Hochschule Konstanz
Fakultät Informatik

Automatisiertes Erkennen von Polygonzügen aus Grundrissbildern

Julia Hansi

Konstanz, 21. Februar 2022

BACHELORARBEIT

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

an der

Hochschule Konstanz

Technik, Wirtschaft und Gestaltung

Fakultät Informatik

Studiengang Wirtschaftsinformatik

Thema: **Automatisiertes Erkennen von Polygonzügen aus Grundrissbildern**

Bachelorkandidatin: Julia Hansi
Am Langen Weg 3
2294 Groissenbrunn
Österreich

1. Prüferin: Prof. Dr. Doris Bohnet
2. Prüferin: Prof. Dr. Rebekka Axthelm

Ausgabedatum: 22. November 2021
Abgabedatum: 21. Februar 2022

Zusammenfassung

Der Gegenstand dieser Bachelorarbeit ist die automatisierte Extraktion von Polygonzügen anhand eines Grundrissbildes. Diese Polygonzüge sollen die Räumlichkeiten wiedergeben. In dieser Bachelorarbeit wurde daher ein Algorithmus für die Grundrissbildverarbeitung mittels Python entwickelt und implementiert. Zuerst wird ein Grundrissbild bereinigt, d. h. es werden unerwünschte Bildstrukturen verwaschen. Mithilfe des Canny-Kantendetektors werden anschließend die Kanten detektiert. Danach werden die Ecken im Grundrissbild via Harris-Eckendetektor lokalisiert. Um die Ecken sinnvoll zu verbinden, wird eine abgewandelte Form des Dijkstra Algorithmus herangezogen. Die daraus gewonnen Daten dienen zur Erstellung der Polygonzüge, welche für die Simulation von pFlow benötigt werden. Der entwickelte Algorithmus eignet sich insbesondere für klare und simple Grundrissbilder.

Schlagworte: Filter, Punktoperationen, Canny-Kantendetektor, Harris-Eckendetektor, Dijkstra Algorithmus, pFlow

Abstract

The subject of this bachelor thesis is the automated extraction of polygons from a floor plan image. These polygons are supposed to represent the rooms. Therefore, in this bachelor thesis an algorithm for floor plan image processing was developed and implemented by using Python. First, a floor plan image is cleaned up, i.e. unwanted image structures are washed out. Then the edges are detected by using the Canny edge detector. Afterwards, corners of the floor plan are localized with the help of the Harris corner detector. To connect the corners in a meaningful way, a modified form of Dijkstra's algorithm is used. The resulting data is used to create the polygons which are needed for the simulation of pFlow. The developed algorithm is especially suitable for clear and simple floor plan images.

Keywords: Filter, Point Operations, Canny Edge Detector, Harris Corner Detector, Dijkstra Algorithm, pFlow

Inhaltsverzeichnis

1 Einleitung

Der Trend geht dazu über, dass die Systeme immer autonomer werden (vgl. [18, Seite 27]). Solche autonomen Systeme können entsprechend der Situation reagieren und selbstständig – ohne der Hilfe des Menschen – Entscheidungen fällen, um ein festgelegtes Ziel zu erreichen.

Wenn so ein System anhand von Bilddaten urteilt, dann spielt die digitale Bildverarbeitung hierbei eine essenzielle Rolle. Für diese gibt es zahlreiche Anwendungsgebiete, welche häufig in Kombination mit Kameras stehen. Ein bekanntes Anwendungsszenario der digitalen Bildverarbeitung ist die Schrifterkennung (engl. *Optical Character Recognition (OCR)*), welche beim Einlesen von Zahlscheinen verwendet wird.

Das Ziel dieser Bachelorarbeit ist das automatisierte Extrahieren von Polygonzügen aus einem digitalen Grundrissbild. Dabei werden sowohl die theoretischen Grundlagen als auch die praxisnahe Anwendung der digitalen Bildverarbeitung behandelt. Daher bietet das gewählte Thema die Option, die Bildverarbeitungsgrundlagen kennenzulernen und im praktischen Teil zu vertiefen.

Ebenso werden im praktischen Teil Grundrissbilder für die Simulation von dem Projekt pFlow so vorverarbeitet, damit Polygonzüge durch die resultierenden Daten erstellt werden können. Da viele Algorithmen und Werkzeuge in diesem Bereich existieren, gilt es herauszufinden, welche davon und in welcher Reihenfolge diese eingesetzt werden, um die gewünschten Ergebnisse zu erzielen.

1.1 Projektvorstellung von pFlow

pFlow ist ein Tool, das die Personenbewegungen simuliert (vgl. [19]). Es soll speziell für die Simulation von Evakuierungsszenarien oder Notsituationen (z. B. bei Großveranstaltungen) eingesetzt werden. Damit wird evaluiert, ob ausreichend Notausgänge vorhanden und gut platziert sind.

Eine weitere mögliche Anwendung ist hierbei die Überprüfung für öffentliche Gebäude oder Supermärkte, ob eine bestimmte Anzahl von Personen diese Fläche optimal ausnutzt und dabei Abstandsregelungen eingehalten werden können.

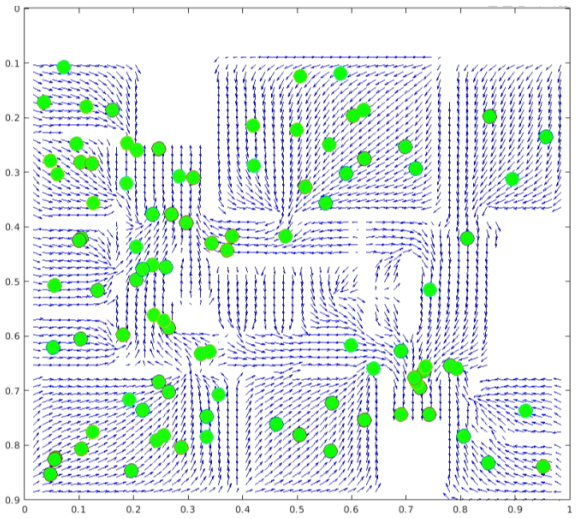

Abbildung 1: Schematische Simulation von pFlow. Die grünen Kreise stellen Personen dar. Die kleinen blauen Pfeile symbolisieren die Richtungsvektoren. (Quelle: [31, Seite 6])

Das Tool arbeitet mit Polygonzügen, welche den Grundriss des Gebäudes wiedergeben (vgl. [31, Seite 4-8]). Auf diesen Polygonzügen wird anschließend ein Dreiecksgitter mittels Triangulierung gespannt. Nachdem die Anzahl von Personen eingestellt ist, kann die Simulation gestartet werden. Während der Simulation werden Kennzahlen, wie z. B. Personendichte und Richtungsvektoren der Laufwege, aufgezeichnet (siehe Abbildung 1). Außerdem werden Flaschenhälse eines Grundrisses hervorgehoben.

