oder selbst im starken Kontrast zu der Umgebung stehen. Im Zusammenhang mit den daraus erstellten Kantenbildern ergibt sich dabei ein gut verwertbares Binärbild. Sind die Buttons aber in demselben einfarbigen, randlosen Farbton gehalten, wie der Untergrund auf dem sie sich befinden, ist es nahezu unmöglich daraus ein entsprechend nützliches Kantenbild zu erstellen, was die akkurate Detektion verhindert (siehe Abbildung 3.2).

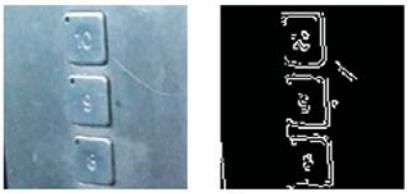


Abbildung 3.2: Ausschnitt von einem Buttonpanel, a) die Buttons und der Untergrund heben sich farblich kaum voneinander ab, sodass im resultierenden Kantenbild b) eher unbrauchbare Kanten entstehen

Die Formen der Buttons beschränken sich wie in Abschnitt 3.1.1 beschrieben auf kreisförmige und quadratische Elemente. Da eine Aufnahme vom Fahrstuhlinnenraum fast immer auch eine perspektivische Verzerrung mit sich bringt, können dadurch die Formen der Buttons nicht exakt auf Kreise und Quadrate abgeglichen werden, da die geometrischen Eigenschaften wie z.B. rechter Winkel nicht mehr gelten. Dies stellt eine weitere Herausforderung dar, da keine Stereobilder zur Verfügung stehen und die genaue Verzerrung nur anhand eines Bildes festgestellt werden kann.

3.2 Pipeline zur Buttondetektion

Im Wesentlichen besteht das Gesamtverfahren zur Buttondetektion aus drei Stufen. In der ersten Stufe, die als Vorverarbeitungsschritt angesehen werden kann, wird das Bild nach Möglichkeit perspektivisch entzerrt[2], indem die zwei Fluchtpunkte im Bild detektiert werden. Mit Hilfe dieser Fluchtpunkte werden die parallelen Linien im Bild bestimmt und eine Homographie berechnet, die das Bild entzerrt. Im Abschnitt 3.3 wird dieser Schritt genauer erläutert.
Die zweite Stufe ist die initiale Detektion von Buttons mit Hilfe von Konturen, die auf den Kantenbildern ausgeführt wird. Dazu beschreibt der Abschnitt 3.4 diesen Vorgang ausführlicher. In der dritten und letzten Stufe, die im Abschnitt 3.5 erklärt wird, werden die initial gefundenen Buttons genutzt, um weitere möglicherweise noch vorhandene Knöpfe per Template-Matching zu finden, die durch die erste Buttondetektion übersehen wurden.
Es kann passieren, dass die Suche nach den Fluchtpunkten, die im ersten Schritt durchgeführt wird, fehlschlägt. Dies kann zwei Gründe haben: Entweder das Kantenbild enthält zu wenige Informationen und ist daher zu schlecht, dass heißt zu verrauscht bzw. es entstehen zu viele Strukturen im Kantenbild durch Spiegelungen, sodass keine Fluchtpunkte extrahiert werden können. Oder aber das aufgenommene Bild ist nur minimal oder gar nicht verzerrt, da die Aufnahme nahezu

[2] rektifiziert

orthogonal[3] zu den Bedienelementen erfolgte. In diesem Fall ist auch keine Rektifizierung mehr notwendig, da die perspektivische Verzerrung so gering ist, dass eine Entzerrung kaum Verbesserung mit sich bringt, da sich die Fluchtpunkte nahezu im Unendlichen befinden. In beiden Fällen, wenn keine Fluchtpunkte detektiert werden konnten, wird auch keine Entzerrung durchgeführt und gleich zu Schritt zwei übergegangen. Abbildung 3.3 zeigt alle Verarbeitungsschritte noch einmal schematisch.

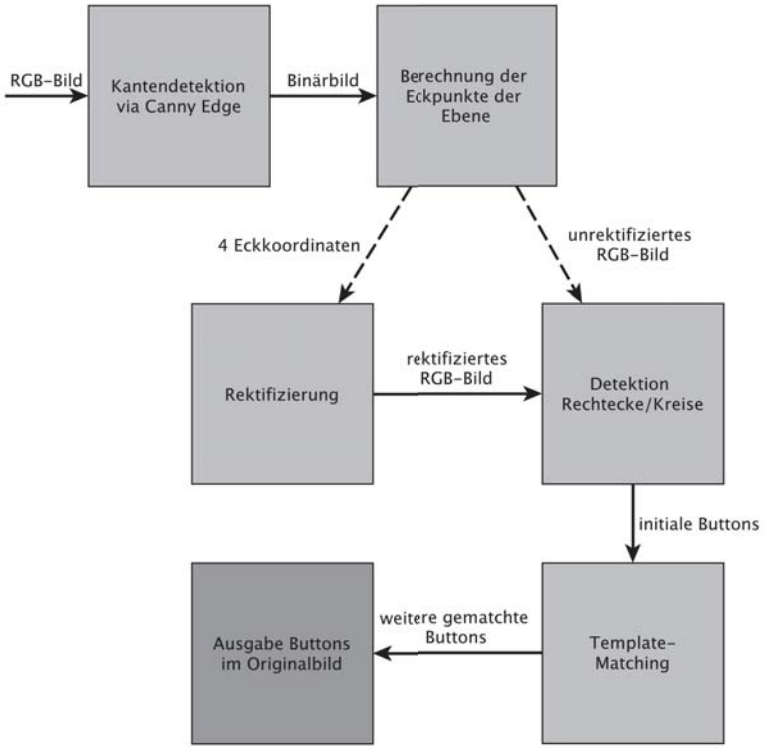


Abbildung 3.3: Pipeline vom initialen Eingangsbild bis zur Ausgabe der detektierten Bedienelemente

[3] senkrecht zur Ebene

3.3 Stufe 1: Entzerrung

Im Normalfall befindet sich ein Fahrstuhl-Panel links oder rechts von der Fahrstuhltür aus auf Brust- oder Hüfthöhe. Wenn sich der Betrachter nun etwa auf Höhe der Eingangstür aufhält, an welcher Stelle auch das Foto aufgenommen wird, kann davon ausgegangen werden, dass es im Bild zu einer perspektivischen Verzerrung kommt. Die Stärke der Verzerrung ist dabei sehr stark von dem Blickwinkel und der Entfernung abhängig, aus der die Aufnahme gemacht wird.
Um mit dieser Verzerrung umzugehen, gibt es zwei Möglichkeiten: Entweder es wird auf dem verzerrten Bild gearbeitet ohne dieses perspektivisch zu entzerren (zu rektifizieren). Dann müssen aber gesonderte Annahmen getroffen werden, damit die verzerrten Bedienelemente trotzdem noch detektiert werden. Oder aber es wird eine Rektifizierung des Bildes durchgeführt und dadurch die tatsächlichen, geometrischen Eigenschaften wiederhergestellt. In diesem Fall hätten dann beispielsweise die im Bild verzerrt abgebildeten, quadratischen Formen alle wieder gleiche Seitenlängen und senkrecht aufeinander stehende Kanten.
Der gewählte Ansatz dieser Arbeit versucht die beschriebene zweite Möglichkeit. Dazu werden im Bild die planare Flächen detektiert, auf dem sich auch die Bedienelemente des Fahrstuhls befinden. Mit Hilfe dieser Ebene kann das Bild dann wieder entzerrt werden. Die Vorgehensweise ist dementsprechend Folgende: Es werden im Bild zunächst die Kanten sichtbar gemacht. Dies erfolgt mit Hilfe des Canny Edge Algorithmus und ist im Abschnitt 3.3.1 beschrieben. Anschließend wird eine Detektion der im Bild befindlichen Linien über die Hough-Transformation durchgeführt. Auch wenn die Hough-Transformation (siehe Abschnitt 3.3.2) für die Suche nach kreisförmigen Objekten im Bild angewendet werden kann, wird sie in diesem Falle nur zur Geradenextrahierung genutzt, da durch das in Abschnitt 3.4 beschriebene Verfahren überzeugendere Ergebnisse entstanden sind. Nachdem die wichtigsten Linien extrahiert sind, können nun aus dieser Menge die Suche nach parallelen und senkrecht aufeinander stehenden Linien beginnen, was genauer in Abschnitt 3.3.3 erläutert wird. Die extrahierten Linien werden nun benutzt, um die gesuchte Ebene aufzuspannen, welche durch vier Eckpunkte begrenzt ist. Diese Eckpunkte werden verwendet, um die Entzerrungsmatrix (Homographie), die im Abschnitt 3.3.4 erklärt wird, berechnen zu können, um schlussendlich das Bild damit zu rektifizieren.

3.3.1 Kantenbild

Wie bereits erwähnt, wird als Vorverarbeitungsschritt der Hough-Transformation, beschrieben im Abschnitt 3.3.2, ein Binärbild in Form eines Kantenbildes benötigt, bei dem an einem Bildpunkt (x, y) nur die Information enthalten ist „Kante“ oder „keine Kante“.

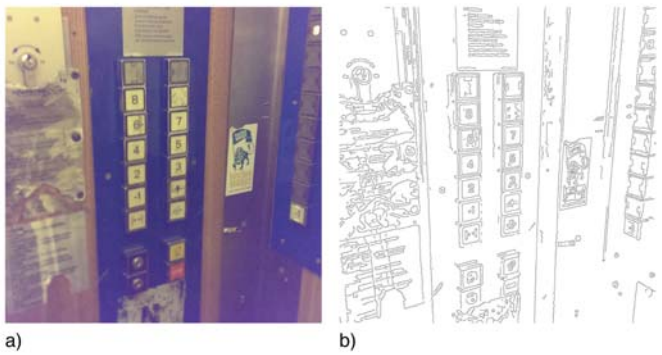


Abbildung 3.4: a) Originalbild b) dazugehöriges invertiertes Canny-Kantenbild

In dieser Arbeit wird der Canny Edge Algorithmus [Can86] genutzt. Der 1986 von John Canny entworfene Kantendetektionsalgorithmus bietet viele Möglichkeiten Kanten im Bild individuell sichtbar zu machen.
Im Abschnitt 3.1.2 wurden bereits auf Herausforderungen mit den einstellbaren Parametern des Algorithmus hingewiesen. Der Canny Edge Algorithmus ist speziell dazu ausgelegt die drei folgenden Kriterien zu erfüllen[4]: Niedrige Fehlerrate, gute Lokalisierung und minimale Antwort. Als Parameter benötigt der Algorithmus die Größe des verwendeten Sobel-Kerns und einen unteren und oberen Schwellwert, deren Bedeutung in [Can86] nachzulesen ist. Für gute Resultate wird ein Verhältnis oberer zu unterer Schwellwert von 3 : 1 oder 2 : 1 vorgeschlagen. Es wurde hier allerdings ein Verhältnis von 10 : 3 gewählt, da hierfür die besten Ergebnisse erzielt wurden. Für eine detaillierte Beschreibung des Verfahrens wird auf [WM01, S.89f.] verwiesen.
Die Schwellwerte können entweder fix eingestellt oder automatisch berechnet werden. Bei einer automatischen Berechnung wird der obere Schwellwert je Fahrstuhlbild mit Hilfe des Schwellwertverfahrens von Otsu berechnet[5] [Ots79] [FYY09]. Der untere Schwellwert ergibt sich dann durch das Produkt von oberem Schwellwert mit dem inversen Verhältnis. Im Abschnitt 4 wird die Verwendung von fixen und variablen Schwellwerten gegenübergestellt und ausgewertet.
Zwar sind auch andere Hochpassfilter[6] zur Kantendetektion wie der einfache Sobel-Operator, Roberts-Operator, Laplace-Operator oder das Prewitt-Filter, denkbar. Es hat sich allerdings herausgestellt, dass der Canny-Edge-Algorithmus am besten geeignet ist, da er in der Regel im Vergleich zu den anderen Algorithmen immer bessere Ergebnisse erzielt [CB12].

3.3.2 Hough-Transformation

Nachdem die Kanten im Bild sichtbar gemacht worden sind, kann nun die Hough-Transformation (HT) angewendet werden. Die HT ist allgemein ein Hilfsmittel der Bildverarbeitung, um geometrische Objekte in einem Kantenbild zu detektieren [DH72]. Das Verfahren ermöglicht es auch, die gewünschte geometrische Form im Bild zu finden, selbst wenn das gesuchte Objekt verdeckt ist [Ver91]. Um die Transformation auf beliebige Kurven (z.B. Kreise, Linien, etc.) anwenden zu können, müssen diese durch mathematische Funktionen eindeutig beschrieben werden können, d.h. sie müssen parametrisierbar sein. Als Eingabebild wird das im Abschnitt 3.3.1 erläuterte Kantenbild benötigt. Auch wenn die HT für beliebige geometrische Objekte angepasst werden kann, findet sie in diesem Verfahren nur für Linien Anwendung, daher soll die Funktionsweise des Algorithmus im Folgenden auch kurz anhand eines Beispiels für Geraden erklärt werden.
Um eine Linie eindeutig bestimmen zu können, wird die Hessesche Normalform für Geraden benutzt:

$$d = x \cdot \cos\theta + y \cdot \sin\theta. \tag{3.1}$$

Dabei ist d der Abstand vom Koordinatenursprung $(0, 0)$ zum Lotfußpunkt P_0, θ der Winkel zwischen x-Achse und der Geraden und (x, y) der jeweilige Pixel im Kantenbild, der die Gleichung 3.1 erfüllt und sich deshalb auf der durch (θ, d) parametrisierten Geraden befindet. Die Abbildung 3.5 zeigt dies noch einmal schematisch.

Formel 3.1 hat gegenüber der bekannten Geradengleichung $y = m \cdot x + n$ den Vorteil, dass auch senkrechte Geraden dargestellt werden können, ohne dass der Anstieg m unendlich wird. Alternativ könnte auch die explizite Form verwendet werden, dann müsste die HT allerdings einmal auf das normale Bild und noch einmal auf ein um 90° gedrehtes Bild angewendet werden.

[4] siehe `http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html`
[5] sehr gut beschrieben in `http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html`
[6] Hochpassfilter: die im Bild enthaltenen hohen Frequenzen, die die Kanten ausmachen, werden durchgelassen, tiefe Frequenzen dagegen werden unterdrückt, wodurch die klassischen Kanten sichtbar werden

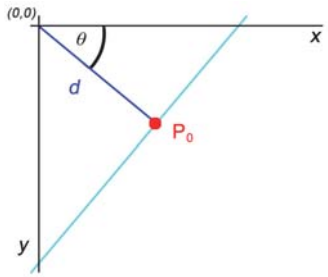


Abbildung 3.5: Geradendarstellung nach der Hesseschen Normalform

In dem Hough-Raum[7] (HR), der den parametrisierten Raum darstellt, sind die Parameter wiederzufinden: Die x-Achse stellt den Winkel θ und die y-Achse den Abstand d dar. Die Größe des HR ist demnach bedingt durch die Größe des Bildes, da der größte d-Wert, auch d_{max} genannt, nur so groß wie die Bilddiagonale sein kann. Der Wertebereich des HR liegt dementsprechend bei $[-d_{max}, +d_{max}]$ und der Definitionsbereich ist durch $[-\frac{\pi}{2}, +\frac{\pi}{2}]$ beschränkt.
Die resultierenden Eigenschaften sind Folgende: eine Gerade im Originalbild (Menge von Bildkoordinaten $\{(x_0, y_0), \cdots, (x_n, y_n)\}$) ist im HR eine einzelne Koordinate (θ_i, d_i) und eine einzelne Koordinate (x_i, y_i) im Originalbild sind im HR eine Menge von Punkten $\{(\theta_0, d_0), \cdots, (\theta_m, d_m)\}$, welche sinusförmig angeordnet sind [ML99].

Erstellung des Hough-Raums

Abbildung 3.6 zeigt die Herangehensweise zur Erstellung des HR: Angenommen das entsprechende Kantenbild hat an genau drei Pixeln eine detektierte Kante, dann werden im Folgenden auch nur über diese drei Kantenpixel iteriert.
Betrachten wir nun Abbildung 3.6 a), hier entspricht die gewählte Auflösung der x-Achse im HR einem Winkel von $\Delta\theta_i = 30°$. Die Auflösung der y-Achse kann auch frei gewählt werden, in dem Beispiel, wird d auf eine Nachkommastelle genau gerundet. Es werden nun Schritt für Schritt für alle Winkel θ_i (in diesem Fall sechs verschiedene) und für alle jeweiligen Kantenkoordinaten (x_i, y_i) der Abstand d zum Lotfußpunkt berechnet, indem die Werte in die Formel 3.1 eingesetzt werden. Jeder der berechneten und nach gewünschter Auflösung gerundeten Wert d gehört zu einem spezifischen θ. Es entsteht so ein Treffer im HR an der Stelle (θ, d), wo dieser Treffer verzeichnet wird.
Im 2. Schritt (Abbildung 3.6 b)) wird nun für den nächsten Kantenpixel d für alle θ berechnet und dies wiederum als Treffer im HR eingetragen. Hier lässt sich bereits erkennen, dass für den Winkel $\theta = 60°$ nahezu derselbe Abstand d berechnet wurde und somit die Kombination $(\theta = 60°, d = 81)$ auf demselben Punkt im HR fällt, wenn d jeweils immer auf ganze Zahlen gerundet wird. Da an dieser Position bereits ein Treffer verzeichnet ist, wird der Wert einfach inkrementiert.
Auch im 3. Schritt (Abbildung 3.6 c)) tritt dieses Ereignis auf, sodass bereits ein drittes Mal an der Stelle $(\theta = 60°, d = 80)$ ein Treffer gelandet wurde. Auf diese Art und Weise entsteht nun der HR als eine Art 2D-Akkumulator.

Linienextrahierung

An den Stellen (θ, d) im HR, an denen die meisten Treffer erzielt wurden, sind auch die signifikanten Linien im Bild zu finden. Das bedeutet, die Suche nach den Linien im Originalbild, ist eine Suche nach den lokalen Maxima (auch Peaks genannt) im HR. Das Zurückrechnen der Linien von dem

[7] in engl. *Hough-Space*

HR aus dem Tupel (θ, d) erfolgt mit Hilfe der Gleichung:

$$P_0 = (P_{0_x}, P_{0_y})$$

$$\text{wobei } P_{0_x} = d \cdot \cos\theta \text{ und } P_{0_y} = d \cdot \sin\theta.$$

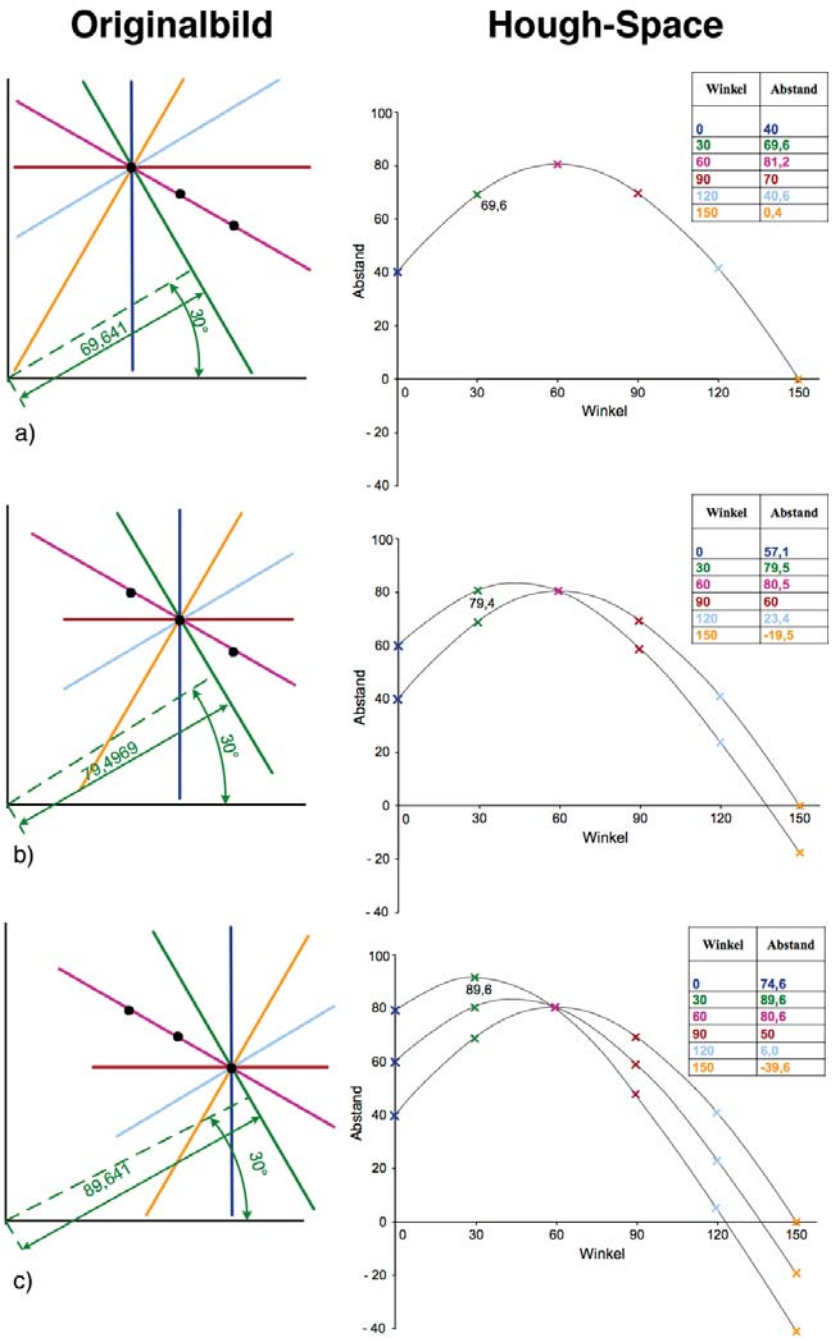


Abbildung 3.6: Prinzip der HT nach [Som09, S.14]

3.3.3 Berechnung paralleler Linien im Hough-Raum

Im vorherigen Abschnitt wurde der Algorithmus zur Erstellung eines Hough-Raumes erläutert, in dem stark ausgeprägte Linien im Bild wiedergefunden werden können. Da aber nicht alle Linien im Bild benötigt werden, sondern nur diejenigen, die senkrecht aufeinander stehen oder parallel zueinander verlaufen und eine Ebene aufspannen, müssen diese nun separat aus der Menge der gefundenen Linien herausgefiltert werden.
Im nächsten Schritt wird dazu der HR benutzt, um darin die für die Entzerrung notwendigen parallelen Liniengruppen zu finden, die eine Ebene aufspannen. Dazu wurde das Verfahren aus der Arbeit von Matessi et al. [ML99] benutzt.

Verfahren

Bei dem Verfahren von Matessi et. al. werden die Fluchtpunkte direkt im HR eines beliebigen Bildes detektiert. Fluchtpunkte sind fiktive Punkte innerhalb oder außerhalb des Bildes (abhängig von dem Grad der Verzerrung), die entstehen, wenn die im 3D Raum parallel verlaufenden Linien sich im 2D Raum in einem Punkt, dem Fluchtpunkt, schneiden. Mit dem Wissen, welche Linien sich in demselben Fluchtpunkt schneiden, liegt auch gleichzeitig die Information vor, welche Linien in der realen Welt parallel zueinander verlaufen.
Zur Vereinfachung des Entzerrungsproblems wird davon ausgegangen, dass sich in einem Fahrstuhlbild entweder immer genau zwei Fluchtpunkte im Endlichen befinden, zu denen genau zwei Paare von Linien gehören oder die Fluchtpunkte im Unendlichen liegen, sodass eine Entzerrung nicht mehr notwendig ist, da keine Verzerrung vorliegt. Die zwei Fluchtpunkte werden horizontaler und vertikaler Fluchtpunkt vp_h bzw. vp_v genannt und die dazugehörigen parallelen Linien, werden als horizontale bzw. vertikale Liniengruppen bezeichnet. Weiterhin wird davon ausgegangen, dass die horizontalen und vertikalen Liniengruppen senkrecht zueinander verlaufen und untereinander natürlich parallel sind.
Als Eingabe wird bei dem Verfahren ein HR benötigt, der wie im Abschnitt 3.3.2 erklärt, erstellt wurde. Da es gilt, immer genau zwei Fluchtpunkte mit den dazugehörigen Liniengruppen zu detektieren, sind die Menge an horizontaler und vertikaler Liniengruppen disjunkt und der HR kann in einen horizontalen und vertikalen Raum aufgeteilt werden:

$$HR_v : [-45°, +45°]$$

$$HR_h : [-90°, -45°[\,, \\ \quad]+45°, +90°]$$

Diese naive Aufteilung in zwei disjunkte HR vereinfacht das noch folgende Minimierungsproblem enorm, da so eine Vorauswahl der Linien getroffen wurde. Die Unterteilung stützt sich auf die Annahme, dass sich horizontale und vertikale Linien im HR des Bildes nur in den definierten Bereichen befinden. Im Umkehrschluss bedeutet dies, dass bei der Aufnahme des Bildes die Kamera nicht schräg gehalten werden darf, da es sonst passieren kann, dass eine eigentlich horizontale Linie als vertikale eingestuft wird. Dies verfälscht das Ergebnis zwar nicht, aber dadurch wird die entsprechende Linie nicht mehr weiter berücksichtigt und kann im schlimmsten Fall dazu führen, dass kein Ergebnis gefunden werden kann. Nun liegen im HR_v alle Tupel (θ, d) von dessen Linien im geometrischen Raum (dem Originalbild) zum einen angenommen wird, dass es sich um vertikale Linien handelt und zum anderen, dass sich ein Teil dieser auch in einem gemeinsamen vertikalen Schnittpunkt schneiden könnte. Dasselbe gilt analog für HR_h mit den horizontalen Linien, die sich in einem gemeinsamen horizontalen Schnittpunkt treffen könnten.

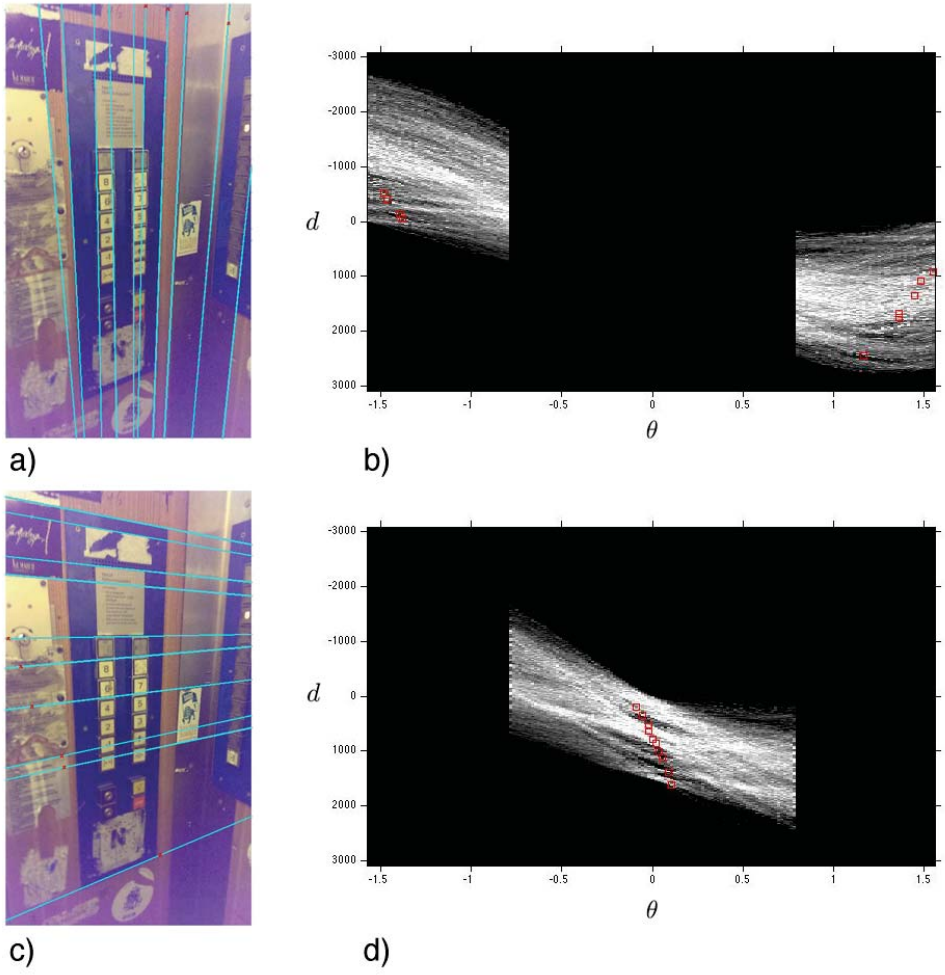


Abbildung 3.7: Zu sehen sind die vertikalen bzw. horizontalen Linien im Bild a) bzw. c) und im HR b) bzw. d)

Die Berechnung des Schnittpunktes für eine solche Menge an horizontalen bzw. vertikalen Linien erfolgt über die Formel nach [ML99] mit Hilfe der kleinsten Fehlerquadrate:

$$e^2 = \min_{x_0,y_0} \sum_{i=1}^{n} W_i(d_i - x_0 \cdot \cos\theta_i - y_0 \cdot \sin\theta_i)^2 \tag{3.2}$$

Hierbei ist n die Anzahl der benutzten Linien und W_i stellt eine Gewichtung dar, die im Experiment aber immer konstant gehalten wurde, da keine Linie präferiert werden soll. Der Vollständigkeit halber wird W_i aber in den Formeln mitgeführt. Durch partielle Ableitungen nach x und y folgen die Gleichungen:

$$\begin{aligned} \frac{de^2}{dx} &= \sum 2W_i(d_i - x_0 \cdot \cos\theta_i - y_0 \cdot \sin\theta_i)' \cdot (d_i - x_0 \cdot \cos\theta_i - y_0 \cdot \sin\theta_i) \\ &= \sum -2W_i \cos\theta_i \cdot (d_i - x_0 \cdot \cos\theta_i - y_0 \cdot \sin\theta_i) \end{aligned} \tag{3.3}$$

bzw.

$$\begin{aligned} \frac{de^2}{dy} &= \sum 2W_i(d_i - x_0 \cdot \cos\theta_i - y_0 \cdot \sin\theta_i)' \cdot (d_i - x_0 \cdot \cos\theta_i - y_0 \cdot \sin\theta_i) \\ &= \sum -2W_i \sin\theta_i \cdot (d_i - x_0 \cdot \cos\theta_i - y_0 \cdot \sin\theta_i). \end{aligned} \tag{3.4}$$

Durch Ausmultiplizieren und Substituierung der Formel 3.3 und 3.4 mit $\cos\theta_i$ durch a_i und $\sin\theta_i$ durch b_i ergibt sich:

$$\begin{aligned} \sum 2W_i a_i^2 x_0 + 2W_i a_i b_i y_0 - 2W_i a_i d_i &= 0 \\ \sum 2W_i b_i a_i x_0 + 2W_i b_i^2 y_0 - 2W_i b_i d_i &= 0. \end{aligned}$$

Eine weitere Substituierung mit

$$\begin{aligned} \mathcal{A} &= 2\sum W_i a_i^2 \\ \mathcal{B} &= 2\sum W_i b_i^2 \\ \mathcal{C} &= 2\sum W_i a_i b_i \\ \mathcal{D} &= 2\sum W_i a_i d_i \\ \mathcal{E} &= 2\sum W_i b_i d_i, \end{aligned}$$

ergibt die übersichtliche Formel:

$$\begin{aligned} x_0 &= \tfrac{\mathcal{D}-\mathcal{C}y_0}{\mathcal{A}} \\ y_0 &= \tfrac{\mathcal{E}-\mathcal{C}x_0}{\mathcal{B}} \end{aligned} \quad . \tag{3.5}$$

Damit können nun bei einer unbestimmten Anzahl an horizontaler oder vertikaler Linien die Schnittpunkte berechnet werden. Aufgrund von Rundungen und Messungenauigkeiten bei der HT werden sich mehr als zwei Linien nie in einem Punkt treffen. Mit der Formulierung als Minimierungsproblem in Gleichung 3.2, kann allerdings der Schnittpunkt gut angenähert werden. Mit dem Prinzip der kleinsten Fehlerquadrate [MH04, S. 111ff.] wird daher derjenige Punkt aus einer beliebigen Menge an Linien berechnet, der als Schnittpunkt aller Linien zueinander die minimale Abweichung besitzt. Je besser der Schnittpunkt ist, desto eher wird dieser als Fluchtpunkt angesehen und die entsprechnenden horizontalen oder vertikalen Linien werden daher als parallel

eingestuft.
Die Qualität des Schnittpunktes (x_1, y_1), der mit Hilfe von Gleichung 3.5 und den Linien $\{(\theta_1, d_1), \cdots, (\theta_n, d_n)\}$ berechnet wurde, kann nun festgestellt werden. Dazu wird für alle Geraden mit dem Winkel θ_1 bis θ_n und dem Schnittpunkt (x_0, y_0) die Geraden $\{(\theta_1, dNeu_1), \cdots, (\theta_n, dNeu_n)\}$ über die ursprüngliche Formel 3.1 berechnet. Je besser der approximierte Schnittpunkt der n Geraden war, desto weniger weichen die Abstände d_i und $dNeu_i$ auch voneinander ab. Das Verfahren arbeitet so ähnlich wie der RANSAC[8]-Algorithmus [FB81]: Es werden n Linien zufällig aus dem Set herausgenommen und geschnitten. Erfüllen die Linien die vorgegebenen Richtlinien (in dem Fall $|d_i - dNeu_i| < d_{Thr}$, wobei durch Experimente $d_{Thr} = 20$ festgestellt werden konnte) handelt es sich um eine gute Näherung ohne Ausreißern, die verwendet werden kann.
Die zufällig gewählte Anzahl n der zu schneidenden Linien, für die der Schnittpunkt berechnet wird, ist entscheidend für die Qualität der berechneten Fluchtpunkte und die Aussage über die Parallelität der Linien. Im Experiment werden immer die zehn prägnantesten horizontalen und prägnantesten vertikalen Linien aus dem HR extrahiert und aus diesen dann jeweils nach möglichen parallelen Linien gesucht.