1.2 Konkrete Aufgabenstellung und Ziele der Bachelorarbeit

Diese Bachelorarbeit soll auf die Bachelorarbeit von N. Steinbrügge (siehe [31]) aufbauen. Gegeben ist ein Bild, auf dem der Grundriss eines Gebäudes dargestellt ist. Um eine Personensimulation durchzuführen, soll der Algorithmus automatisiert Polygonzüge auf dem übergebenen Bild finden. Diese Polygonzüge sollen die Wände und Hindernisse aus dem Grundriss wiedergeben. Dabei wird sich mit der Forschungsfrage auseinandergesetzt, wie Polygonzüge aus einem Grundrissbild automatisiert ermittelt werden.

Ein weiteres Ziel dieser Bachelorarbeit ist es, auf der einen Seite dem Leser [1] ein gutes Verständnis im Bereich der Bildverarbeitung zu vermitteln und einen Einblick in die programmierte Erweiterung von pFlow zu bieten. Dies soll erreicht werden, indem auch auf der anderen Seite die allgemeinen Ziele einer wissenschaftlichen Arbeit verfolgt werden.

1.3 Aufbau der Bachelorarbeit

Im Kapitel 2 werden zunächst Fachbegriffe sowie Bestandteile der verwendeten Algorithmen definiert. Anschließend werden die verwendeten Algorithmen beschrieben. Danach folgt der Einblick in die Implementierung. Hierbei wird wie folgt vorgegangen:

1. **Vorverarbeitung:**
   Das übergebene Bild wird bereinigt, um Bildrauschen sowie ungewollte Texturen zu verwaschen.

2. **Kantendetektion:**
   Die Kanten auf dem Bild werden detektiert und ausgedünnt.

3. **Eckendetektion:**
   Anhand der detektierten Kanten werden die Ecken lokalisiert.

4. **Polygonzugerstellung:**
   Zuletzt werden die Polygonzüge anhand dieser Ecken erstellt.

Die generierten Polygonzüge werden an pFlow weitergegeben und anschließend verrechnet. Auf die genaue Vorgehensweise wird im Kapitel 3 näher eingegangen.

Zum Schluss wird die Bachelorarbeit mit einem Fazit und Ausblick beendet.

[1] In vorliegender Bachelorarbeit wird aus Gründen der inhaltlichen Fokussetzung und Funktionalität stellenweise die Sprachform des generischen Maskulinums verwendet. Dies geschieht überall dort, wo kein geschlechtsneutraler Begriff (z. B. „die Belegschaft“) zur Verfügung steht, eine lexikalische und grammatikalische Berücksichtigung beider Geschlechter jedoch die Aufmerksamkeit in eine falsche Richtung lenken würde (z. B. nicht: „der/die Bediener/-in der Anlage“). Das Maskulinum ist in diesen Fällen als sprachlich neutrale Form zu verstehen, die alle Geschlechteridentitäten gleichermaßen mit einschließt. Überall dort, wo Personen in ihrer Individualität gemeint sind, werden spezifische Formen verwendet (z. B. „die einzelnen Mitarbeiterinnen und Mitarbeiter“).

1.4 Auswahl der Werkzeuge

In diesem Unterkapitel wird die Materialbasis der Bachelorarbeit vorgestellt. Ein schneller Überblick der verwendeten Werkzeuge wird in Tabelle 1 gegeben.

Verwendetes Werkzeug	
Programmiersprache	Python (Version: 3.8)
Entwicklungsumgebung	PyCharm 2021.3.1 (Community Edition)
Bibliotheken	OpenCV, matplotlib, numpy
Betriebssystem	WINDOWS10

Tabelle 1: Auswahl der Werkzeuge

Als Programmiersprache wird Python verwendet, weil es eine einfache und intuitive Programmiersprache ist (vgl. [17, Seite XXXI]). Denn Python liest sich am ähnlichsten zur englischen Sprache im Gegensatz zu anderen Programmiersprachen. Dadurch kann der Entwickler die gewünschten Funktionen problemlos implementieren und der geschriebene Code ist auch für andere Entwickler einfacher zu lesen.

Die Einsatzgebiete von Python sind die wissenschaftliche Forschung sowie die Software-Entwicklung, aber auch Künstliche Intelligenz (KI), Webentwicklung uvm. (vgl. [22, Seite XV]).

Außerdem gibt es eine Vielzahl von Bibliotheken für Python, welche frei zugänglich sind. Für diese Bachelorarbeit wird die Bibliothek Open Source Computer Vision (OpenCV) verwendet, welche eine breite Palette an Bildverarbeitung-Funktionen mitliefert (siehe [20]). Ebenso wird die Bibliothek matplotlib für die Analyse und die Darstellung der Bilder verwendet. Beide Bibliotheken bauen auf numpy auf. Als Entwicklungsumgebung (engl. *Integrated Development Environment (IDE)*) wird PyCharm 2021.3.1 (Community Edition) eingesetzt.

Python bringt auch weitere Vorteile mit sich, die sich in den Prinzipien und der Philosophie von Python finden, welche in „The Zen of Python“ entweder bei Python Enhancement Proposals (PEP) 20 (siehe [29]) oder detaillierter bei J. Browning (siehe [4, Seite 1-13]) nachgelesen werden können.

Aufgrund des passenden Einsatzgebietes, Verwendungsmöglichkeit der passenden Bibliotheken, der Kombination von Python aus Einfachheit und Leistungsfähigkeit wird in dieser Bachelorarbeit auf Python zurückgegriffen (vgl. [17, Seite XXXI]).

2 Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, welche für das Verständnis nachfolgender Implementierung notwendig sind. Hierzu werden die verwendeten Fachbegriffe erklärt und die notwendigen mathematischen Grundlagen wiedergegeben.

Um aus einem gegebenen Grundrissbild die gewünschten Daten zu extrahieren, wird sowohl auf Filter, Punktoperationen als auch auf Algorithmen zurückgegriffen.

2.1 Bild

Es gibt viele verschiedene Definitionen für Bilder. Für diese Bachelorarbeit wird zwischen natürlichen und digitalen Bildern unterschieden.

Ein **natürliches Bild** kann etwas Abstraktes (z. B. Gemälde), aber auch eine Kopie, eine Ansicht (z. B. Grundriss) oder eine imaginäre Vorstellung von beliebig vielen Objekten (Bildmotiven) sein (vgl. [7]). Es ist eine Art von Abbildung, welche noch nicht digitalisiert ist.