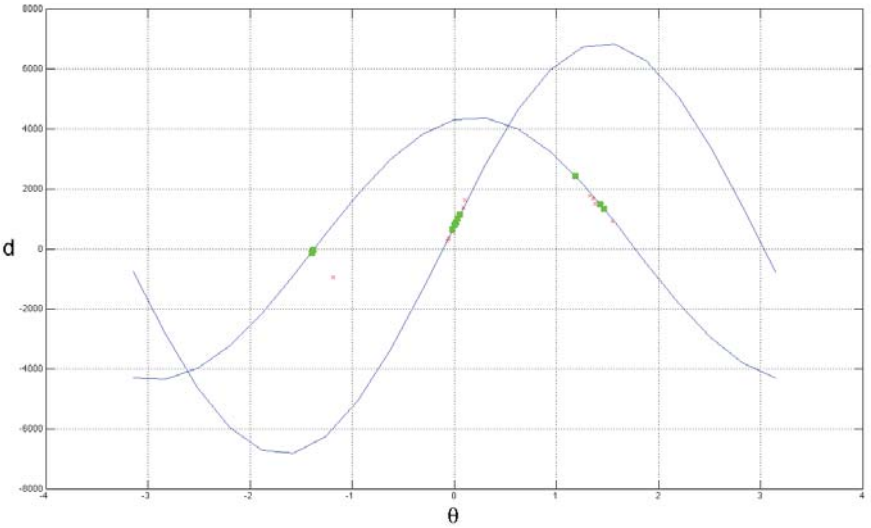


Abbildung 3.8: Beste gefundene Sinus-Kurven der jeweils horizontalen bzw. vertikalen Linien aus Abbildung 3.7; grüne Punkte stellen hier die verwendeten (θ, d)-Tupel für eine gute Näherung dar, rote Punkte dagegen entfallen; die blauen Näherungskurven entsprechen den Fluchtpunkten vp_h bzw. vp_v

Es hat sich weiterhin herausgestellt, dass der Schnittpunkt gut berechnet werden kann, wenn $n = 4$, d.h. wenn immer vier der jeweils zehn prägnantesten Linien aus dem HR miteinander geschnitten werden. Bei $n = 2$ würde es immer einen exakten Schnittpunkt geben, der nicht zwangsweise eine Fluchtpunkt darstellt. Drei Linien ($n = 3$) geschnitten, haben zwar nie einen exakten Schnittpunkt, oftmals aber zufällig einen relativ guten, obwohl sie auch nicht auf einen Fluchtpunkt zulaufen. Bei fünf oder mehr Linien $n \geq 5$ miteinander geschnitten, ist die Gefahr groß, dass auch Linien enhalten sind die nicht zum gleichen Fluchtpunkt gehören und somit keine gute Approximation liefern, sodass diese Gruppe von Linien verworfen werden würde.
Bei der Wahl von $n = 4$ ergibt sich eine mögliche Anzahl von $2 \cdot \binom{10}{4} = 420$ Linienkombinationen, die sequentiell geprüft werden, wobei die beste Lösung ausgegeben wird.

[8] RANSAC: engl. random sample consensus

Ein Fluchtpunkt wird auch erst dann als korrekt angesehen, wenn er sich außerhalb des Bildes befindet. Der Grund hierfür liegt zum einen darin, dass eine nicht so starke projektive Verzerrung angenommen wird und der Fluchtpunkt außerhalb des Bildausschnittes liegen muss. Je stärker nämlich die Verzerrung, desto wahrscheinlicher liegt ein Fluchtpunkt innerhalb des Bildes. Zum anderen kann es bei der HT vorkommen, dass Pseudo-Linien auftreten, die wegen zu schlechten Kantenbildern entstehen. Dadurch verlaufen Linien mehrfach durch dasselbe Objekt und es entstehen Schnittpunkte innerhalb des Bildes, die ausgeschlossen werden müssen, da sie in diesem Falle keine Fluchtpunkte darstellen. Abbildung 3.8 zeigt ein konkretes Beispiel.

3.3.4 Berechnung der Homographie

Im vorherigen Schritt wurden die horizontalen und vertikalen Liniengruppen gefunden, die den besten gemeinsamen Schnittpunkt besitzen und in der realen Welt parallel verlaufen. Im nächsten Schritt werden die Eckpunkte der Ebene gesucht, auf denen sich die Bedienelemente befinden. Mit ihnen wird dann eine Entzerrungsmatrix (Homographie) berechnet, mit der das Bild entzerrt werden kann.
Eine Homographie [HZ04, S.33] ist eine 3×3 Matrix H mit neun Elementen:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}.$$

Mit einer solchen Homographie, können perspektivische Projektionen durchgeführt werden, beispielsweise die projektive Transformation eines Vektors $\vec{x}$ in den Vektor $\vec{x}'$:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = H \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}. \tag{3.6}$$

Gesucht ist daher eine Homographie H, die die Gleichung 3.6 erfüllt und jeden Punkt x_i auf den Punkt x'_i abbildet. Um so eine Homographie berechnen zu können, werden mindestens vier Korrespondenzpunkte benötigt [HZ04, S.35], die aus den zuvor bestimmten horizontalen und vertikalen Liniengruppen entnommen werden können.
Wie zuvor erwähnt, wurde eine gewisse Menge an horizontalen Liniengruppen $L_{hori} = \{(\theta_1, d_1), (\theta_2, d_2), \cdots, (\theta_n, d_n)\}$ und vertikalen Liniengruppen $L_{verti} = \{(\theta_1, d_1), (\theta_2, d_2), \cdots, (\theta_m, d_m)\}$ gefunden, wobei $m, n \geq 4$. Es werden nun aus der Menge L_{hori} jeweils die oberste und unterste horizontale Linie und aus der Menge L_{verti} die äußere linke und äußere rechte vertikale Linie extrahiert. Die äußeren Linien der einen Menge werden nun paarweise mit den Linien der anderen Menge geschnitten, sodass genau vier Eckpunkte A, B, C, D der Ebene entstehen. Die Berechnung dieser Schnittpunkte erfolgt wiederum mit Hilfe von Gleichung 3.2.
Die Korrespondenzpunkte A', B', C', D' für jeden Eckpunkt ist ein Punkt eines einfachen Rechtecks. Daher ist die Berechnung der Homographie H die Lösung des folgenden Gleichungssystems:

$$A' = \begin{pmatrix} A_x \\ A_y \\ 1 \end{pmatrix} \cdot H$$

$$B' = \begin{pmatrix} B_x \\ B_y \\ 1 \end{pmatrix} \cdot H$$

$$C' = \begin{pmatrix} C_x \\ C_y \\ 1 \end{pmatrix} \cdot H$$

$$D' = \begin{pmatrix} D_x \\ D_y \\ 1 \end{pmatrix} \cdot H.$$

Die korrespondierenden Referenzpunkte wurden wiefolgt berechnet:

$$A' = \begin{pmatrix} B_x - \overline{AB} \\ B_y \\ 1 \end{pmatrix}$$

$$B' = B$$

$$C' = \begin{pmatrix} B_x \\ B_x + \overline{BC} \\ 1 \end{pmatrix}$$

$$D' = \begin{pmatrix} A_x \\ C_y \\ 1 \end{pmatrix}.$$

Das Mapping basiert auf der Annahme, dass sich alle Punkte auf den Eckpunkten eines Rechtecks befinden ausgehend von dem Punkt B (siehe Abbildung 3.9).

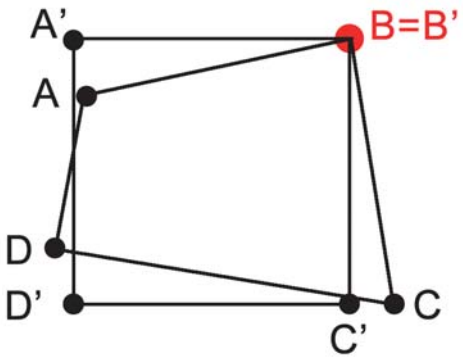


Abbildung 3.9: Mapping der einzelnen Eckpunkte der verzerrten Ebene auf das Rechteck

Auf diese Weise lässt sich die Homographie H zur perspektivischen Entzerrung des Bildes berechnen. Die Homographie wird nun für jeden Bildpunkt des Originalbildes einzeln angewandt und der Pixelwert an der resultierenden neuen Position eingetragen, sodass das gesamte Bild entzerrt wird.
Bei den Eck- und Korrespondenzpunkten handelt es sich um homogene Koordinaten [HZ04, S.2]. Zur Darstellung der homogenen Koordinaten im zweidimensionalen Raum, wird die homogene Komponente w wieder entfernt, indem durch sie geteilt wird: $(wx, wy, w)^T \rightarrow (x, y)^T$.
Umgekehrt können die Koordinaten im entzerrten Bild natürlich auch wieder auf das Originalbild zurückgerechnet werden, indem einfach mit der inversen Homographie multipliziert wird:

$$\vec{x} = \vec{x}' \cdot H^{-1}.$$

Das Bild wurde entzerrt und die Suche nach den eigentlichen Bedienelementen kann nun beginnen.

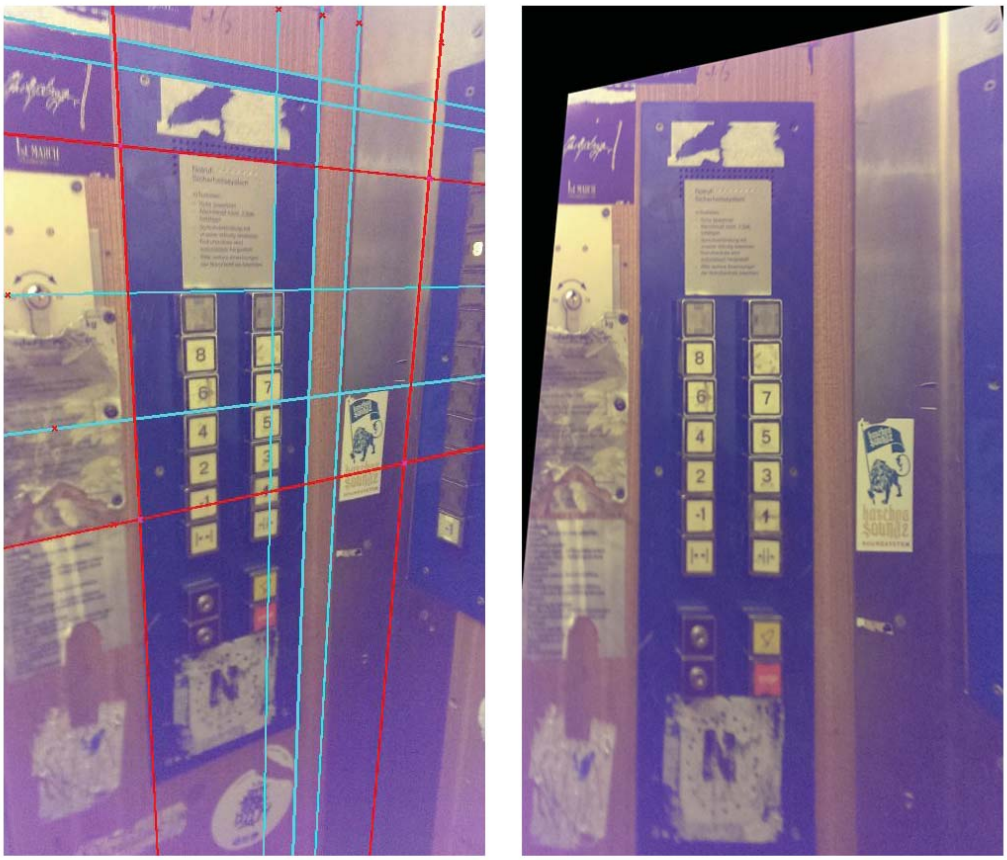


Abbildung 3.10: a) Berechnete parallele horizontale und vertikale Liniengruppen (cyan), spannen eine planare Ebene auf (rot) deren Eckpunkte (magenta) für die Rektifizierung benutzt werden; es ist zu beachten, dass im Vergleich zur Abbildung 3.7 nur vier bzw. sechs der ursprünglichen vertikalen bzw. horizontalen Linien als parallele Linien identifiziert wurden, b) dazugehöriges rektifiziertes Bild

3.4 Stufe 2: Detektion der Bedienelemente

In der ersten Stufe (Abschnitt 3.3) wurde, sofern möglich, das Bild rektifiziert oder bei zu geringer perspektivischer Verzerrung in Originalform beibehalten. In der zweiten Stufe erfolgt nun die Hauptaufgabe, die Detektion der Buttons.
Im Wesentlichen wurden hier die Vorüberlegungen aus Abschnitt 3.1 angewandt. Besonders Eigenschaft 3 und 4 sind hierbei entscheidend: Es werden in dem vorverarbeiteten Fahrstuhlbild nach einfachen Formen wie Rechtecke und Kreise gesucht, was in Abschnitt 3.4.1 erklärt wird. Diese detektierten Formen werden jeweils in einem separaten Graphen als Datenstruktur gespeichert. Da die Fahrstuhlbedienelemente untereinander nicht von der Form abweichen, erfolgt eine Suche nach der Form, die am häufigsten auftritt und bei denen die einzelnen Objekte derselben Form dicht beieinander liegen, sowie etwa dieselbe Größe bzw. Fläche besitzen. Dies wird über Graphenoperationen ermöglicht und im Abschnitt 3.4.2 genauer erläutert.

3.4.1 Initiale Detektion der Formen

Wie bereits erwähnt, ist der erste Schritt die Bestimmung einfacher Formen im Bild. Dazu wurde in dieser Arbeit der Algorithmus von Suzuki et. al. [SS85] benutzt. Das Verfahren detektiert Umrisse in einem Kantenbild, die auch als Konturen, Grenzen oder Koordinatenketten bezeichnet werden und kann diese in topologischen und hierarchischen Strukturen gruppieren.

Verfahren zur Konturenbestimmung

Der Algorithmus von Suzuki et. al. [SS85] erstellt von einem binären Kantenbild eine Sequenz von Koordinaten, auch Kettencode genannt, der Umrisse zwischen den jeweiligen verbundenen Objekten, die sich zwischen dem Bildhintergrund und den Löchern[9] im Bild befinden. Der Algorithmus extrahiert also die topologische Struktur des Bildes, wobei zwischen zwei Arten von Grenzen unterschieden werden kann: die äußeren Grenzen und die Grenzen zu den Löchern.
Das Verfahren hat den Vorteil, dass jeder Umriss bzw. jede Koordinatenkette einen eindeutigen Bezeichner bekommt, sodass die einzelnen Grenzen voneinander unterschieden werden können (*border labeling*). Als zweite Erweiterung wurde außerdem eine Prozedur entwickelt, die es ermöglicht zu jeder aktuellen Kontur die übergeordnete Kontur, auch Eltern-Kontur genannt, herauszufinden. Auf diese Weise können die Beziehungen umliegender Konturen zueinander bestimmt werden und hierarchische Strukturen erstellt werden.
Der erste Schritt zur Konturenbestimmung im Bild ist das Einlesen eines Binärbildes (dem Kantenbild, siehe Abschnitt 3.3.1). Es wird nun pixelweise über das Kantenbild iteriert und unterbrochen, wenn ein Pixel an der Stelle (i, j) gefunden wurde, der die Bedingung a) erfüllt und eine äußere Grenze ist oder Bedingung b) erfüllt und eine Loch-Grenze darstellt.

Bedingungen:

a)

	$j-1$	j
i	0	1

b)

	j	$j+1$
i	≥ 1	0

[9] Bereiche die von Kanten umringt sind, selbst aber keine Kantenpixel darstellen

Werden beide Bedingungen a) und b) gleichzeitig erfüllt, so handelt es sich um den Startpunkt für die äußere Kontur. Ausgehend vom gefundenen Startpixel werden nun den Kantenpixeln entlang der Grenze gefolgt, wodurch die Kontur entsteht. Nachdem die aktuelle Kontur vollständig gefunden wurde, wird mit den verbliebenen Pixeln gleichermaßen fortgefahren und bei dem letzten Pixel (im Bild rechts unten), gestoppt. Für eine detailliertere Beschreibung des Verfahrens wird auf die beschriebene Veröffentlichung [SS85] verwiesen, in der es auch ein Schritt-für-Schritt Beispiel gibt.
Als Vorteil des Algorithmus wird von den Autoren selbst genannt, dass das Verfahren in der Lage ist, Umgebungseigenschaften entlang der verbundenen Komponenten im binären Bild zu bestimmen. Deshalb stellt das Verfahren auch ein mächtiges Werkzeug zur Formerkennung und Merkmalsextrahierung dar.