Da mit *Grundrissbildern* in dieser Bachelorarbeit gearbeitet wird, werden diese nun näher beschrieben. Ein Grundriss ist eine technische Zeichnung, die die räumlichen Gegebenheiten eines Gebäudes zweidimensional und maßstabgerecht abbildet (vgl. [8]). Er wird horizontal normalerweise etwa einen Meter über dem Fußboden gelegt. Dadurch sind alle Wände, Stützen, Tür- und die Fensteröffnungen sichtbar. Ggf. können auch Wohnelemente (z. B. Sofa, Regale, Kühlschrank o. Ä.) eingezeichnet sein. Diese werden unter dem Begriff *Hindernisse* in dieser Bachelorarbeit zusammengefasst.

Wenn ein Grundriss auf dem Bildschirm eines Rechners sichtbar ist, wurde dieser von einem natürlichen in ein **digitales Bild** umgewandelt. Um natürliche und digitale Bilder genauer zu unterscheiden, wird zunächst in nachfolgenden Kapiteln beschrieben, was die Bestandteile eines digitalen Bildes sind und anschließend auf die verschiedenen Merkmale genauer eingegangen.

2.1.1 Pixel

Das Wort „Pixel“ (auch „Bildpunkt“ oder „Bildzelle“ genannt) entstand aus der Kombination **Pic**ture und **El**ement, was übersetzt Bildelement bedeutet (vgl. [5, Seite 16]). Ein Pixel ist ein Punkt in einem digitalen Bild (vgl. [10]). Wenn ein Gitter über ein Bild I gelegt wird, dann entspricht ein Pixel eine

Zelle in diesem Gitter. Daher wird ein Pixel vom Bild I in dieser Bachelorarbeit mit der Variable $I(u, v)$ beschrieben (siehe Abbildung 2). Die Variable u steht dabei für die Koordinate auf der x-Achse und v für die Koordinate auf der y-Achse des Bildes. Durch diese beiden Koordinaten wird die Position des Pixels eindeutig im Bild bestimmt. Er nimmt je nach Dateiformat zum Beispiel Grau-, Farb- oder Transparenzwerte an.

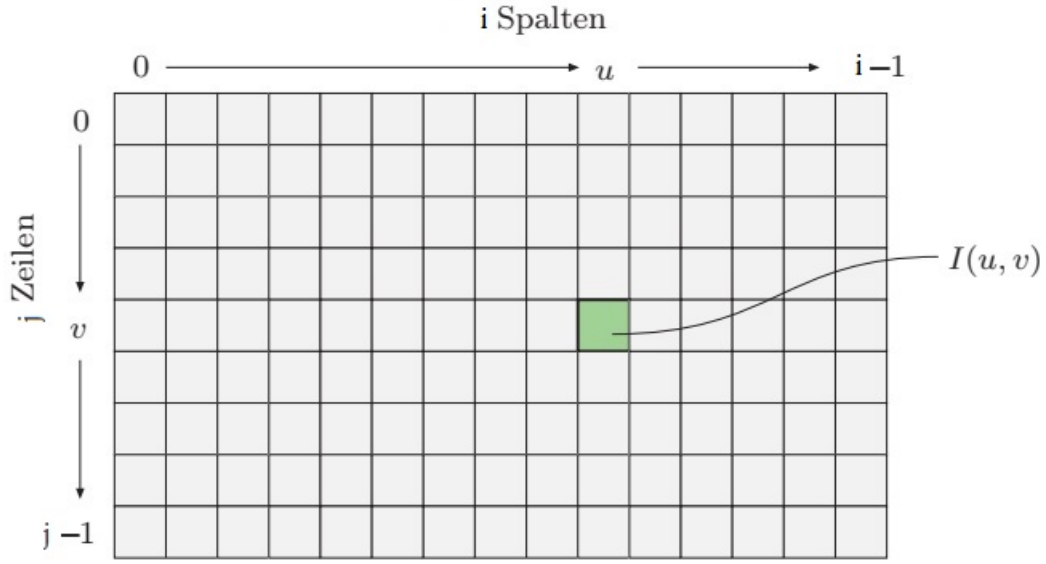

Abbildung 2: Ein Pixel $I(u, v)$ ist ein Bestandteil des Bildes I. Dabei entspricht u der Koordinate auf der x-Achse und nimmt einen Wert aus $[0; i-1]$ an. Die Variable i beschreibt die Anzahl Spalten, die das Bild breit ist. Hingegen die Variable v der Koordinate auf der y-Achse einen Wert aus $[0; j]$ annimmt. Dabei ist j die Anzahl Zeilen, die das Bild hoch ist. (In Anlehnung an: [6, Seite 11])

2.1.2 Intensitätswert eines Pixels

Ein Intensitätswert kann unterschiedliche Abstufungen annehmen. Im Fachjargon werden diese von der dunkelsten bis zur hellsten Abstufung wie folgt genannt (vgl. [37, Seite 276]) (siehe Abbildung 3): Schwarz, Tiefen (auch „Schatten“ genannt), dunkle Mitteltöne, Mitteltöne, helle Mitteltöne, Lichter und Weiß.

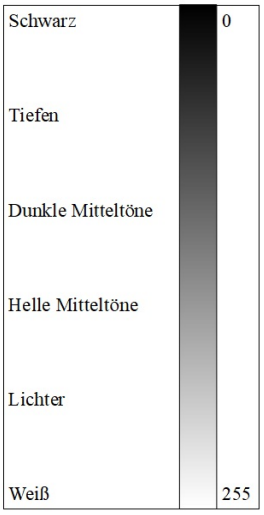

Abbildung 3: Links sind die Begriffe für Abstufungen der Intensitätswerte. Rechts davon sind die dazugehörigen Intensitätswerte $[0; 255]$ eingezeichnet.

Mit dem Intensitätswert (auch „Pixel-, Helligkeits- oder Grauwert“ genannt) wird beschrieben, wie dunkel (schwarz) oder hell (weiß) der Wert des Pixels ist. Der Intensitätswert wird „[...] durch das Verhältnis des Farbtons zu Schwarz oder Weiß bestimmt“. [5, Seite 71]

Die Intensitätswerte sind binäre Wörter der Länge n (vgl. [6, Seite 10-11]). Da das Binärsystem die Basis 2 hat, bedeutet dies, dass ein Pixel 2^n unterschiedliche Intensitätswerte annehmen kann. Der Intensitätswert eines Pixels befindet sich in einem Intervall $[0; 2^n - 1]$. Dieses Intervall wird als Intensitätsbereich (auch „Intensitätswertebereich“ oder „Bereich von Intensitätsstufen“ genannt) eines Bildes bezeichnet. Der Intensitätsbereich ist vom Bild-Dateiformat abhängig.

2.1.3 Linien, Kanten, Ecken und Konturen

In Grundrissbilder werden Wände und Hindernisse mit beliebig viele Linien eingezeichnet. Eine **Linie** ist auf beiden Seiten durch Kanten begrenzt (vgl. [27, Seite 164], siehe Abbildung 4). Diese beiden Kanten laufen weitgehend parallel zueinander.

Die **Kante** entsteht, wenn sich die Intensitätswerte auf kleinem Raum und entlang einer ausgeprägten Richtung stark ändert (siehe Abbildung 5). Je stärker diese Änderung ist, desto ausgeprägter ist die Kante. Im Bild selbst ist diese Kante als klare Linie ersichtlich.

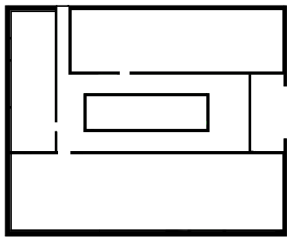

Abbildung 4: Ein Linienverlauf mit Verzweigungen und Unterbrechungen wird anhand eines simplen Grundrissbildes dargestellt.
(In Anlehnung an: Grundriss des Erdgeschosses vom O-Gebäude der Hochschule Konstanz Technik, Wirtschaft und Gestaltung (HTWG))

Treffen mindestens zwei Kanten aufeinander, so entstehen **Ecken** (vgl. [6, Seite 155-156]). **Konturen** hingegen stellen die Umrisse bzw. Silhouette von Objekten dar (vgl. [9]). Die Kanten und Konturen spielen eine wichtige Rolle im menschlichen Sehen (vgl. [6, Seite 125]). Schon mit wenigen Kanten können Bildmotive rekonstruiert werden. Daher soll mithilfe der Anwendung von geeigneten Kantendetektoren diese Kanten extrahiert werden (siehe Kapitel 2.8), um die Grundrissbilder wiederzugeben. Dadurch können die Wände und Hindernisse erkannt werden. Infolgedessen können Räume identifiziert und als Eingabe an pFlow weitergegeben werden.

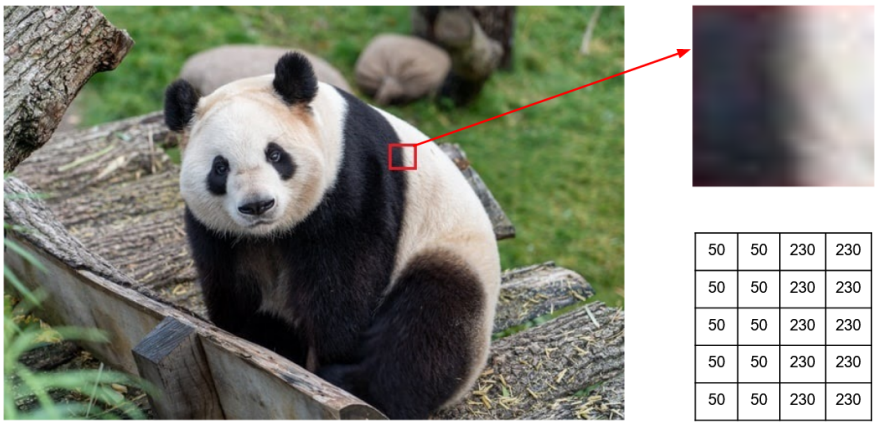

Abbildung 5: Dies ist ein Beispiel für eine Kante im digitalen Bild. Links ist das Originalbild angezeigt. Rechts oben ist der rot umrandete Bildausschnitt vergrößert dargestellt. Darunter befindet sich eine schematische Darstellung der Intensitätswerte als Matrix. (Quelle des Originalbildes: [36])

2.1.4 Bild-Dateiformat

Ein Dateiformat ist eine innere logische Struktur einer Datei (vgl. [5, Seite 23]). Grundsätzlich wird beim Dateiformat für digitale Bilder zwischen Vektorgrafik- und Rastergrafik-Formaten unterschieden (siehe Tabelle 2).

	Pixelgrafik	**Vektorgrafik**
Speicherplatz	Hoch (aber: Komprimierung möglich)	Wenig
Bildqualität beim Vergrößern/-kleinern des Bildes	Schlecht (verpixeltes Bild)	Gut
Bearbeitung	Einfach	Schwer
Kompatibilität	Hoch	Wenig

	PNG (Portable Network Graphics)	**JPEG** (Joint Photographic Experts Group)
Kompression	***Verlustfrei*** *(weniger Störungen)*	Verlustbehaftet

Tabelle 2: In der Tabelle werden die unterschiedlichen Merkmale zwischen den grundsätzlichen Bilddateiformaten Pixel- und Vektorgrafik aufgelistet. In der unteren Tabelle werden Pixelgrafik-Dateiformate Portable Network Graphics (PNG) und Joint Photographic Experts Group (JPEG) miteinander verglichen. Die rot umrandeten Begriffe zeigen die gefällten Entscheidungen bzgl. der Verwendung des Bilddateiformates für diese Bachelorarbeit auf.

Beim **Vektorgrafik-Format** (auch „vektorielles Format“ genannt) werden die Bestandteile des digitalen Bildes (z. B. Linienform) jeweils als eine eigene mathematische Gleichung gespeichert (vgl.[26, Seite 24]). Ein Vorteil hierbei ist, dass die Bildqualität beim Vergrößern oder Verkleinern des digitalen Bildes im Gegensatz zum Rastergrafik-Format nicht abnimmt (siehe Abbildung 6). Das bedeutet, dass das digitale Bild nicht verpixelt angezeigt, sondern aufgrund der Besonderheit des Vektorgrafik-Formats bei jedem Vergrößern und Verkleinern des digitalen Bildes auf dem Endgerät neu berechnet wird. Ein konkretes Beispiel für diese Dateiformat-Art ist Scalable Vector Graphics (SVG), welches u. a. für Logos auf Webseiten verwendet wird.

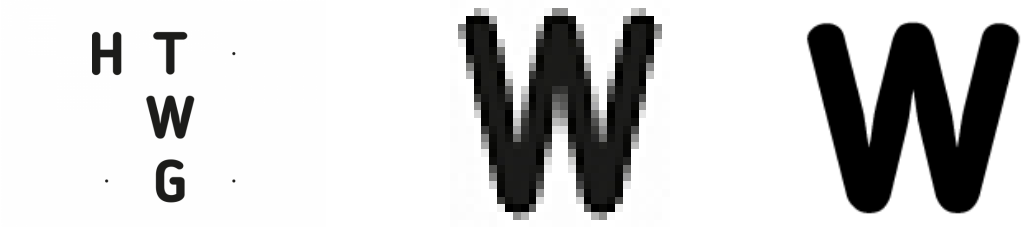

Abbildung 6: Vergleich des Verhaltens beim Vergrößern eines digitalen Bildes zwischen Pixelgrafik- und Vektorgrafik-Format. Links ist ein Bild vom Logo der HTWG. In der Mitte ist ein verpixeltes W ersichtlich, welches aus dem Bilddateiformat PNG resultiert. Hingegen auf der rechten Seite ist trotz Vergrößerung ein klar erkennbares W, welches mittels SVG dargestellt ist.