Verfahren zur Konturenvereinfachung

Im vorherigen Abschnitt wurden die Hülleninformationen (Konturen) des Kantenbildes als gruppierte Listen von Pixeln berechnet. Der nächste Schritt ist, den detektierten Konturen im Kantenbild eine konkrete geometrische Form zuzuordnen. Dazu muss allerdings vorher die Anzahl der Pixel pro Kontur auf eine Minimum reduziert werden.
Die Vereinfachung der Konturen erfolgt anhand der Vorüberlegungen in Abschnitt 3.1 auf die Basisformen Kreis und Viereck. Zur Minimierung der Konturpixel wird der Ramer-Douglas-Peucker-Algorithmus[10] [Ram72, DP73] verwendet, der im Folgenden kurz beschrieben werden soll. Die Zielsetzung des Algorithmus ist es, eine beliebige zweidimensionale Kurve bestehend aus einer Menge an Punkten (der Koordinatenkette) so weit zu approximieren, dass nur noch eine kleine aber nicht notwenigerweise minimale Anzahl an Punkten übrig bleibt, sodass die ursprüngliche Form weiterhin erkennbar bleibt. Im konkreten Fall sind das die geometrischen Formen Kreis und Viereck.
Der Algorithmus, der auch wie in [Dut99] zur Vereinfachung von geographischen Karten verwendet wird, arbeitet nach dem Divide and Conquer Prinzip. Dabei wird der Eingabevektor, der in diesem Falle die Konturen (Koordinatenketten) sind, an einer bestimmten Stelle immer wieder aufgeteilt und rekursiv aufgerufen. Zunächst wird als erster Iterationsschritt die gröbste Approximation der Strecke vom ersten zum letzten Punkt $\overline{P_1 P_n}$ genommen und geprüft, ob diese Näherung bereits ausreichend ist. Dazu wird in allen Punkten, die zwischen dem ersten und letzten Punkt liegen, derjenige Punkt P_i gesucht, dessen senkrechter Abstand zur Geraden $\overline{P_1 P_n}$ am größten ist und den Schwellwert ϵ nicht unterschreitet. Der Punkt P_i ist demnach für die Gestalt der Koordinatenkette zu wichtig, um vernachlässigt zu werden und teilt nun die Strecke $\overline{P_1 P_n}$ in zwei weitere Abschnitte. Mit den so neu entstandenen zwei Teilen links und rechts von P_i wird nun genauso fortgefahren und es wird jeweils wieder nach dem Punkt gesucht, der den größten Abstand zur neuen Geraden besitzt. Abgebrochen und somit in keine weiteren Teilabschnitte mehr untergliedert wird erst dann, wenn entweder der maximale Abstand ϵ unterschritten wird oder es gilt $n = 2$, wobei n die Anzahl der im jeweiligen Teilabschnitt enthaltenen Punkte darstellt. Ist das Abbruchkriterium nicht erfüllt, wird die Näherung wie beschrieben auf die Teilabschnitte angewendet und rekursiv fortgeführt.
Auf diese Weise kann die Anzahl der ursprünglich gefundenen Konturpunkte (der Koordinatenkette) stark reduziert werden.
Bei Experimenten konnte herausgefunden werden, dass bei $\epsilon = 0.02 \cdot u$ (mit u als Umfang der ursprünglichen Kontur) die Konturen soweit vereinfacht werden können, dass zwischen den verschiedenen geometrischen Formen (siehe nächster Abschnitt) unterschieden werden kann und keine zu starke Vereinfachung auf andere Formen gemacht wird.

[10] 1972 von Urs Ramer und unabhängig davon 1973 von David Douglas und Thomas Peucker entworfen

Formbestimmung

Nachdem die Konturen auf ein Minimum reduziert wurden, können diese nun zu Formen zugeordnet werden. Die Zuordnung zu den geometrischen Formen kann einfach über die Anzahl n der approximierten Punkte erfolgen: Ist $n = 3$ handelt es sich um ein Dreieck, bei $n = 4$ um ein Viereck, bei $n = 5$ um ein Pentagon, usw.
Bei n>6, kann es sich womöglich um einen Kreis handeln, was allerdings nochmal geprüft werden muss. In diesem Fall wird um den möglichen Kreis, der jetzt nur noch aus einer geringen Anzahl an Konturpunkten besteht, eine Bounding Box[11] gelegt, die eine bestimmte Höhe h und Breite w besitzt. Um als Kreis erkannt zu werden, muss zunächst gelten, dass h und w um weniger als ein bestimmter Prozentwert voneinander abweichen. Die zweite Bedingung ist ähnlich zur ersten: Die Fläche der um den möglichen Kreis bestimmten Kontur und die Kreisfläche $\pi \cdot \frac{r^2}{2}$ (wobei angenommen wird, dass der Radius $r = \frac{(w+h)}{4}$ beträgt) dürfen auch nicht um mehr als dem Prozentwert voneinander abweichen. Im konkreten Fall wurden 30% verwendet, um auch noch Ellipsen als Kreise detektieren zu können.
Es ist also möglich nach der approximierten Gestalt zu unterscheiden, wobei in diesem Falle nur nach Vierecken und Kreisen gesucht wird, da alle anderen Formen durch die Grundannahmen aus Abschnitt 3.1.1 ausgeschlossen wurden.

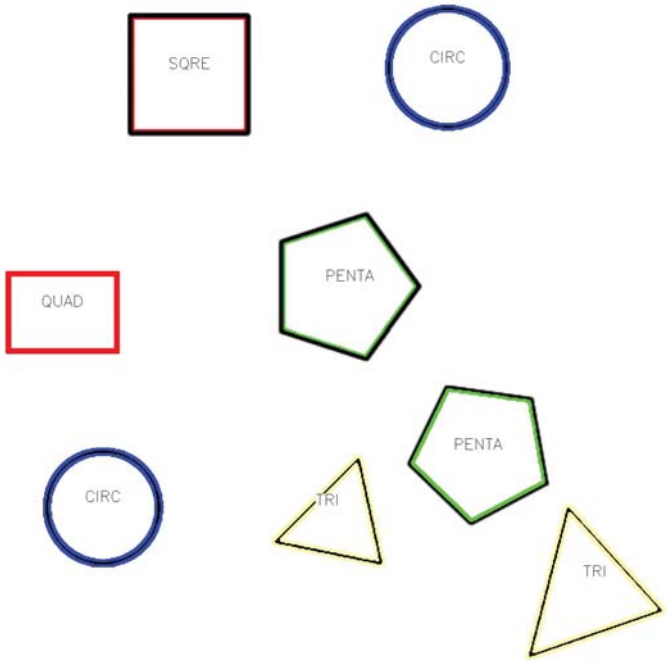


Abbildung 3.11: Mit Hilfe der in 3.4 beschriebenen Konturberechnung konnten die jeweiligen Objekte im Beispielbild erkannt und durch die Konturvereinfachung den jeweiligen geometrischen Formen korrekt zugeordnet werden.

[11] Hüllkörper

Alle im rektifizierten Bild gefundenen Vierecke werden untersucht und geprüft, ob es sich um Rechtecke oder die Spezialform Quadrate handelt. Weil bei einer solchen Bilderkennungsaufgabe die geometrischen Eigenschaften (wie Rechtwinkligkeit, gleiche Seitenlängen) trotz Rektifizierung nicht vollständig wiederhergestellt werden können, wird eine gewisse Toleranz eingeführt, um die Objekte, besonders die Buttons, trotzdem zuordnen zu können.
Jedes gefundene Viereck wird daher geprüft, ob die gegenüberliegenden Seitenlängen a, c und b, d um weniger als 10% von den Mittelwerten $m_1 = \frac{a+c}{2}$ und $m_2 = \frac{b+d}{2}$ voneinander abweichen, d.h. es muss gelten:

$$\{a, c\} > m_1 - thr_1 \wedge \{a, c\} < m_1 + thr_1 \quad \text{mit } thr_1 = 0.1 \cdot \frac{(a+c)}{2} \tag{3.7}$$

bzw.

$$\{b, d\} > m_2 - thr_2 \wedge \{b, d\} < m_2 + thr_2 \quad \text{mit } thr_2 = 0.1 \cdot \frac{(b+d)}{2}. \tag{3.8}$$

Sind die Bedingungen erfüllt, handelt es sich um ein Rechteck und es kann weiter geprüft werden, ob die Bedingung 3.7 oder 3.8 erweitert auf alle Kanten gleichermaßen gilt. Ist auch diese Bedingung erfüllt, handelt es sich um ein Quadrat.

3.4.2 Zusammenfassen in einen Graphen

Zuvor wurde gezeigt, wie in einem Kantenbild die jeweiligen Objekte den geometrischen Formen zugeordnet werden können. Als nächstes müssen aus den detektierten Formen diejenigen extrahiert werden, bei denen es sich aller Wahrscheinlichkeit nach um Fahrstuhlbedienelemente handelt.
Dabei ist es besonders hilfreich einen Graphen zu erstellen, da dort effektiv nach Häufungen gesucht werden kann. Die Bestimmung der möglichen Bereiche, wo sich Buttons befinden können, hängt also davon ab, wo ähnliche geometrische Formen vermehrt auftreten. Dabei wird sich auf Rechtecke und Kreise beschränkt.
Es werden nun für die beiden mögliche Button-Formen Kreis und Rechteck ein einfacher ungerichteter Graph $G = (N, E)$ erstellt, die im Folgenden mit G_{circ} und G_{quad} bezeichnet werden, wobei N die Menge der Knoten und E die Menge der Kanten bezeichnet. Ein bestimmter Knoten N_i stellt in diesem Fall den Massenschwerpunkt c_i gespeichert als Bildkoordinate (x_i, y_i) der jeweiligen geometrischen Form dar, welcher zuvor berechnet werden muss, bevor N_i zu $G_{circ/quad}$ hinzugefügt werden kann. Die Berechnung des Massenschwerpunktes erfolgen mit Hilfe einer einfachen arithmetischen Mittelwertsberechnung jeweils über alle im vorherigen Schritt vereinfachten Punktmengen:

$$\frac{1}{n} \sum_{i=1}^{n} (x_i, y_i).$$

Nachdem alle Knoten E auf diese Weise zum jeweiligen Graph G_{circ} bzw. G_{quad} hinzugefügt wurden, werden nun alle Kanten E berechnet. Zwei Knoten N_i und N_j sind über die Kante E_{ij} verbunden, wenn der euklidische Abstand d_{ij} zwischen den dazugehörigen Massenschwerpunkten $c_i = (x_i, y_i)$ und $c_j = (x_j, y_j)$ ein bestimmten Schwellwert nicht überschreitet. Dieser Wert wurde aus dem durchschnittlichen Maximalabstand benachbarter Knöpfe von einer ausgewählten Menge an Bildern berechnet und in den Experimenten auf einen relativen Wert von 18% der Bilddiagonale gesetzt. Der euklidische Abstand d_{ij} wird als Gewicht der Kante E_{ij} mit gespeichert.
Da in einem Graphen alle Objekte derselben Form detektiert und gespeichert wurden, sind auch

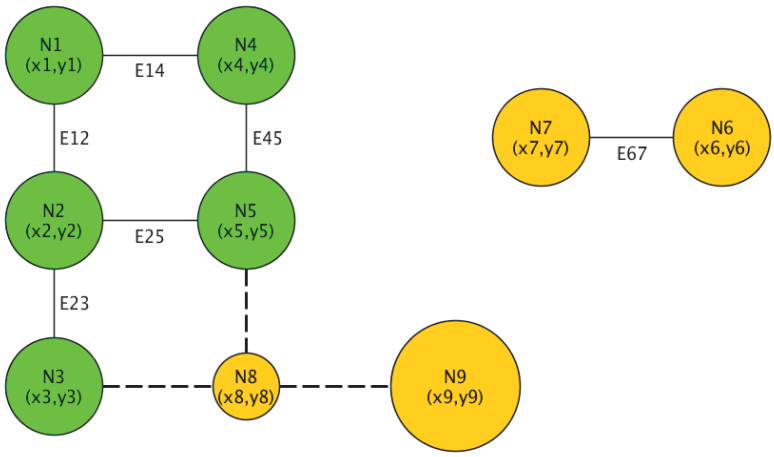


Abbildung 3.12: Die Abbildung soll die im Abschnitt 3.4.2 beschriebene Erstellung eines Graphen beschreiben. Die Knoten N_1 bis N_5 sowie N_6 und N_7 sind miteinander verbunden, da die Abstände den Schwellwert nicht überschreiten. N_8 und N_9 überschreiten den Abstand zwar auch nicht, werden aber nicht in die jeweiligen Graphen mit aufgenommen, da die Fäche zu groß bzw. zu klein ist. Die grünen Knoten stellen hier den am stärksten verbundenen Subgraphen $G_{circBts}$ dar.

folglich unterschiedlich große Elemente enthalten. Da aber wie in den gemeinsamen Eigenschaften, in Abschnitt 3.1.1, dargelegt, sich weder die Form noch die Größe der Buttons untereinander ändert, muss weiterhin gelten, dass zwischen zwei Knoten N_i und N_j nur dann eine Kante E_{ij} hinzugefügt werden kann, wenn zusätzlich zu dem Abstand die Flächen beider Objekte zueinander nicht zu stark von einander abweichen. Sind beide Bedingungen erfüllt, kann die Kante hinzugefügt werden und es wird mit allen verbleibenden Knoten analog fortgefahren.
Da grundsätzlich davon ausgegangen wird, dass die Form der Buttons untereinander nicht variieren, sind die tatsächlichen Buttons in nur einem der beiden Graphen G_{circ} oder G_{quad} enthalten. Es wird nun nach dem Subgraphen mit den meisten Knoten gesucht. D.h. es wird nach der größten Menge an Knoten im Graphen gesucht, die untereinander über die zuvor berechneten Kanten miteinander verbunden sind. Enthält die so entstandene Menge $N_{connectedCirc}$ mehr verbundene Knoten, als die Menge $N_{connectedQuad}$, wurden im Bild kreisförmige Buttons detektiert oder umgekehrt. Der so neu entstandene Subgraph entweder $G_{circBts}$ oder $G_{quadBts}$ enthält nun die initial detektierten Buttons. Abbildung 3.12 stellt das Vorgehen noch einmal graphisch dar.

3.4.3 Morphologische Operationen zur Kantenverbesserung

Es kann passieren, dass zwar die Rektifizierung des Bildes erfolgreich durchgeführt wurde, aber trotzdem die im Abschnitt 3.4.1 erklärte initiale Detektion keine Ergebnisse für Buttons liefert. Der Grund dafür liegt meist in dem Kantenbild. Die topologische Struktur der Bildobjekte, kann nur dann korrekt erkannt werden, wenn die Hülle geschlossen ist, d.h. im Kantenbild dürfen die Umrisse der möglichen Bedienelemente keine Unterbrechungen haben. Da dies abhängig ist von der Qualität des Kantenbildes, müssen die Kanten ggf. erweitert werden, wenn initial keine Konturen erkannt werden konnten.

Dilatation

Eine Möglichkeit dazu ist die Dilatation[12]. Dieser lokale Operator erweitert die Kanten, indem er zu den Bereichsrändern, je nach Größe des Operators, eine bestimmte Anzahl an Pixeln hinzufügt. Ist die Maske beispielsweise 3×3 Pixel groß, wird der Pixel (x, y) im neuen Bild I_{neu} genau dann auf eins gesetzt, wenn in der 3×3 Umgebung irgend einer der Bildpunkte den Wert eins besitzt, d.h.

$$I_{neu}(x,y) = \begin{cases} 1, \text{ wenn } \sum_{i=-1,j=-1}^{i=1,j=1} I_{alt}(x+i, y+j) \geq 1 \\ 0, \text{ sonst.} \end{cases} \tag{3.9}$$

Abbildung 3.13 zeigt das Ergebnis bei der Anwendung eines Dilatations-Operators. Die Größe des Dilatationsmaske ist ein möglicher Parameter, der die Qualität der Erkennung beeinflusst. Deshalb wird dieser in der Evaluation im Kapitel 4 weiter untersucht.

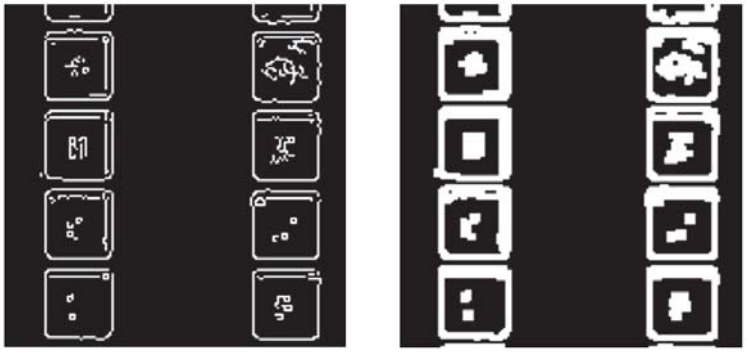


Abbildung 3.13: a) Ausschnitt aus Original-Kantenbild b) Bild nach angewendetem 3×3 Dilatationsoperator

[12] Verdickungsoperator

Schließen

Ein weiterer morphologischer Operator stellt das Schließen (engl. Closing) dar. Dabei wird auf das betreffende Kantenbild zuerst eine Dilatation und anschließend eine Erosion durchgeführt [MH04]. Ist die Erosionsmaske beispielsweise 3×3 Pixel groß, wird der Pixel (x, y) im neuen Bild I_{neu} nur genau dann auf eins gesetzt, wenn alle in der 3×3 Umgebung befindlichen Bildpunkte den Wert eins besitzen, d.h.

$$I_{neu}(x,y) = \begin{cases} 1, \text{ wenn } \sum_{i=-1,j=-1}^{i=1,j=1} I_{alt}(x+i, y+j) = 9 \\ 0, \text{ sonst.} \end{cases} \tag{3.10}$$

Anders als bei der Dilatation verdicken sich die Kanten bei dem Schließen nicht gleichmäßig.