Die digitalen Bilder, die im **Rastergrafik-Format** abgespeichert werden, bestehen aus einem Gitter mit Pixeln. Daher ist dieses Format auch als Pixelformat bekannt. Mithilfe von diesem Dateiformat können auch detailreiche digitale Bilder (z. B. Fotos) abgespeichert werden. Allerdings benötigt es viel Speicherplatz, welcher mithilfe von geeigneten Kompressionsverfahren stark reduziert werden kann. Im Gegensatz zum Vektorgrafik-Format bietet das Rastergrafik-Format eine einfachere Bearbeitung des digitalen Bildes und ist bekannter.

Da das Rastergrafik-Format sich für die Bildverarbeitung gut eignet (vgl. [26, Seite 24]), wird im Zuge dieser Bachelorarbeit mit digitalen Bildern in diesem Format gearbeitet. Bei den oben erwähnten Kompressionsverfahren wird zwischen verlustfrei und -behaftet unterschieden. Um unnötige Störungen zu vermeiden, wird auf eine verlustfreie Kompression gesetzt. Dazu bietet sich das Bilddateiformat PNG an.

2.1.5 Unterschied zwischen natürlichen und digitalen Bild

Da das natürliche Bild stetig ist, enthält es unendliche viele Pixel sowie Intensitätswerte, d. h. es gibt keine Pixel in der Natur. Im digitalen Bild wird nur eine bestimmte Anzahl von Pixeln dargestellt, daher werden die Datenmengen vom natürlichen Bild bei der Umwandlung (auch „Transformation“ genannt) in ein digitales Bild begrenzt. Dadurch wird unter anderem das Speichern und Verarbeiten dieser Bilder im Rechner ermöglicht.

Im weiteren Verlauf dieser Bachelorarbeit wird mit „Bild“ das digitale Bild bezeichnet. Bei Bedarf oder Interesse kann der genaue Transformationsvorgang vom natürlichen bis zum digitalen Bild in der Literatur von W. Burger und M. J. Burge (siehe [6, Seite 4-10]) oder B. Neumann (siehe [26, Seite 21-26]) nachgelesen werden. Bei dieser Transformation wird jedem Pixel ein diskreter Intensitätswert zugeordnet. Je nach Bild-Dateiformat dürfen diese Intensitätswerte innerhalb eines bestimmten Intervalls liegen (siehe Tabelle 3):

Bilddateiformat	Intervall
Binärbild:	0= schwarz, 1 = weiß
Zweipegelbild:	z. B. rot, blau (vgl. [27, Seite 34])
Grauwertbild:	0=schwarz bis 255 = weiß
Farbbild:	meistens rot, grün, blau von jeweils [0, 255]

Tabelle 3: Digitale Bildformate

In dieser Bachelorarbeit werden 8-Bit-Grauwertbilder verwendet, daher sind 256 Intensitätswerte verfügbar. Abstrakt gesehen sind digitale Bilder einfache Matrizen (engl. *Arrays*), welche gelesen, verändert und gespeichert werden können.

2.2 Digitale Bildverarbeitung

Die digitale Bildverarbeitung dient zum Herausfiltern von speziellen Informationen, die das Bild bereitstellt (vgl. [6, Seite 3]).

Sie ist ein interdisziplinäres Feld, da viele Bereiche aus Mathematik, Informatik und Physik dort zusammentreffen (vgl. [21, Seite 16]).

Ein Teilbereich der digitalen Bildverarbeitung ist die industrielle Bildverarbeitung. Sie beschreibt die Simulation der visuellen Intelligenz des Menschen und das Verarbeiten von Bilddaten in einem Rechner, um damit technische Probleme zu lösen. Beliebte Aufgaben hierfür sind Mess- und Steuerungsaufgaben, die ohne Berührung durchgeführt werden, um z. B. Bohrlöcher in Werkstücken oder verschlossene Verpackungen zu prüfen. Dafür werden Kameras, Beleuchtungssysteme, Computer sowie Software zusammen verwendet (vgl. [13, Seite 5]).

2.2.1 Bildverarbeitungssystem

Ein Bildverarbeitungssystem löst anhand von eingehenden Bilddaten bestimmte Aufgabenstellungen. Dabei versucht es das menschliche Sehen zu imitieren. Dies wird dann „maschinelles Sehen“ genannt.

Ein industrielles Beispiel dafür ist ein Bildverarbeitungssystem, welches entscheidet, ob das aktuell gezeigte Bild in Ordnung oder fehlerhaft ist. Es findet Fehler bzw. Abweichungen entsprechend vorgegebener Kriterien. Diese Kriterien werden durch die digitale Bildverarbeitung definiert und können beispielsweise Farbabweichungen sein. Tritt ein unbekannter Fehler auf, der auch in die beschriebenen Kriterien passt, wird er übersehen (vgl. [13, Seite 8]).

Das Bildverarbeitungssystem bringt die Vorteile mit sich, dass es schneller und objektiv im Gegensatz zum Menschen entscheidet. Das bedeutet, es entscheidet ohne Emotion und kennt keine Müdigkeit.

Das Bildverarbeitungssystem erhält eine **Bildszene** mit beliebig vielen darzustellenden Objekten. Die Bildszene wird vom **Bildgeber** (z. B. eine Kamera) aufgenommen. Dabei wird mittels **Video-Interface** das analoge in ein digitales Signal transformiert. Die Bildszene wird z. B. beim Bildschirm des **Rechners** angezeigt. Die Abbildung 7 zeigt das Zusammenspiel dieser Bestandteile.

In dieser Bachelorarbeit wird ein Bildverarbeitungssystem implementiert, welches Polygonzüge anhand eines Grundrissbildes automatisiert erstellt.

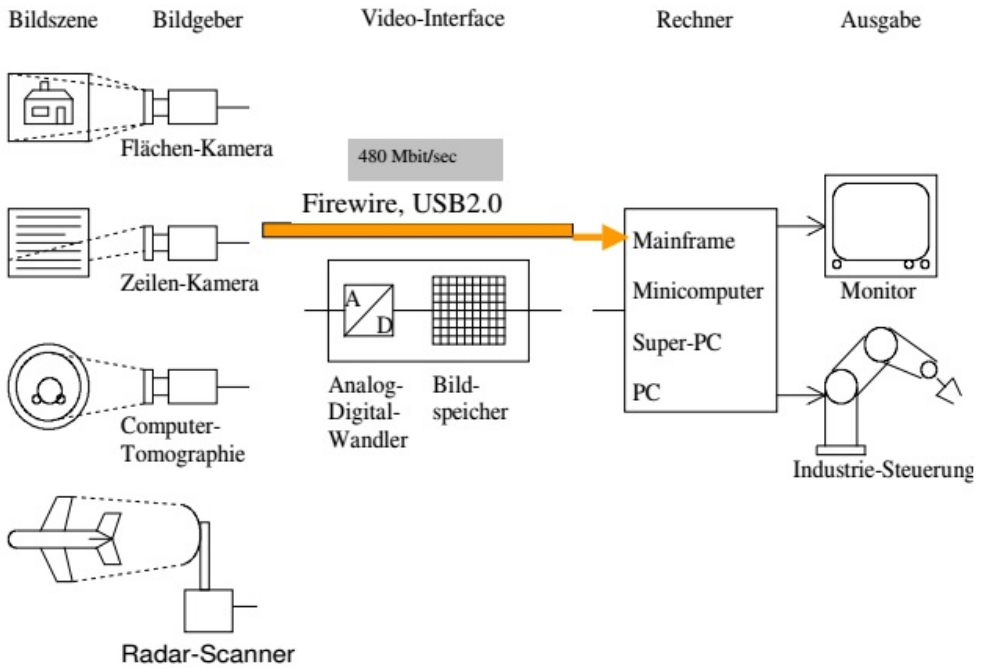

Abbildung 7: Aufbau eines Bildverarbeitungssystems
(Quelle: [13, Seite 21])

2.2.2 Prozesse

Ein Bildverarbeitungssystem umfasst viele Prozesse. Einige davon, auf die in dieser Bachelorarbeit näher eingegangen wird, sind (vgl. [6, Seite 2], [12, Seite 2]):

Bildtransformation
Die Bildtransformation beschreibt den Vorgang, wie ein natürliches in ein digitales Bild umgewandelt wird. Dies wurde bereits im Kapitel 2.1.5 beschrieben.

Bildbearbeitung
Bei der herkömmlichen Bildbearbeitung manipulieren Menschen schrittweise Bilder mittels Software wie z. B. Adobe Photoshop. Dieser Vorgang wird für die künstlerische Darstellung von einem oder mehreren Objekten eingesetzt. Dabei wird das Ziel verfolgt, Bilder fürs Auge zu verbessern und die Ästhetik des Bildes besser für den Betrachter zu vermitteln. Die Beispiele hierfür sind die Entfernung von roten Augen und Veränderung von Farben und Sättigung. Diese Bearbeitungsmöglichkeiten werden v. a. in der Werbeindustrie einge-

setzt. Die Werbung wird vor der Veröffentlichung digital bearbeitet, um für die Zielgruppe möglichst ansprechend zu sein. Ein weiteres Einsatzgebiet ist die Bildbearbeitungssoftware wie z. B. Foto-Anwendungen von Smartphones oder Adobe Photoshop.

Die Bildverarbeitung wendet auch die gleichen Verfahren wie die herkömmliche Bildbearbeitung an. Allerdings werden diese Verfahren bei der digitalen Bildverarbeitung automatisiert eingesetzt, z. B. mithilfe von Punktoperationen (siehe Kapitel 2.4) und Filter (siehe Kapitel 2.5).

Außerdem beschäftigt sich die Bildverarbeitung mit der Konzeption und Erstellung von Software (z. B. Entwicklung (oder Erweiterung) von Bildbearbeitungsprogrammen) und ist somit der Grundstein für jegliche digitale Bildbearbeitung.

Bildauswertung
Mithilfe der Bildauswertung wird das übergebene Bild analysiert. In diesem Prozess werden Histogramme und Kennzahlen erstellt, aber es können auch weitere Parameter daraus entnommen werden. Weitere Details dazu folgen im Kapitel 2.3.

2.3 Histogramm

Wenn ein Histogramm h auf das Bild angewendet wird, so repräsentiert es die Häufigkeit der einzelnen Intensitätswerte (vgl. [3, Seite 56]). Es gruppiert Daten in Eimer (auch „Klasse“ genannt, engl. *Bins*) i, welche in Balken visualisiert werden. Diese Klassen werden entlang der x-Achse aufgetragen. Auf der y-Achse befindet sich die Häufigkeit oder das Vorkommen $h(i)$. Ein hoher Balken bedeutet, dass die Werte in der betroffenen Klasse oft vorkommen, wohingegen ein niedriger Balken bei einer Klasse ein geringeres Auftreten anzeigt. Der Mittelwert ist Schwerpunkt des Histogramms.

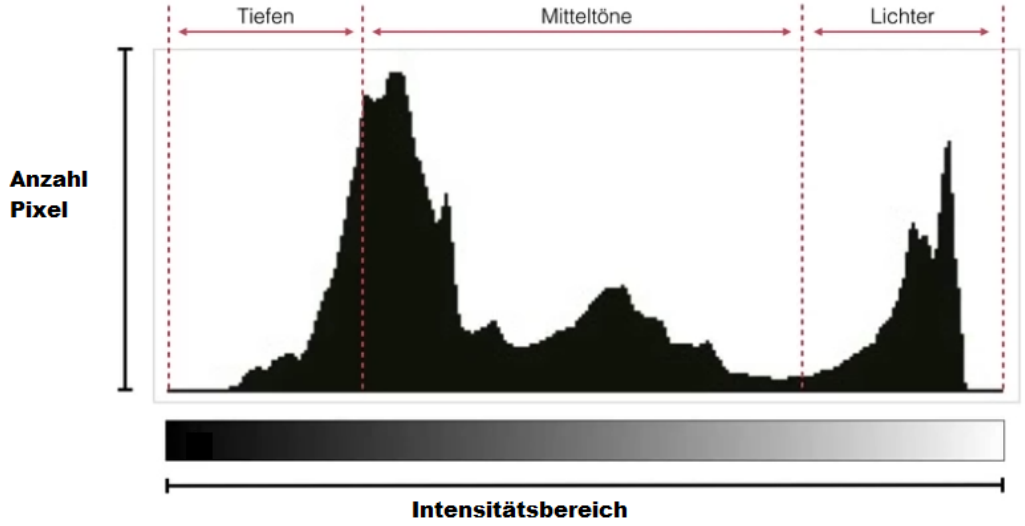

Abbildung 8: Ein beispielhaftes Histogramm mit Intensitätswerten auf der x-Achse und Häufigkeit der Pixel auf der y-Achse.