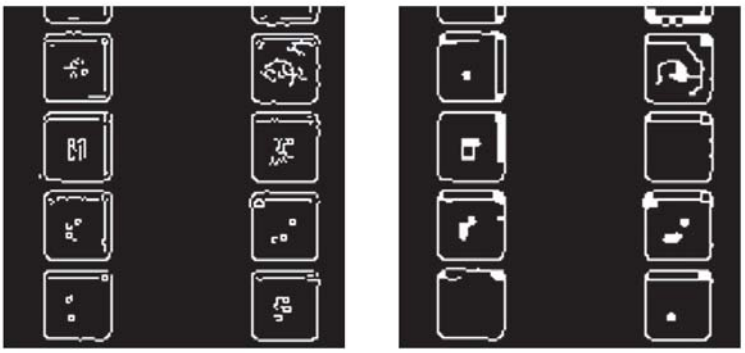


Abbildung 3.14: a) Ausschnitt aus Original-Kantenbild b) Bild nach angewendetem Schließen mit einer 3×3 Maske

3.5 Stufe 3: Template-Matching

In der zweiten Stufe wurde erklärt, wie die Buttons initial im Bild detektiert werden können. Diese Methode funktioniert relativ akkurat, solange das jeweilige Kantenbild gut genug ist, d.h. die Kanten der Buttons müssen deutlich aus dem Bild hervorgehen, was aber nicht immer bei allen Bedienelementen durch die in 3.1 beschriebenen Probleme gelingt. Da die Hauptaufgabe darin besteht, so viele Buttons wie möglich zu finden, wird versucht, den Rest des Bildes mit Hilfe von Template-Matching[13] (TM) auf weitere übersehene Buttons zu überprüfen. Als Template dienen die im vorherigen Schritt gefundenen initialen Buttons. Da zu einem Bedienelement keine allgemeinen Annahmen außer der geometrischen Form gemacht werden können, kann kein generelles Template erstellt werden, sodass die Stufe zwei (Abschnitt 3.4) in jedem Fall ein Ergebnis liefern muss, bevor das TM durchgeführt werden kann.
Die aus Abschnitt 3.4.2 erläuterte Graphenstruktur wird hier beibehalten und der Graph wird um die neuen Buttons erweitert.

3.5.1 Verfahren

Jeder initiale Button, der im zuvor berechneten Subgraphen $G_{circBtn}$ oder $G_{quadBtn}$ als Knoten N_i auftaucht, wird einmal in seiner Originalform (im RGB[14]-Bild) und einmal als Kantenbild (in Binärform, Abschnitt 3.3.1) auf dem Fahrstuhlbild abgeglichen. Ein Button wird nur dann als zulässiger Match im Bild wiedererkannt, wenn beide Matching-Methoden in demselben Bereich eine Übereinstimmung haben. Das doppelte Matching soll verhindern, dass die Buttons wegen zu hoher Übereinstimmung in den Farbwerten fälschlicherweise in bestimmten Bereichen des Bildes wiedergefunden werden (siehe Abbildung 3.15).
Das Prinzip des TMs ist immer dasselbe: Es wird eine Schablone definiert (Template), mit der im gesamten Bild nach Übereinstimmtungen (Matches) gesucht wird. Das Template selbst ist immer ein rechteckiger Ausschnitt aus dem Originalbild bestimmt durch die initialen approximierten Knoten der jeweiligen Form aus Stufe 2. Dabei wird jedes Template um einen Offset von 5% in jede Richtung erweitert, um auch den unmittelbaren Rand mit einzubeziehen. Die Größe der Suchmaske (Template) ist demnach variabel und abhängig von der initial detektierten Knopfgröße. Es werden wie bereits angedeutet pro Button immer zwei Templates erstellt: Ein binäres Template, das nur die Buttonkanten enthält und ein farbiges Template.

Zum TM im RGB-Farbraum wird nun die folgende Formel benutzt:

$$R_{colored}(x,y) = \frac{\sum\limits_{x',y'} (T_c(x',y') - I_c(x+x',y+y'))^2}{\sqrt{\sum\limits_{x',y'} T_c(x',y')^2 \cdot \sum\limits_{x',y'} I_c(x+x',y+y')^2}} \tag{3.11}$$

Das Matching auf dem Kantenbild erfolgt über die Formel:

$$R_{edge}(x,y) = \frac{\sum\limits_{x',y'} (T_e(x',y') \cdot I_e(x+x',y+y'))}{\sqrt{\sum\limits_{x',y'} T_e(x',y')^2 \cdot \sum\limits_{x',y'} I_e(x+x',y+y')^2}}. \tag{3.12}$$

Dabei ist $R_{colored}$ bzw. R_{edge} der Ergebnisraum beim TM, jeweils für I_c als farbiges bzw. I_e als binäres Originalbild und T_c als farbiges bzw. T_e als binäres Template jeweils an der Stelle (x, y).

[13] Schablonenabgleich
[14] Rot-Grün-Blau

Abbildung 3.15: a) Im Originalbild wird der durch Stufe 2 initial gefundene Button 6 im gesamten Bild noch einmal gesucht, b) Ergebnisbild $R_{colored}$ nach Matching im RGB-Bild; dabei stellen helle Bereiche geringe und dunkle Bereiche große Übereinstimmung dar, c) invertiertes Ergebnisbild R_{edge} nach Matching im Kanten-Bild, (wegen der invertierten Darstellung) sind auch hier die hellen Bereiche geringe und die dunklen Bereiche große Übereinstimmung (im Normalfall umgekehrt)

Die Formel 3.11 summiert die quadrierten Differenzen zwischen Template und den einzelnen Bildausschnitten auf und normiert sie. Da der Unterschied zwischen dem Button-Template und dem jeweiligen Bildausschnitt minimal sein muss, um einen Match zu erzielen, wird im Ergebnisbild $R_{colored}$ nach lokalen Minima gesucht. Ein Minimum im Ergebnisbild bedeutet hier eine maximale Übereinstimmung mit dem Template.
Bei Formel 3.12 handelt es sich um eine normierte Kreuzkorrelation[15], die jedes Pixel aus dem Kantenbild des Templates mit dem entsprechenden Pixel aus dem Kantenbild des Originals multipliziert und anschließend auf denselben Wert normiert wie in Formel 3.11. An den Stellen im Ergebnisbild R_{edge}, wo sich auf diese Weise große Werte ergeben, gibt es auch eine große Übereinstimmung mit dem Kanten-Template. Es wird davon ausgegangen, dass sich dort mögliche neue Buttons befinden, wenn zuvor beim TM mit dem farbigen Template in derselben nahen Umgebung ein guter Match vorlag.
Nun kann es passieren, dass wegen der in Abschnitt 3.1.2 beschriebenen schwierigen Lichtverhältnisse die initial gefundenen Buttons als Template heller sind, als andere noch im Bild enthaltene. Zwar ist die normierte Kreuzkorrelation aus Gleichung 3.12 helligkeitsinvariant, diese wird aber nur für das Kanten-Template verwendet. Das TM für das farbige Template wird auf dem Originalbild dagegen mit der Gleichung 3.11 umgesetzt, da hier bessere Resultate erzielt werden konnten. Da diese Methode nicht helligkeitsinvariant ist, wird vorgeschlagen, die Helligkeit jedes Button-Templates mehrfach anzupassen, sodass jedes farbige Template im Bild mehrmals gesucht wird. Dazu wird jeder Pixelwert pro Farbkanal einfach neu berechnet:

$$I_{neu}(x, y) = c_1 \cdot I_{alt}(x, y) + c_2.$$

[15] Korrelation: Erwartungswert des Produkts zweier Zufallsvariablen [MH04, S.74]

c_1 steht dabei für den Faktor, um den der Kontrast des Bildes verändert werden soll, was allerdings hier nicht benutzt wurde. c_2 dagegen ist ein einfacher Helligkeits-Offset, der zu der aktuellen Helligkeit dazu addiert wird.
Im Experiment hat sich herausgestellt, dass es günstig ist, wenn das Template jeweils dreifach mit den Helligkeitsstufen von $c_2 = -25$, $c_2 = 0$ und $c_2 = 25$ im Bild gematcht wird (wobei der Wertebereich der Graustufen $[0, 255]$ ist).
Nachdem auf diese Weise die Anzahl vorhandenen Buttons erweitert werden konnte, müssen diese Änderungen auch im Graphen angepasst werden.

3.5.2 Neuberechnung des Button-Graphen

Aus den neu hinzugekommenen Buttons wird erneut ein Graph G_{TMBtn} erstellt. Dieser enhält nun zum Teil Knoten, die durch die Stufe 2 (Abschnitt 3.4) schon detektiert wurden und solche, die untereinander doppelt gematcht wurden und daher dicht beieinander liegen. Diese Doppelungen gilt es nun durch einfache Abstandsberechnung untereinander und zu den initialen Buttons in $G_{circBtn}$ bzw. $G_{quadBtn}$ zu eliminieren.
Zwei Buttons N_i und N_j im Graph G_{TMBtn} gelten dann als dieselben, wenn zum einen der Flächeninhalt um weniger als 10% voneinander abweicht und zum anderen das Buttonzentrum um weniger als eine halbe Kantenlänge des Buttons voneinander entfernt liegt. Sind beide Voraussetzungen erfüllt, wird der Knoten N_j mit dem kleineren Flächeninhalt aus dem Graphen entfernt.
Nun werden beide Graphen ineinander verschmolzen, d.h. alle verbliebenen Knoten N_i werden in den Graphen $G_{circBtn}$ bzw. $G_{quadBtn}$ überführt. Es werden nun alle Kanten E_i abhängig von den Abständen der Buttons zueinander, wie bereits in Abschnitt 3.4.2 beschrieben, erneut berechnet und der größte verbundene Subgraph $G_{results}$ extrahiert. In diesem sind dann die über das Gesamtverfahren detektierten Buttons enthalten. Nach dem Zurückrechnen der Koordinaten durch die inverse Homographie (Gleichung 3.3.4) können die Buttons im Originalbild markiert und ausgewertet werden.

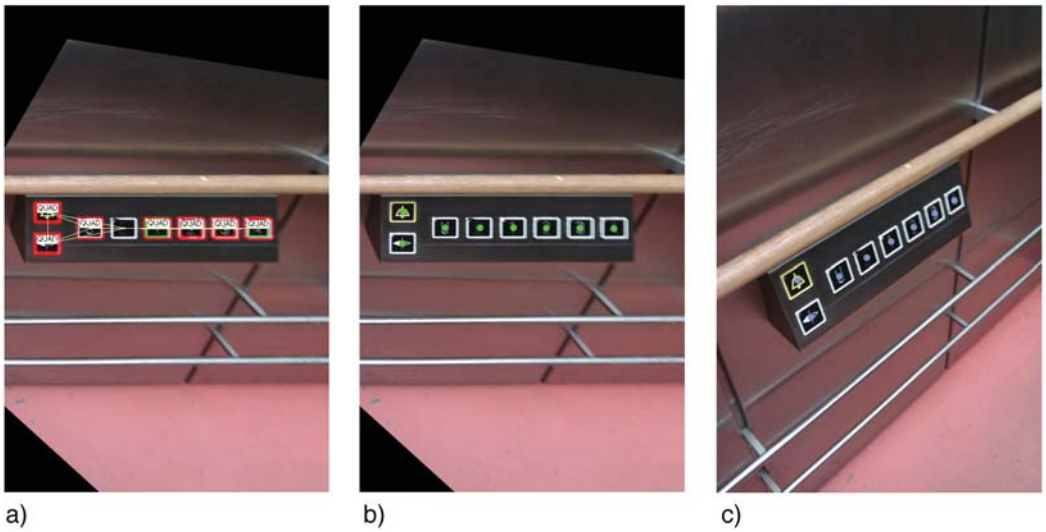


Abbildung 3.16: a) Rektifiziertes Bild mit gefundenen Buttons, dargestellt als Graph (Knoten: rot, Kanten: gelb) b) exportierte Button-Knoten auf rektifiziertes Bild (grün) c) exportierte Button-Knoten zurückgerechnet auf das Originalbild (hellblau)

3.6 Implementierung

Die Implementierung der Button-Detektion wurde zum größten Teil in Python 2.7[16] mit Hilfe von OpenCV[17] Version 2.4 umgesetzt (Stufe 2 und 3). OpenCV steht für Open Source Computer Vision Library und ist eine Bibliothek mit hunderten von Bildverarbeitungs-Algorithmen, die unter BSD-Lizenz[18] frei zur Verfügung steht. Die Ermittlung paralleler Linien zur Rektifizierung in Stufe 1 ist in Matlab[19] erfolgt.

3.6.1 Kantenbild

Das entsprechende Kantenbild lässt sich in OpenCV über vorimplementierte Funktionen gut berechnen. Dabei wurde auf die Funktion *canny*() zurückgegriffen, die mit Hilfe des Schwellwertverfahrens von Otsu durch die Funktion *threshold*() und dem Parameter $cv2.THRESH_OTSU$ den oberen und unteren Schwellwert im Bild variabel berechnet.
Die Dilatation kann mit OpenCV durch die Verwendung der Funktion *dilate*() und das Schließen durch die Funktion $morphologyEx()$ und dem Parameter $cv2.MORPH_CLOSE$ durchgeführt werden.

3.6.2 Rektifizierung

Die Vorberechnungen für die Rektifizierung ließen sich besonders gut mit Matlab umsetzen, da Matlab hier die Möglichkeit bietet, den über die interne Funktion $[hit, theta, d] = hough(edgePicture)$ erstellten und in einer Matrix gespeicherten Hough-Raum wie in Abschnitt 3.3.3 erklärten Weise aufzuteilen und weiterzuverarbeiten. Außerdem ist die Lösung des Gleichungssystems 3.5 durch symbolisches Lösen hier möglich und Zwischen- und Endergebnisse können gut visualisiert werden.
Nachdem die Eckpunkte über die in Abschnitt 3.3 beschriebene Art und Weise ermittelt wurden, wird die eigentliche Rektifizierung dann in Python ausgeführt. Dazu müssen beide Programme miteinander interagieren, was mit Hilfe des Python-Moduls *pymatbridge*[20] umgesetzt werden konnte. Matlab wird dann aus Python heraus gestartet und die entsprechenden Funktionen werden nacheinander aufgerufen. Nachteil dieses Moduls ist es, dass eine Installation von Matlab auf dem System vorhanden sein muss.
Die Berechnung der Homographie erfolgt über die Funktion $findHomography()$, die von OpenCV bereitgestellt wird und die die zuvor über Matlab berechneten Parameter verlangt. Die eigentliche Rektifizierung wird dann mit der Funktion $warpPerspective()$ durchgeführt.

3.6.3 Formendetektion und Graphenerstellung

Zur Detektion von Formen gibt es bereits viele Online Tutorials[21]. OpenCV bietet hier die Möglichkeit die im Abschnitt 3.4.1 beschriebenen Konturen und die Approximation auf einfachere Strukturen mit Hilfe der Funktionen $findContours()$ und *approxPolyDP()* umzusetzen.
Als sehr nützliches Hilfsmittel in der Erstellung und Anpassung von Graphen hat sich das freie Modul *networkx*[22] erwiesen, da hier der volle Funktionsumfang zum Arbeiten mit Graphen gegeben ist. Die verschiedenen miteinander verbundenen Knoten im Graphen können hier beispielsweise über die Funktion *connected_components*() berechnet werden. Eine einfache Abfrage, der Anzahl

[16] `https://docs.python.org/2/`
[17] `http://docs.opencv.org/modules/core/doc/intro.html`
[18] `http://opensource.org/licenses/bsd-license.php`
[19] `http://www.mathworks.de/products/matlab/`
[20] `https://pypi.python.org/pypi/pymatbridge`
[21] gutes Beispiel: `http://opencvproject.wordpress.com/projects-files/detection-shape/`
[22] `http://networkx.github.io/index.html`

der enthaltenen Elemente, liefert dann den größten Subgraphen mit den tatsächlichen Buttons zurück.

3.6.4 Template Matching

Das TM (aus Abschnitt 3.5) kann ebenso über die OpenCV Funktion *matchTemplate*() erfolgen. Dabei werden die Ergebnisse in einem *numpy*-Array[23] gespeichert und anschließend nach den besten Matching Ergebnisse gefiltert. Dazu wird das TM wie beschrieben auf zwei verschiedene Methoden durchgeführt: Mit dem Parameter *cv2.TM_CCOEFF_NORMED*, wird eine Korrelation über Formel 3.12 auf dem Kantenbild ausgeführt und der Parameter *cv2.TM_SQDIFF_NORMED* matcht das farbige Bild anhand der Formel 3.11. Bevor die Ergebnisse ausgegeben werden, müssen die entstandenen Ergebnisräume der Größe nach entweder absteigend (beim TM auf dem Kantenbild) oder aufsteigend (beim TM auf Farbbild) sortiert werden, bevor die n besten Ergebnisse ausgegeben werden können.

3.6.5 Evaluation

Um den Algorithmus auf einer größeren Menge an Fahrstuhlbildern testen zu können, bietet es sich an, das Verfahren in Hinblick auf die Auswertung (siehe Kapitel 4) so zu automatisieren, dass die gefundenen Buttons automatisch abgeglichen und statistisch ausgewertet werden. Dazu wurde zu jedem untersuchten Fahrstuhlbild eine Maske manuell erstellt, die den Bereich der Buttons definieren. Die Erstellung ist simpel: Die Maske ist ein initial schwarzes Graustufenbild, in dem alle Bereichen, in denen sich ein Button befindet, kreisförmig mit weiß[24] ausgefüllt sind (siehe Abbildung 3.17).

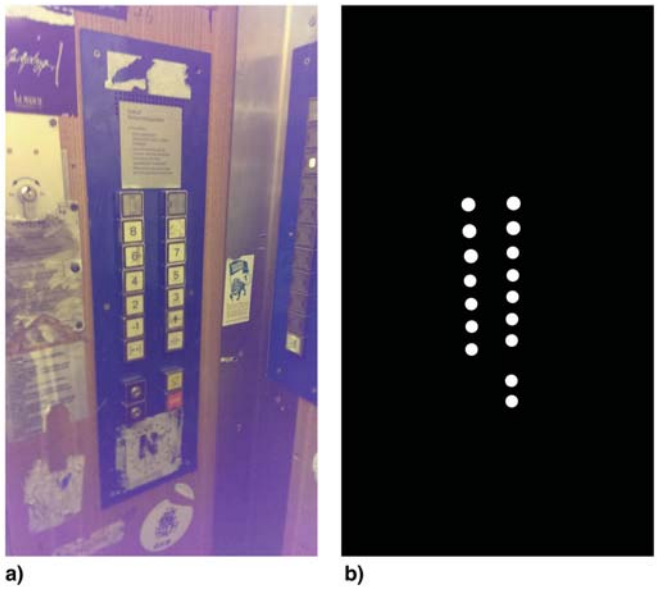


Abbildung 3.17: a) Originalbild b) dazugehöriges Maskenbild

[23] `http://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html`
[24] weiß bedeutet ein Pixelwert von 255, schwarz heißt ein Pixelwert von 0

Alle gefundenen Buttons, werden nun auf dieser Maske abgeglichen. Ist der Pixelwert an der gefundenen Koordinate im Maskenbild weiß so liegt ein positiver Treffer vor. Handelt es sich um einen zulässigen Treffer, werden nun in einer bestimmten Umgebung der Koordinate alle Pixelwerte wieder auf schwarz gesetzt, damit vermeintliche Buttons in der gleichen Umgebung als falscher Treffer gezählt werden. Die Anzahl der im Originalbild befindlichen Buttons, sowie die Größe der kreisförmigen Pixelumgebung werden einfach in dem Namen des Maskenbildes kodiert und dort auch wieder ausgelesen. So kann pro Fahrstuhlbild einfach berechnet werden, wie viele Buttons korrekt erkannt und wie viele ggf. falsch detektiert wurden.

Kapitel 4

Auswertung

In diesem Kapitel werden die verwendeten Verfahren evaluiert. Wie bereits angedeutet, können die Ergebnisse abhängig von den benutzten Parametern, sowie den aufgenommenen Bildern stark variieren. Dazu wird als Erstes ein Überblick der in der Datenbank vorhanden Bilder gemacht und anschließend geklärt, wie sich die später angegebenen Maßzahlen berechnen lassen. Es wird dann ein Überblick über die einstellbaren Parameter gegeben und in dem darauffolgenden Abschnitt die Detektionsrate beschrieben sowie die Erkenntnisse wiedergegeben, die bei der Benutzung verschiedener Parameter gemacht werden konnten.
Dabei wird sowohl untersucht, welchen Einfluss die Dilatation im Vergleich zum Operator Schließen auf die Detektion hat, als auch welchen Gewinn die Stufen 1 (Rektifizierung) und 3 (TM) haben.
Im letzten Teil des Kapitels gibt es einen Abschnitt, in dem einige geeignete und ungeeignete Beispielbilder und deren Ergebnisse gezeigt werden und es wird nochmals auf besondere Probleme eingegangen.

4.1 Statistische Auswertung

Zur Evaluation des im Kapitel 3 beschriebene Detektionsverfahren wurden insgesamt 100 Fahrstuhlbilder genutzt, die aus über 70 verschiedenen Fahrstühlen stammen. Aus diesem Grund varieren die Größe, Form und die Anzahl der zu detektierenden Buttons in den verschiedenen Bildern sehr stark. Viele der Fotos wurden selbst aufgenommen und der restliche Teil stammt aus öffentlichen Bildern aus dem Web. Bis auf eine Mindestanzahl an Pixeln von 0.2 Megapixel, gleichmäßiger Ausleuchtung und Schärfe des Bildes wurden keine weiteren Voraussetzungen an die Bilder gestellt. Es wurden auch bewusst zum Teil schwierige Bilder zugelassen.
In der Auswertung wurde versucht herauszufinden, wie sich bestimmte Parameter auf das Detektionsergebnis auswirken und bei welchen Werten die höchste Detektionsrate erzielt werden konnte. Dazu wurden die Werte für Precision und Recall, beschrieben in Abschnitt 4.1.1, berechnet und miteinander verglichen. Die verwendeten Parameter können in der Tabelle 4.2 eingesehen werden. Es wurde außerdem getestet, wie gut die Verzerrung der Bilder detektiert werden konnte und welchen Einfluss die Rektifizierung auf die Detektion der Bedienelemente besitzt.

4.1.1 Precision und Recall

Precision[1] und Recall[2] sind Maßzahlen zur Beurteilung der Qualität eines Klassifikationsverfahrens. Im Zusammenhang mit diesem Verfahren werden sie wie folgt berechnet:

$$precision = \frac{buttonTrueHits}{buttonFalseHits + buttonTrueHits}$$

$$recall = \frac{buttonTrueHits}{buttonNumber}$$

Dabei sind $buttonTrueHits$ die richtig Positiven, d.h. die Anzahl an Buttons, die erkannt wurden und auch tatsächlich einen Button darstellen. $buttonFalseHits$ sind die falsch Positiven, d.h. die Anzahl Objekte, die für Buttons gehaltenen wurden, tatsächlich aber gar keine Buttons sind. $buttonNumber$ ist in diesem Zusammenhang die Gesamtanzahl der im Bild vorhandenen Buttons, die gesucht werden sollen.
In der Tabelle 4.1 sind noch einmal die unterschiedlichen Testfälle beschrieben. Es sei zu beachten, dass aufgrund der Vorgehensweise bei der Detektion die richtig Negativen und falsch Negativen nicht gespeichert und somit auch nicht weiter ausgewertet wurden.

	positiv	negativ
richtig	Buttons, die korrekterweise als Buttons erkannt wurden ($buttonTrueHits$)	nicht-Buttons, die korrekterweise als nicht-Buttons erkannt wurden
falsch	nicht-Buttons, die fälschlicherweise als Buttons erkannt wurden ($buttonFalseHits$)	Buttons, die fälschlicherweise als nicht-Buttons erkannt wurden

Tabelle 4.1: Wahrheitsmatrix

Der Recall kann auch als eigentliche Button-Detektionsrate angesehen werden und die Precision ist dementsprechend die Zuverlässigkeit des Verfahrens, d.h. das Verhältnis, wie viele der gefundenen Buttons auch tatsächlich solche in der Realität darstellen.

4.1.2 Parameter

Die nachfolgende Tabelle 4.2 soll einen Überblick über die verwendeten Parameter und ihre Standardwerte geben, die größtenteils bereits in dem vorherigen Kapitel erwähnt wurden.

[1] auch Relevanz, Genauigkeit, positiver Vorhersagewert
[2] auch Trefferquote, Sensitivität

Parameter	Typ	Wertebereich	Standard	Bedeutung
$_dilatation$	*int*	$\{0, 3, 5\}$	-	Größe der Dilatationsmaske
$_closing$	*int*	$\{0, 3, 5\}$	-	Größe der Maske zum Schließen
$_rectify$	*bool*	$\{True, False\}$	-	Rektifizierung wird durchgeführt oder nicht
$_thrMinAreaPercentage$	$float$	$[0, 100]$	0.05	Prozentwert, wie groß ein beliebiges kreisförmiges oder viereckiges Objekt mindestens sein muss, um als Knoten zu einem Graphen hinzugefügt werden zu können; abhängig von der Bildfläche
$_thrMaxAreaPercentage$	$float$	$[0, 100]$	2	Prozentwert, wie groß ein beliebiges kreisförmiges oder viereckiges Objekt maximal sein darf, um als Knoten zu einem Graphen hinzugefügt werden zu können; abhängig von der Bildfläche
$_areaPercentage$	$float$	$[0, 100]$	70	Prozentwert, um den zwei gleichförmige Objekte mindestens flächenmäßig übereinstimmen müssen
$_distancePercentage$	*int*	$[0, 100]$	18	Prozentwert, um den zwei gleichförmige Objekte maximal voneinander entfernt sein dürfen; abhängig von der Bilddiagonalen
$_tm$	*bool*	$\{True, False\}$	True	Template-Matching wird durchgeführt oder nicht
$_brightnessLevelsTM$	*int*	1:0, 3:0, ± 25	3	Anzahl der Helligkeitsstufen
$_maxDifferentMatches$	*int*	$[0, 20]$	5	Anzahl der maximal zu findenden Matches beim TM
$_tmPercentEdge$	*int*	$[0, 100]$	30	Prozentwert, um den die Matches im Ergebnisraum größer sein müssen
$_tmPercentColored$	*int*	$[0, 100]$	20	Prozentwert, um den die Matches im Ergebnisraum kleiner sein müssen

Tabelle 4.2: Übersicht aller einstellbaren Parameter

4.1.3 Ergebnisse

Die folgenden Ergebnisse werden in der Tabelle 4.3 bis 4.8 dargestellt. Dabei wurden in 100 Bilder nach 1435 verschiedenen Buttons (entspricht $buttonNumber$) gesucht. Die Parameter sind hierbei, falls nicht explizit genannt, auf den jeweiligen Standardwert gesetzt (siehe Tabelle 4.2).

Auswirkungen von fixen oder variablen Schwellwerten bei Kantendetektion

Aus der Tabelle 4.3 ist gut zu erkennen, wie sich die Wahl der Schwellwerte für die Kantendetektion auf die Detektion der Buttons auswirkt. Wie im Abschnitt 3.3.1 beschrieben, wurde das binäre Bild, was insbesondere für Stufe 2 verwendet wird, mit variablen Schwellwerten für alle 100 Fahrstuhlbilder ausgerechnet. Zum Vergleich wurde noch einmal das Kantenbild mit festen Schwellwerten berechnet. Es ist deutlich zu erkennen, dass durch die variable Berechnung der Schwellwerte im Vorverarbeitungsschritt der Wert nach der Detektion für Recall um 0.12 ansteigt, die Precision dagegen aber fast unverändert bleibt, weshalb die variable Berechnung der Schwellwerte in jedem Fall bevorzugt und in den folgenden Untersuchungen immer variabel eingestellt wird.

Canny Schwellwerte	$buttonFalseHits$	$buttonTrueHits$	Precision	Recall
fix	143	838	0.85	0.58
variabel	197	1010	0.84	0.70

Tabelle 4.3: Ergebnisse bei Verwendung von variablen Schwellwerten im Canny-Algorithmus (siehe 3.3.1), wobei $_rectify = False$, $_dilatation = 0$, $_closing = 0$ und $buttonNumber = 1435$

Auswirkungen von morphologischen Operatoren

Aus der Tabelle 4.4 wird ersichtlich, wie sich die Detektionsergebnisse bei einer angewendeten Dilatation im Vergleich zum Schließen verhält. Der positive Einfluss von variablen Schwellwerten bei der Kantendetektion konnte bereits gezeigt werden. Da die Detektionsergebnisse weiter verbessert werden sollen, wurde zum einen Schließen und zum anderen die Dilatation angewandt. Es ist deutlich erkennbar, dass beide Operationen einen positiven Einfluss auf die Gesamtanzahl der detektierten Knöpfe hat, sodass der Recall in beiden Fällen um 0.02 ansteigt. Auffällig ist hier allerdings auch, dass die Anzahl der falsch positiven beim Schließen deutlich geringer ist, als bei der Dilatation. Trotzdem ist die Gesamtanzahl an detektierten Buttons bei Dilatation am größten.

$_dilatation$	$_closing$	$buttonFalseHits$	$buttonTrueHits$	Precision	Recall
0	0	197	1010	0.84	0.70
3	0	293	1040	0.78	0.72
0	3	198	1028	0.84	0.72

Tabelle 4.4: Ergebnisse der Dilatation im Vergleich zum Schließen, wobei $_rectify = False$ und $buttonNumber = 1435$

Auswirkungen der Rektifizierung

Erstaunlich sind die Ergebnisse im Direktvergleich bei der Rektifizierung, die sich aus Tabelle 4.5 entnehmen lassen. Die Rektifizierung ist dafür gedacht, dass die Buttons ihre ursprünglichen Größen wieder erhalten, sodass beispielsweise gegenüberliegende Kanten gleich lang werden und die Flächeninhalte vergleichbar sind. Dadurch sollten möglichst viele dieser gleichen Formen detektiert werden.

$_rectify$	$buttonFalseHits$	$buttonTrueHits$	Precision	Recall
$True$	169	975	0.85	0.68
$False$	197	1010	0.84	0.7

Tabelle 4.5: Ergebnisse bei verwendeter Rektifizierung, wobei $_dilatation = 0, _closing = 0$ und $buttonNumber = 1435$

In den Ergebnissen allerdings hat sich herausgestellt, dass die Rektifizierung in der Gesamtheit keine Verbesserung bringt, da die Detektionsraten hier nicht größer wurden.
Zum einen liegt es daran, dass in bestimmten Fällen nicht die dominanten Linien des Button-Panels selbst extrahiert wurden, sondern die einer anderen Ebene. Dementsprechend wurden nicht die Bedienelemente entzerrt, sondern eine andere Fläche, was dazu führen kann, dass Buttons der anderen Ebene nicht korrekt detektiert werden können. In einem solchen Fall sind mehr als zwei Fluchtpunkte im Bild enthalten und damit ist eine korrekte Entzerrung schwer möglich.
Die Vermutung liegt nahe, dass ein weiterer Grund hierfür in dem Schwellwert $_areaPercentage$ liegt, da dieser möglicherweise zu klein gewählt wurde (siehe Parametertabelle). Dieser Parameter ist ein Schwellwert, der angibt, wie groß die Flächenübereinstimmung von einem bereits erkannten Button mit einer weiteren Form mindestens sein muss, damit diese mit als Button aufgenommen wird.
Um dies zu untersuchen wurden weitere Experimente durchgeführt, bei denen der Standardwert für $_areaPercentage$ von 70% auf 90% Übereinstimmung erhöht wurde. Das Resultat ist in der nachfolgenden Tabelle 4.6 dargestellt. Es ist zu beachten, dass hier nicht mehr alle 100 Fahrstuhlbilder verwendet wurden, sondern nur noch diejenigen, bei denen eine korrekt Rektifizierung durchgeführt werden konnte.
Es wurden insgesamt 47 der 100 Bilder rektifiziert, wovon allerdings nur bei 36 Bildern eine korrekte Entzerrung erzielt werden konnte. Bei 53 der insgesamt 100 Bilder war eine stärkere Verzerrung im Bild vorhanden, bei der eine Rektifizierung auch sinnvoll gewesen wäre. Demnach wurden 31 dieser stärker verzerrten Bilder korrekt rektifiziert (entspricht fast 60% aller notwendigen Rektifizierungen) und bei weiteren 5 Bildern konnte eine korrekte Rektifizierung trotz geringerer Verzerrung durchgeführt werden.

$_rectify$	$buttonTrueHits$	$buttonFalseHits$	$buttonNumber$	Precision	Recall
$False$	215	19	424	0.92	0.51
$True$	279	31	424	0.90	0.66

Tabelle 4.6: Ergebnisse auf 31 korrekt rektifizierten Bildern einmal mit und ohne Rektifizierung, wobei $_dilatation = 0, _closing = 0$ und $_areaPercentage = 90$

Wie aus der Tabelle 4.6, abgelesen werden kann, ist die Detektionsrate bei einer korrekten Rektifizierung und strikterem *_areaPercentage*-Parameter deutlich besser, da der Recall um 0.15 steigt. Das liegt vor allem auch daran, dass die entsprechenden Flächen gleicher Buttons weniger stark variieren dürfen, was allerdings der Fall ist, wenn das Bild verzerrt ist. Durch eine Entzerrung werden dagegen die Flächeninhalte annähernd gleich und mehr Knöpfe werden gefunden.
Die Vermutung konnte bestätigt werden und es kann weiterhin gesagt werden, dass eine Rektifizierung grundsätzlich geeignet ist, um gute Detektionsergebnisse zu erhalten. Allerdings wird hierdurch auch die Anfälligkeit der Rektifizierung deutlich, wenn eine Fahrstuhlaufnahme nicht nach den gegebenen Anforderungen gemacht wurde.
Probleme bei der Rektifizierung können immer dann auftreten, wenn die Bedienelemente im Fahrstuhlbild auf einer anderen Ebene liegen, als die mit den signifikantesten Linien. Das kann insbesondere dann auftreten, wenn die Knöpfe schräg zur Wand auf einer separaten Ebene angebracht sind. Weitere Probleme stellen hier besonders Fahrstühle mit wenig Struktur dar. Es können in diesem Fall nicht ausreichend Linien extrahiert werden, sodass keine Rektifizierung stattfindet. In manchen Fällen kommt es allerdings auch vor, dass aufgrund von den bereits beschriebenen Pseudolinien (siehe 3.3.3) falsche Geraden entstehen, die zwar beim Lösen des Gleichungssystems zu Ergebnissen führen können, die allerdings eine falsche Ebene detektieren und somit eine fehlerhafte Rektifizierung berechnen.

Auswirkungen des TM

Im Folgenden wird untersucht, wie sich das TM auf die Anzahl der in der Gesamtheit detektierten Knöpfe auswirkt. Die genauen Auswirkungen verschiedener Parameter speziell beim TM wird im nächsten Abschnitt untersucht. Die Tabelle 4.7 zeigt deutlich, dass sich die Anzahl der gefundenen Buttons im Bild drastisch erhöht, wenn das TM zusätzlich angewendet wird. So kann bei der Anwendung der Stufe 3 der Recall um 0.13 vergrößert werden, was allerdings auch zu Lasten der Genauigkeit geht. Diese verschlechtert sich deutlich, da das Verfahren auch weitere Buttons erkennt, die in Wirklichkeit nicht vorhanden sind.