Angenommen, es wird ein 8-Bit-Grauwertbild im Histogramm abgebildet. So hat dieses bis zu k Bins, wobei gilt $k = 2^n = 2^8 = 256$. Wenn k Bins im Histogramm vorhanden sind, stellen diese jeweils einen Intensitätswert dar. Allerdings kann auch konfiguriert werden, dass weniger als k Bins im Histogramm in Verwendung sind. Dabei fallen mehrere Intensitätswerte zu einer Klasse i zusammen.

Es gibt jeweils ein Zähler $h(i)$ pro Klasse, welcher im Histogramm die Balkengröße der jeweiligen Klasse anzeigt. Jedes Pixel $I(u, v)$ wird im Bild I durchlaufen. Während diesen Vorgang wird für jedes Pixel der Zähler $h(i)$ von der entsprechenden Klasse aufgerufen und um eins erhöht.

Im resultierenden Histogramm sind die Intensitätswerte von der dunkelsten auf der linken Seite bis zur hellsten Abstufung auf der rechten Seite von

der x-Achse sortiert (siehe Abbildung 8). Im mittleren Bereich entlang der x-Achse des Histogramms befinden sich die Mitteltöne. Die x-Achse ist demnach eine Intensitätswerte-Skala. Auf der y-Achse wird beschrieben, wie oft die jeweiligen Intensitätswerte im Bild enthalten sind.

Wird nun beispielsweise die Klasse $h(255)$ näher betrachtet, so kann die Anzahl aller weißen Pixel, welche den Intensitätswert $255(= k - 1)$ haben, herausgelesen werden. Je nach Konfiguration des Histogramms können absolute oder relative Häufigkeiten dargestellt werden. Die Form des Histogramms bleibt dadurch unverändert.

In den nachfolgenden Unterkapiteln werden einige der Verwendungszwecke eines Histogramms beschrieben. Für diese Bachelorarbeit werden Histogramme für die Analyse der Bildeigenschaften verwendet.

2.3.1 Informationsgehalt eines Histogramms

Die Hauptaufgabe eines Histogramms ist es, nur wichtige Informationen eines Bildes so kompakt wie möglich darzustellen. Deswegen enthält es beispielsweise die Information nicht, woher die einzelnen Einträge stammen. Somit ist es nicht möglich, das Originalbild auf Basis eines Histogramms zu rekonstruieren (vgl. [6, Seite 42-43]). In der Abbildung 9 werden Bilder gezeigt, die eine unterschiedliche Anordnung von Pixeln haben, aber das gleiche Histogramm aufweisen.

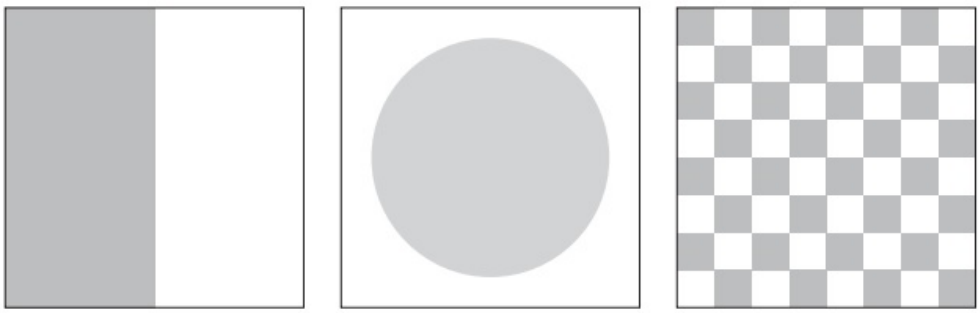

Abbildung 9: Diese verschiedenen Bilder weisen das gleiche Histogramm auf. (Quelle: [6, Seite 42])

2.3.2 Beurteilung von Bildeigenschaften

Die Histogramme eignen sich als Hilfsmittel für die Beurteilung von Bildeigenschaften bei der Bildaufnahme. Diese werden in diesem Kapitel erklärt und auch auf deren Ausprägungen in Histogrammen näher eingegangen.

Belichtung

Das Histogramm stellt die Belichtung eines Bildes dar.

Es macht aber keine Aussage darüber, ob das Bild gut oder schlecht belichtet ist. Denn um diese Aussage treffen zu können, müssen entsprechende Kriterien abhängig von den Bildmotiven und Bildwirkung beachtet werden.

Beispielsweise ist ein Bild von einer Schneelandschaft ein extrem helles Bild, muss es aber auch sein, sonst würde es nicht wirken (siehe Abbildung 10).

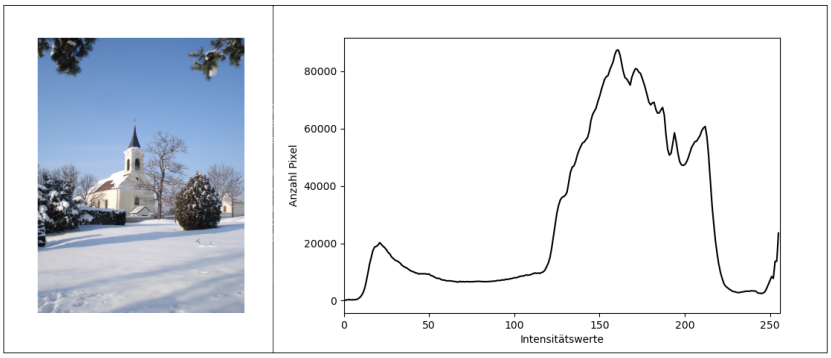

Abbildung 10: Auf der linken Seite wird ein Bild von einer Kirche in einer Schneelandschaft mit blauen Himmel gezeigt. Auf der rechten Seite wird das dazugehörige Histogramm gezeigt. Da das Bild in der Natur entstanden, aber der Schnee die natürliche Umgebung bedeckt, ist im Histogramm keine Gauß-Glocke, sondern eine Häufung von hellen Intensitätswerte, ersichtlich.. (Bild links wurde am 4. Dezember 2010 in Groissenbrunn aufgenommen. Histogramm rechts wurde in Python3 erstellt.)

Bei einer idealen Belichtung in einer natürlichen Bildszene wird am Anfang und Ende der x-Achse beim Histogramm die Intensitätswerte eher weniger und die Intensitätswerte im mittleren Bereich der x-Achse vermehrt genutzt. Dabei gleicht die Verteilung einer Gaußschen Glockenkurve (siehe Abbildung 11).