_tm	*buttonFalseHits*	*buttonTrueHits*	*buttonNumber*	Precision	Recall
False	74	820	1435	0.92	0.57
True	197	1010	1435	0.84	0.70

Tabelle 4.7: Ergebnisse im Vergleich bei angewendetem TM, wobei $_rectify = False$, $_dilatation = 0$, $_closing = 0$

Auswirkungen verschiedener TM-Parameter

Anhand der Ergebnisse aus Tabelle 4.8 ist zu erkennen, dass wenn die Prozentwerte für das TM variieren, auch die Ergebnisse sehr verschieden sind.
So wird der Recall bei höherem Schwellwert für das Matching auf dem binären Template und geringerem Schwellwert für das Matching auf dem farbigen Template, um 0.04 schlechter. Die Anzahl der falsch positiv erkannten nicht-Buttons dagegen wird gleichzeitig geringer.
Dieses Ergebnis war zu erwarten, da die Regeln für das TM in der 3. Stufe bei erhöhtem *_tmEdge Percentage* bzw. verringertem *_tmColoredPercentage* nun strikter sind und dadurch auch mehr potenzielle Buttons eher verworfen werden, als zuvor. Durch diese Restriktionen fallen gleichzeitig natürlich auch wiederum weniger falsch Positive in das Ergebnis. Dies wirkt sich jeweils positiv auf die Precision aus, die sich um 0.03 erhöht.

$_tmEdge\ Percentage$	$_tmColored\ Percentage$	$buttonFalseHits$	$buttonTrueHits$	Precision	Recall
30	20	197	1010	0.84	0.7
60	10	135	942	0.87	0.66

Tabelle 4.8: Ergebnisse bei veränderten TM-Parametern, wobei $_rectify = False, _dilatation = 0, _closing = 0$

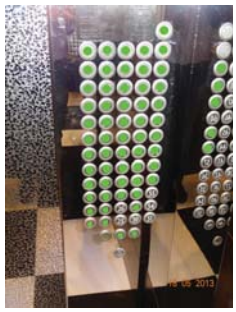
Die verschiedenen Werte für die Parameter beim TM zeigen den notwendigen Kompromiss zwischen Detektionsrate und Präzision der gefundenen Buttons, der hier eingegangen werden muss.

4.2 Beispielbilder

Die besten Detektionsergebnisse werden in den Bildern erzielt, in denen sich die Buttons klar vom Untergrund auf dem sie sich befinden, hervorheben. Dies ist oft genau dann der Fall, wenn die Bedienelemente beispielsweise in einer anderen Farbe oder durch eine äußeren Rand visuell abgegrenzt werden. Dadurch entsteht eine deutliche Bildkante, die durch den Kantendetektionsalgorthmus (Abschnitt 3.3.1) zuverlässig sichtbar gemacht werden können.
Demgegenüber stehen Fahrstuhlbilder, dessen Buttons metallisch sind, wie deren Untergrund auf dem sie sich befinden. In diesen Fällen ist eine Detektion schwieriger. Besonders bei der initialen Konturendetektion (siehe Abschnitt 3.4) sind dann gute Ergebnisse wichtig und eine Voraussetzung um die nicht gefundenen Buttons per TM zu finden.
Im Folgenden werden Beispielbilder gegenübergestellt und miteinander verglichen, die beispielsweise sehr gut für die Rektifizierung und die Detektion der Buttons geeignet sind und welche Bilder sich als eher unpraktisch für eine akkurate Buttondetektion erwiesen haben.

#	**Fahrstuhlbild**	**Auswertung**
1		Das Bild ist trotz der teils verschwommenen Buttons sehr gut für eine Buttondetektion geeignet, da sich die Buttons von dem Untergrund deutlich hervorheben. Trotz verwendeter Dilatation und TM, können hier auf dem Originalbild nicht alle Buttons gefunden werden, da dieses nicht rektifiziert wurde.
1		Dank Rektifizierung können hier deutlich mehr Buttons gefunden werden, da zum einen die Formen besser zugeordnet werden konnte und zum anderen bessere Templates für das TM entstehen.

2		Das Bild stellt eine Herausforderung dar, da die zu detektierenden Buttons sich kaum von dem ebenso metallischen Untergrund, auf den sie sich befinden hervorheben. Hier konnten allerdings trotz des schlechten Kantenbildes mit Hilfe vom TM weitere Buttons gefunden werden. Es ist aber zu erkennen, dass durch das TM ein falsch positiver Button erkannt wird.
3		Auch dieses Fahrstuhlbild ist nicht optimal, grundsätzlich aber besser als Bild 2. Die Buttonkanten grenzen sich hier auch optisch von dem Untergrund ab, sodass eine Detektion einfacher wird. Hier wurde allerdings auch durch das TM in der Nähe eines andere Bedienelements ein weiterer Button gefunden, der nicht vorhanden ist.
4		Auf den ersten Blick scheint dieses Bild wie Bild 2 eher ungeeignet, da die Buttons metallisch sind und die Beleuchtung ungleichmäßig ist. Da die Buttons aber deutlich dunkler sind als der Untergrund, können die Formen gut gefunden werden. Trotz der teilweise sehr ähnlichen Struktur und Farbe der einzelnen Buttons und der im Bild befindlichen Umrandungen, wird nur ein falsch positiver Button detektiert. Das liegt zum einen an den Schwellwerten beim TM und zum anderen an der relativen Entfernung, die die Buttons maximal voneinander entfernt sein dürfen.
5		Dieses Bild eignet sich sehr gut zur Buttondetektion. Da allerdings in den meisten Fällen der Button, der den Alarm auslöst, gelb oder rot ist, schlägt das TM auf diesen Button nicht an, die Farbe des Template-Buttons weiß ist.

6		Auch dieses Bild stellt eine Herausforderung dar, da die Buttons sich nur unmerklich von dem Hintergrund abheben. Durch korrekte Rektifizierung und mit Hilfe von TM konnten allerdings weitere Buttons gefunden werden.
6		Dieses Bild wurde in demselben Fahrstuhl aufgenommen, wie das Bild zuvor. Man kann den Unterschied der Bedienelemente deutlich erkennen. Bei gedrückten Buttons ergeben sich auch deutlich bessere Kantenbilder, sodass durch das initiale Verfahren aus Stufe 2 bereits die besten Treffer erzielt werden können.
7		Das Bild weist eine starke perspektivische Verzerrung auf die detektiert und enterrt werden konnte. Durch die weißen Umrandungen der Buttons fällt die Suche vergleichsweise einfach aus und selbst in einem nicht entzerrten Bild konnten wie hier alle Bedienelemente gefunden werden.
8		In diesem Bild gelingt eine vollständige Detektion der Bedienelemente, wie auch zuvor im Fahrstuhlbild 7. Die Buttons heben sich hier auch wieder deutlich vom metallischen Untergrund ab.
9		Trotz dessen, dass sich die Bedienelemente deutlich vom Untergrund herausheben, stellt dieses Bild eine Herausforderung dar. Viele Buttons sind nämlich gespiegelt und dadurch noch einmal vollständig am rechten Bildrand zu betrachten. Da diese Spiegelungen alle Voraussetzungen für einen Button besitzten, werden sie auch als solche sehr häufig detektiert, was zu Lasten der falschpositiv Rate geht. (Bildquelle: Rahul Gaur)

Ein Überblick aller verwendeten Bilder kann im Anhang A betrachtet werden.

4.2.1 Probleme

Wie anhand der Beispielbilder ersichtlich wurde, können zwar die meisten Buttons detektiert werden, oftmals werden aber auch falsch positive Buttons erkannt. Das liegt zum einen daran, dass der Algorithmus beispielsweise nicht auf Spiegelungen sensibilisiert ist und nur abhängig von Form und Größe klassifiziert. Eine mögliche Lösung ist eine genaue Analyse der Zahlen auf den Buttons durchzuführen, da hierdurch festgestellt werden könnte, ob die Ziffern gespiegelt dargestellt sind oder nicht.
Durch den formbasierten Algorithmus kann sich ein weiteres Problem ergeben. Viele der verwendeten Fahrstuhlbilder stammen aus amerikanischen Fahrstühlen und stellten eine weitere Besonderheit dar: Die zu den Buttons dazugehörigen Ziffern oder Symbole sind oftmals nicht direkt auf dem Button selbst aufgedruckt, sondern befinden sich links daneben. Grundsätzlich stellt dies zwar kein Problem dar, da aber die Ziffern meist auch auf einer rechteckigen oder runden Form abgedruckt werden, schlägt das Detektionsverfahren hier auch an und erkennt diese Form, die sich in großer Anzahl, gleicher Form und Größe wiederholt. Zwar gehören diese Elemente immer direkt zu einem Button, sind selbst aber keine Buttons. Dies könnte dann im Fall einer Weiterverwendung für Robotersysteme zu Problemen führen, wenn der jeweilige Knopf gedrückt werden soll. Das Problem könnte aber durch weitere Restriktionen umgangen werden, beispielsweise dadurch, dass sich die Ziffern und Symbole direkt auf dem Button befinden müssen.

Kapitel 5

Ausblick

5.1 Zusammenfassung

Die Auswertung der Ergebnisse der in dieser Arbeit implementierten Pipeline zeigt, dass die Buttons in einigen Fällen mit sehr hoher Genauigkeit detektiert werden, allerdings wiederum in anderen Fällen keine guten Ergebnisse erzielt werden können. Die Detektion ist sehr von der Qualität des Bildes abhängig und eine Generalisierung des Problems bis zur vollständigen Detektion aller Buttons ist daher nach wie vor komplex.
Die Detektionsrate von Fahrstuhlbuttons kann bis zu 72% bei einer Menge von 100 ausgewählten Bildern betragen. Dabei ist die Verwendung von morphologischen Operationen zur Verbesserung der Gesamtergebnisse günstig.
Bei der Rektifizierung wurde überraschend gezeigt, dass die Resultate auf einer größeren Menge an Testbildern nicht besser werden. Das zeigt zum einen, dass nicht alle Bilder geeignet sind, um korrekt rektifiziert zu werden. Zum anderen aber zeigt es auch, dass die konturenbasierte Detektion der Bedienelemente in Kombination mit TM in den Stufen 2 und 3 relativ robust gegenüber Blickwinkel und perspektivischer Verzerrung ist. Weiterhin konnte gezeigt werden, dass auf eine bestimmte Menge an Bildern, in denen die Verzerrung durch das beschriebene Verfahren in Stufe 1 detektiert und rektifiziert werden konnte, die Ergebnisse durch weiteres Einschränken in den Parametern bei der Genauigkeit (Precesion) mit Rektifizierung besser ausfallen als ohne.
Es konnte weiterhin belegt werden, dass das TM einen positiven Einfluss auf die Gesamtklassifikation hat und für deutlich bessere Ergebnisse sorgt als ohne. Die Stufe 3 der Pipeline erlangt dadurch große Bedeutung im Verfahren.

5.2 Verbesserungsvorschläge und Erweiterungen

Das vorgeschlagene Detektionsverfahren könnte gerade hinsichtlich der falsch positiven detektierten Buttons weiter verbessert werden. Dazu ist es durchaus denkbar die Anordnung der Buttons zueinander als Hinweis darauf zu verwenden, wo sich weitere Bedienelemente befinden können. Oftmals sind die Knöpfe im Fahrstuhl in einer regelmäßigen Gitternetzstruktur angeordnet. Diese Information kann verwendet werden, entweder um weitere im TM gefundene Elemente auszuschließen, da sie nicht in die Anordnung passen oder aber um weitere rechtmäßige Buttons aufzunehmen, wenn die Gitterstruktur eingehalten wird.
Das entwickelte Verfahren ist nicht echtzeitfähig und beschränkt sich daher auf Einzelbilder. Um einen praktischen Nutzen daraus zu gewinnen und beispielsweise, wie in der Einführung beschrieben, für blinde Menschen eine Orientierungshilfe zu geben, muss das Verfahren allerdings schneller

arbeiten, um auch eine Live Erkennung zu ermöglichen. Dazu wäre es notwendig die Entzerrung des Bildes entweder so zu verändern, dass die Bestimmung paralleler Linien schneller erfolgt oder diese vernachlässigt wird, wobei außerdem die Parameter neu angepasst werden müssten. Zusätzlich sollte das TM auf einem deutlich verkleinerten Bild erfolgen, um das Verfahren weiter zu beschleunigen.
Es wäre hier auch denkbar, eine App für Smartphones zu entwickeln, die beim Betreten des Fahrstuhls die Umgebung über die interne Kamera aufnimmt und in Echtzeit analysiert. Über ein akkustisches Feedback, beispielsweise mittels Stimme oder Piepton, kann dabei auf die relative Position des Panels hingewiesen werden. Zusätzlich könnten interne Sensoren wie das Gyroskop benutzt werden, um die Neigung des Bildes festzustellen.
Eine umfangreiche Analyse der detektierten Buttons z.B. mittels OCR[1] wäre hier auch sehr praktisch, da dadurch gleichzeitig die Bedeutung des jeweiligen Buttons ausgegeben werden kann. OCR ist im Allgemeinen allerdings sehr anfällig gegenüber Verzerrung. Das hier vorgestellte Verfahren bietet die Möglichkeit der Entzerrung in der Stufe 1. Allerdings hat sich herausgestellt, dass es anfällig gegenüber schlechten Fahrstuhlbildern ist und der Grad der Verzerrung nicht immer erkannt und somit rektifiziert werden kann. Daher ist es denkbar ein anderes Verfahren zu verwenden oder das bestehende noch so zu verbessern, dass die Ebene mit den Bedienelementen korrekt entzerrt werden kann, was sich positiv auf OCR auswirkt. Dazu ist es denkbar, mehrere Aufnahmen aus verschiedenen Perspektiven derselben Szene zu machen, um diese dann miteinander zu verrechnen und dadurch bessere Ergebnisse in der Rektifizierung zu erhalten.
Die Angabe der relativen Position im Bild und deren Bedeutung kann für Menschen sehr hilfreich sein. Roboter allerdings benötigen in der Regel noch zusätzlich die räumliche Information, um die jeweiligen Bewegungen mit den Maschinenarmen berechnen zu können. Auch hier ist eine Erweiterung des Grundverfahrens denkbar, sofern der Roboter vollständig autonom die Knöpfe bedienen soll. Dazu wäre es sinnvoll, eine weitere Kamera einzusetzen, sodass ein Stereobild des Button-Panels gemacht werden kann. Der Einsatz von Laserscanner würde sich in so einem Fall auch positiv auswirken.

[1] OCR = Optical Character Recognition, bezeichnet Verfahren, um Ziffern und Buchstaben aus Bildern heraus zu extrahieren

Anhang A

Bilder

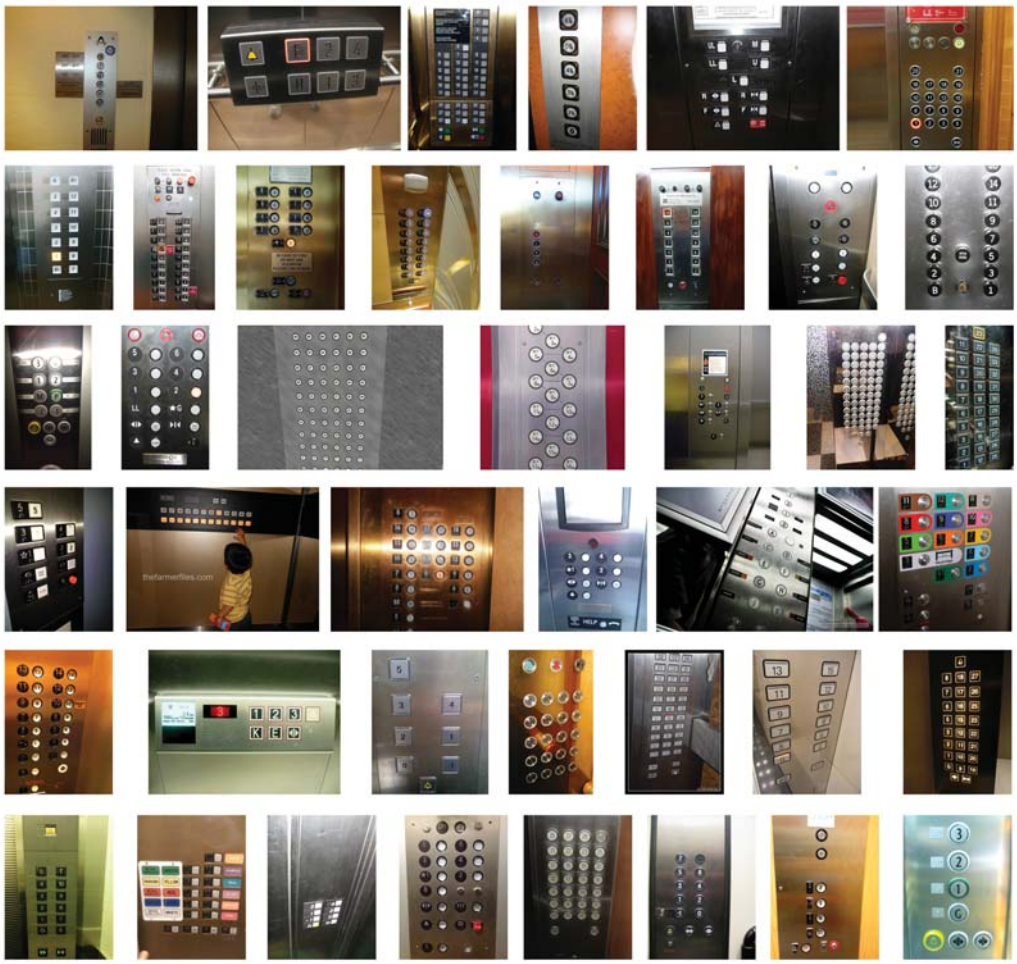


Abbildung A.1: verwendete Fahrstuhlbilder Teil 1.1 (Quellen siehe B.1)

Abbildung A.2: verwendete Fahrstuhlbilder Teil 1.2 (Quellen siehe B.1)

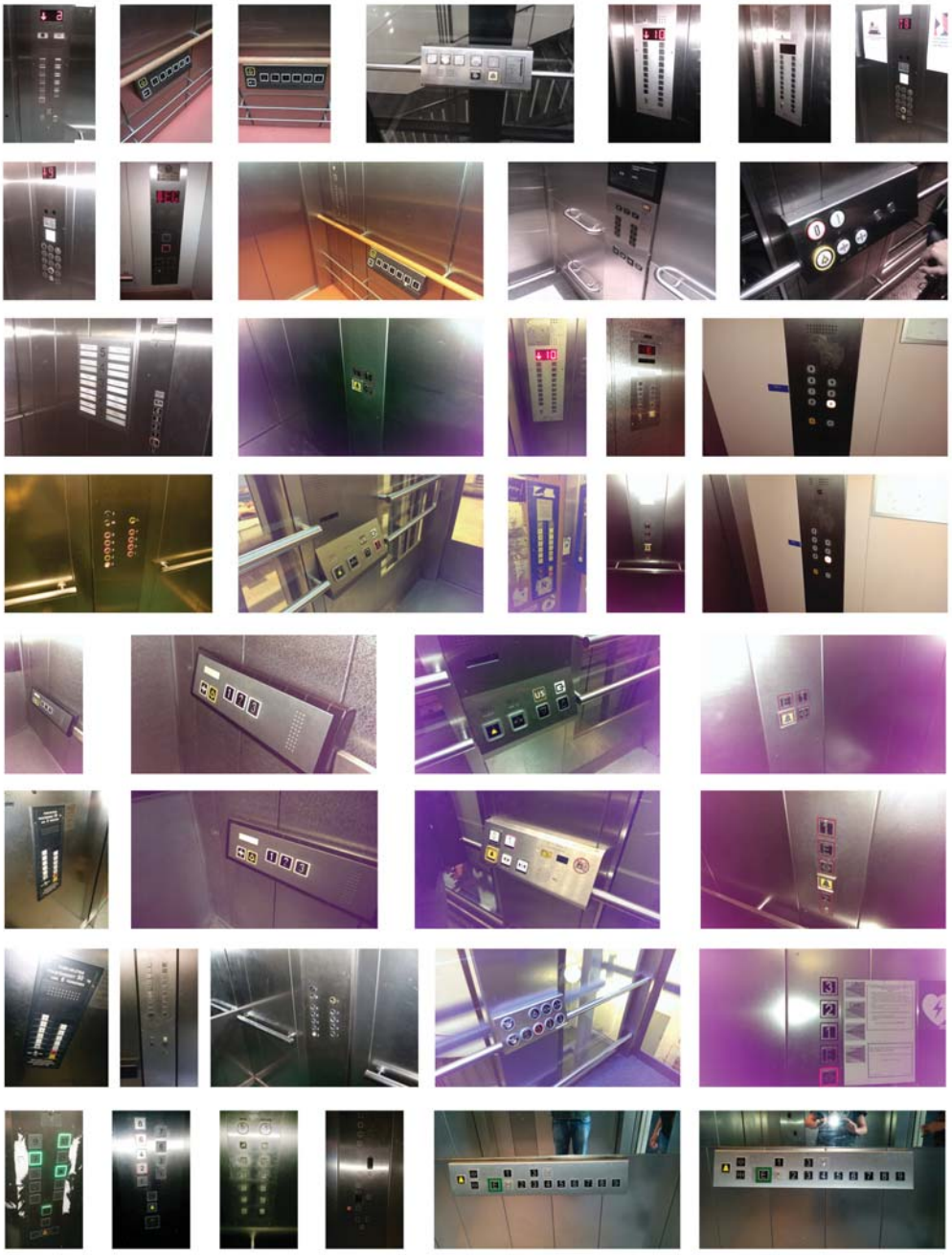


Abbildung A.3: verwendete Fahrstuhlbilder Teil 2 (eigene Bilder)

Anhang B

Quellen

B.1 Bildquellen

Im Folgenden werden die Bildquellen der Fahrstuhlbilder aus Teil 1.1 und 1.2 genannt (in Reihenfolge der Erscheinung).

#	**Autor**	**Copyright**	**URL**
1	holidaycheck.de	©	http://www.holidaycheck.de/hotel-Urlaubsbilder_Arcadia+Hotel+Sonnenhof+Grafenau-ch_ub-hid_64205.html?action=detail&mediaId=1156477736
2	holidaycheck.de	©	http://www.holidaycheck.de/vollbild-Ibis+Styles+Hotel+Muenchen+Ost+Messe+Fahrstuhl-ch_ub-id_1157404994.html
3	Nemo's great uncle	cc pd	https://www.flickr.com/photos/maynard/1932593072/
4	akenin	©	https://www.flickr.com/photos/59809784@N00/326901638/
5	patnedz	©	https://www.flickr.com/photos/patricknedz/3439213253/
6	Michael Roberts	©	https://www.flickr.com/photos/l-michael-roberts/4774708770/
7	Joel Abroad	cc by nc sa	https://www.flickr.com/photos/40295335@N00/6075541102/
8	Diesel Ducy	©	http://www.panoramio.com/photo/61069981
9	Diesel Ducy	©	http://www.panoramio.com/photo/61070259
10	Reto Kurmann	©	https://www.flickr.com/photos/retokurmann/6393792877/
11	Diesel Ducy	©	https://www.flickr.com/photos/dieselducy/6916693440/
12	Diesel Ducy	©	https://www.flickr.com/photos/dieselducy/6955517081/
13	John J. BARTHOLDI, III	©	http://www2.isye.gatech.edu/~jjb/misc/elevators/images/asian-civilizations-museum-singapore.jpg
14	-	©	http://www.elevatorbobs-elevator-pics.com/images2/Buttons/Car/bc100HR.jpg
15	smarty09	©	http://smarty09.files.wordpress.com/2011/10/city-hotel-lift-buttons.jpg

16	John J. BARTHOLDI, III	©	http://www2.isye.gatech.edu/~jjb/misc/elevators/images/concorde-hotel-kl.jpg
17	citymaster13	©	http://i136.photobucket.com/albums/q168/citymaster13/crystal000-1.png
18	Yumekichi11	©	http://i1207.photobucket.com/albums/bb480/Yumekichi11/Picture%209/d8123f83.jpg
19	Evil saltine	CC PD	http://upload.wikimedia.org/wikipedia/commons/c/c9/Dover_elevator_passenger_controls_%283_floors%29.jpg
20	Rahul Gaur	©	http://4.bp.blogspot.com/-awBxOtQs_rU/UdTjVhgu09I/AAAAAAAABzA/3s84R_OTCwk/s1600/DSC03662.JPG
21	Brian Scanell	©	http://1.bp.blogspot.com/-JRmdfoqHgKo/UD8uQuMcOQI/AAAAAAAADOM/i7-53dhgMCQ/s1600/DSC05059.jpg
22	Amanda H. Young	©	http://simplifiedmom.com/wp-content/uploads/2013/07/elevator-buttons-e1374873605485.jpg
23	-	©	http://nourtek.com/wp-content/uploads/2012/04/elevator-buttons2-1024x768.jpg
24	Lori	©	http://laffwithlori.files.wordpress.com/2012/10/elevator-buttons.jpg
25	Susie	©	http://www.newdaynewlesson.com/wp-content/uploads/2012/10/elevator-buttons.jpg
26	Paul Gilowey	©	http://media.capetowndailyphoto.com/wp-content/uploads/2011/07/elevator_IMG_7088.jpg
27	Betty	©	http://3.bp.blogspot.com/_QdEZ277dQbk/TFDXqLmwdbI/AAAAAAAABhM/Jl8kS2uOPiw/s1600/ElevatorButtons.jpg
28	David Williams	©	http://3.bp.blogspot.com/-UOXC9bOhay8/T3TXTtIXn3I/AAAAAAAAAj0/FR2yGcWcELk/s1600/ElevatorButtons.JPG
29	Jan H. Neumann	©	http://www.azgn.de/fileadmin/_fotoware/extratip/08/05/02/ex021980.jpg
30	Anke Gröner	©	http://www.ankegroener.de/Bilder/fahrstuhl.jpg
31	John J. Bartholdi, III	©	http://www2.isye.gatech.edu/~jjb/misc/elevators/images/fraser-suites-singapore.jpg
32	guerrilla exploring.com	©	http://www.guerrillaexploring.com/gesite/public_html/images/guerrilla/GESNK/GESNK-01.jpg
33	Teun Spaans	©	https://justpuzzles.files.wordpress.com/2013/09/hk_admiralty_citic_tower_lift_botton_panel_sept-2012.jpg?w=225&h=300
34	Guaopengtakcm	CC BY	http://upload.wikimedia.org/wikipedia/commons/thumb/b/b7/HK_IFC_FV_%E6%9B%89%E8%96%88_High_Place_18B_showflat_lift_button_panel_Dec-2013.JPG/640px-HK_IFC_FV_%E6%9B%89%E8%96%88_High_Place_18B_showflat_lift_button_panel_Dec-2013.JPG
35	Perinaz Bhada-Tata	©	http://2.bp.blogspot.com/_tkJo2aX0GiY/S7TLYw8k4uI/AAAAAAAADAI/ArYZcJrPm3s/s1600/IMG_0113.JPG

36	Robyn's Posterous	©	http://4.bp.blogspot.com/_hrCjqEFBjyE/TNcxvEKqwfI/AAAAAAAAAPY/jKMWgtm5yfw/s1600/IMG_0636.jpg.scaled.1000.jpg
37	EmissionsNebel	©	http://verstoerer.files.wordpress.com/2010/12/img_1228.jpg
38	Corey Maass	©	http://gelform.com/wp-content/uploads/2013/04/IMG_1529-768x1024.jpg
39	Marc Romanoff	©	https://marcromanoff.files.wordpress.com/2012/01/img_3108.jpg
40	pphant	©	https://pphant.files.wordpress.com/2013/03/img_4840.jpg
41	TibetanTailor	©	http://tibetantailor.com/wp-content/uploads/2013/03/IMG_6214.jpg
42	Mike Naylor	©	http://3.bp.blogspot.com/_upwVwaXwxUM/TLBfUPOz62I/AAAAAAAAElU/MxobDLiiPOQ/s1600/IMG_6412.JPG
43	-	©	http://sfcitizen.com/blog/wp-content/uploads/2011/06/IMG_7139-copy.jpg
44	Joyce Ng	©	http://i578.photobucket.com/albums/ss229/joyaholic/sub%20album%201/IMG_8884.jpg
45	David W. Bowser	©	http://3.bp.blogspot.com/-D87JFCz2TIM/Ta1tL9IuMFI/AAAAAAAABIO/jaG6bkaBKGU/s1600/buttons.jpg
46	SchindlerLift1874	🅭🅯🅭	http://img4.wikia.nocookie.net/__cb20121211154243/elevation/images/2/29/KONE_KSS_470_Buttons.jpg
47	Dave Gustafson	©	http://4.bp.blogspot.com/_EbhbPbkypPY/TBueGxLj08I/AAAAAAAABzc/mEjRd55mh6A/s1600/koreatown-elevator.jpg
48	Josh St John	©	http://www.jstjwrites.com/wp-content/uploads/2014/03/lift-264092_1280.jpg
49	Şafak Cakır	©	http://www.123rf.com/photo_10719438_elevator.html
50	uniqueelevator.com	©	http://www.uniqueelevator.com/wp-content/uploads/2010/03/MG_0592-b.jpg
51	Tomkasing	©	http://upload.wikimedia.org/wikipedia/commons/6/69/HK_Central_New_World_Centre_Lift_button_numbers_Oct-2011.jpg
52	Monique	©	http://mzansi2008.files.wordpress.com/2008/08/p10105031.jpg
53	Veronica Erb	©	http://verbistheword.files.wordpress.com/2010/10/p1010944.jpg
54	Merzmensch	©	http://kwassl.net/wp-content/uploads/R0015246.jpg
55	Johnny Jet	©	http://www.johnnyjet.com/2013/06/20-reasons-why-to-sail-on-royal-princess/royal-princess-cruise-ship-southampton-england-2013-069/
56	OtisElevatorGuy1	©	http://img2.wikia.nocookie.net/__cb20130303011550/elevation/images/thumb/6/60/Series_1_Elevator_Fixtures.PNG/359px-Series_1_Elevator_Fixtures.PNG

57	shutterstock.com	©	http://www.more.com/sites/default/files/shutterstock_124890778.jpg
58	John J. Bartholdi, III	©	http://www2.isye.gatech.edu/~jjb/misc/elevators/elevators.html

Literaturverzeichnis

[BETG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

[Can86] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679–698, Nov 1986.

[CB12] Nidhi Chandrakar and Devanand Bhonsle. Study and comparison of various image edge detection techniques. *International Journal of Managment, IT and Engineering*, 2(5):499–509, 2012.

[DH72] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, January 1972.

[DKK+08] Christian Dornhege, Alexander Kleiner, Rainer Kümmerle, Bastian Steder, Wolfram Burgard, and Bernhard Nebel. Sp-freiburg techx challenge technical paper. In *TechX Challenge*, 2008. Finalist.

[DP73] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, October 1973.

[Dut99] Geoffrey Dutton. Scale, sinuosity, and point selection in digital line generalization. *Cartography and Geographic Information Science*, 26(1):33–54, 1999.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[FYY09] Mei Fang, GuangXue Yue, and QingCang Yu. The study on an application of otsu method in canny operator. In *International Symposium on Information Processing (ISIP)*, pages 109–112. Citeseer, 2009.

[HS88] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[HZ04] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[KAyO07] Jeong-Gwan Kang, Su-Yong An, and Se young Oh. Navigation strategy for the service robot in the elevator environment. In *Control, Automation and Systems, 2007. ICCAS '07. International Conference on*, pages 1092–1097, Oct 2007.

[KCRN10] Ellen Klingbeil, Blake Carpenter, Olga Russakovsky, and Andrew Y Ng. Autonomous operation of novel elevators for robot navigation. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 751–758. IEEE, 2010.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.

[LTJ+09] Zhaojin Lu, H.Q. Truong, Seokwoo Jang, T.B. Nguyen, and Sukhan Lee. Robust elevator button detection in completely reflective surfaces. In *Assembly and Manufacturing, 2009. ISAM 2009. IEEE International Symposium on*, pages 333–338, Nov 2009.

[Lum09] Jackson Guan Ting Lum. Pathfinders tech x dsta challenge - elevator control vision module : elevator button detection activation., 2009.

[MH04] Beate Meffert and Olaf Hochmuth. *Werkzeuge der Signalverarbeitung: Grundlagen, Anwendungsbeispiele, Übungsaufgaben*. Pearson Studium, München, 2004.

[MIS05] J. Miura, K. Iwase, and Y. Shirai. Interactive teaching of a mobile robot. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3378–3383, April 2005.

[ML99] Andrea Matessi and Luca Lombardi. Vanishing point detection in the hough transform space. In Patrick Amestoy, Philippe Berger, Michel Daydé, Daniel Ruiz, Iain Duff, Valérie Frayssé, and Luc Giraud, editors, *Euro-Parâ99 Parallel Processing*, volume 1685 of *Lecture Notes in Computer Science*, pages 987–994. Springer Berlin Heidelberg, 1999.

[Ots79] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Systems, Man and Cybernetics, IEEE Transactions on*, 9(1):62–66, Jan 1979.

[Ram72] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244 – 256, 1972.

[Som09] Martin Henning Hjalmar Sommerfeld. Klassifikation und präzise Lokalisation von Briefmarken mittels kantenbasierter Merkmale, 2009.

[SS85] Keiichi Abe Satoshi Suzuki. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.

[TSG+13] Daniel Troniak, Junaed Sattar, Ankur Gupta, James J. Little, Wesley P. Chan, Ergun Calisgan, Elizabeth A. Croft, and H. F. Machiel Van der Loos. Charlie rides the elevator - integrating vision, navigation and manipulation towards multi-floor robot locomotion. In *CRV*, pages 1–8. IEEE, 2013.

[Ver91] David Vernon. Machine vision-automated visual inspection and robot vision. *NASA STI/Recon Technical Report A*, 92:40499, 1991.

[Wik14] Wikipedia. Blindenschrift — Wikipedia, Die freie Enzyklopädie, 2014. `http://de.wikipedia.org/w/index.php?title=Blindenschrift&oldid=131376558`.

[WM01] P.F. Whelan and D. Molloy. *Machine Vision Algorithms in Java: Techniques and Implementation*. Springer London, 2001.

[YDLH09] Xinguo Yu, Li Dong, Liyuan Li, and Kah Eng Hoe. Lift-button detection and recognition for service robot in buildings. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 313–316, Nov 2009.

[ZG04] Philip G. Zimbardo and Richard J. Gerrig. *Psychologie*. Psychologie in Erziehung und Unterricht. Pearson Studium, München, 16., aktualisierte aufl. edition, 2004.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Bachelorarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 16. Juli 2014 .....