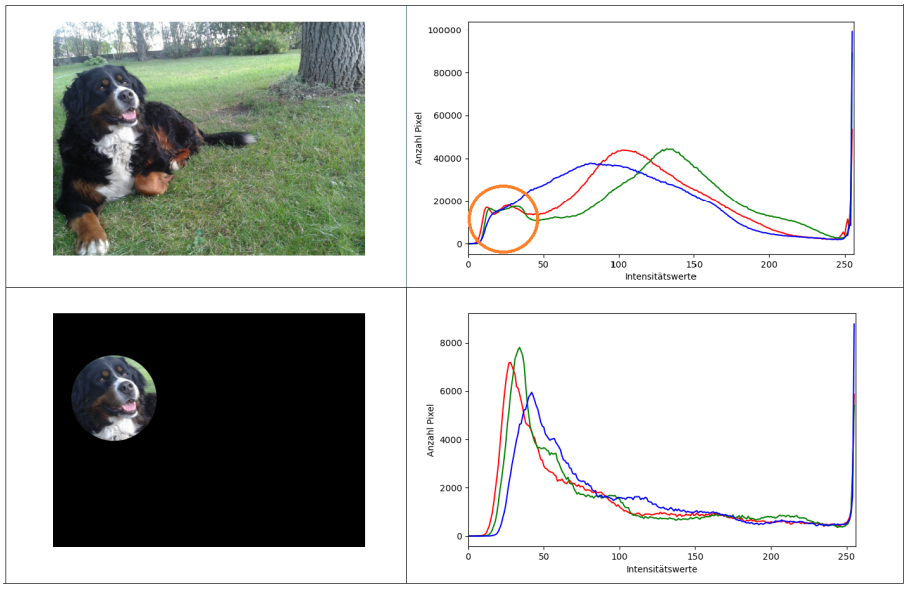

Abbildung 11: Links sind natürliche Bilder mit einem Hund im Garten zu sehen. Rechts davon ist das dazugehörige Histogramm. Darauf ist eine Gaußsche Glockenkurve ersichtlich, welche einen ungewöhnlichen Verlauf aufweist (siehe orange eingekreister Bereich). Links unten in der 2. Reihe der Abbildung wird über das Gesicht des Hundes eine Maske gelegt. Im dazugehörigen Histogramm rechts unten wird nun die Ursache für den ungewöhnlichen Verlauf dargestellt. (Bild links oben wurde am 1. April 2013 in Groissenbrunn aufgenommen. Histogramme rechts wurde in Python3 erstellt.)

Bei der Bildaufnahme können auch Belichtungsfehler entstehen. Dabei können folgende Arten von Belichtungsfehlern am Histogramm ausfindig gemacht werden:

- *Unterbelichtetes Bild*: Intensitätswerte am Ende der x-Achse sind ungenutzt, aber am Anfang der x-Achse werden die Intensitätswerte vermehrt genutzt.
- *Überbelichtetes Bild*: Intensitätswerte am Anfang der x-Achse sind ungenutzt, aber am Ende der x-Achse werden die Intensitätswerte vermehrt genutzt.

Licht ist ein entscheidendes Element für die Abbildung von Informationen im Bild (vgl. [15, Seite 3]). Zu viel oder zu wenig Licht kann zu Informationsverlust führen.

Kontrast

Der Kontrast gibt den tatsächlich genutzten Intensitätsbereich des Bildes an (vgl. [6, Seite 44]).

Das bedeutet, der Kontrast a ist die Differenz zwischen größten a_{max} und kleinsten a_{min} Intensitätswert, der im Bild vorkommt.

Der kleinste Intensitätswert a_{min} ist der erste Balken von links auf der x-Achse im Histogramm. Dieser Wert ist der dunkelste Intensitätswert im Bild.

Der größte Intensitätswert a_{max} ist der letzte Balken auf der x-Achse, welcher ganz rechts im Histogramm eingezeichnet ist. Dieser Wert ist zugleich der hellste Intensitätswert, der im Bild existiert.

Im Idealfall hat das Bild einen hohen Kontrast. Das bedeutet, dass potentiell der gesamte Intensitätsbereich genutzt werden kann. Dann gilt: $a = [a_{min}; a_{max}] = [0; k-1]$.

Wenn der Kontrast eines Bildes erhöht wird, um den Intensitätsbereich voll auszuschöpfen, dann wird in Folge das Bild schärfer (siehe Abbildung 12). Der Kontrast kann so weit erhöht werden, bis der maximale Wertebereich erreicht ist.

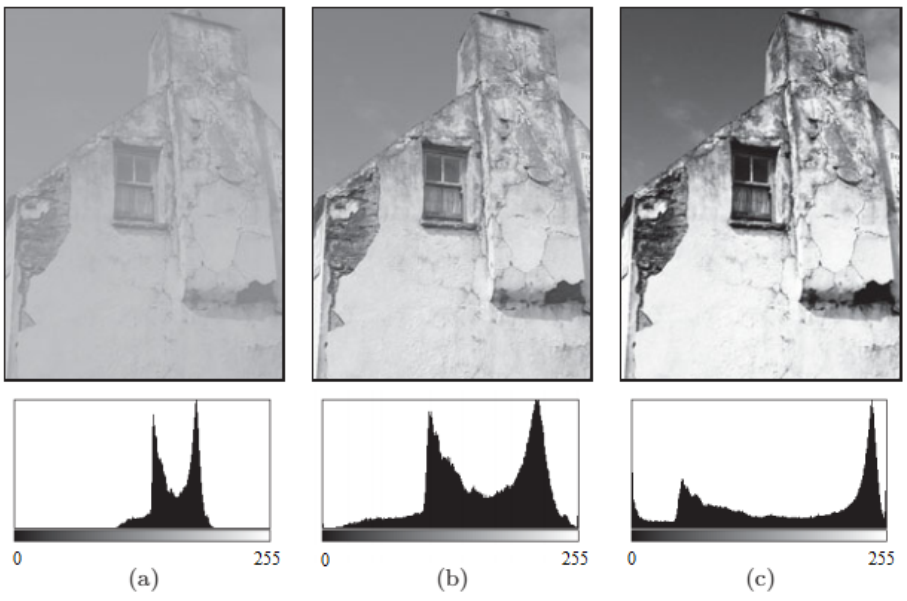

Abbildung 12: Unterschiedliche Kontrastabstufungen im Histogramm. Links wird das Bild (a) mit niedrigen Kontrast, in der Mitte (b) mit normalen und rechts (c) mit hohen Kontrast und mit dem dazugehörigen Histogramm dargestellt. (In Anlehnung an: [6, Seite 45])

Die Histogramme können auch Bildfehler (z. B. Spitzen und Löcher) aufzeigen, die während der Bildaufnahme oder nachfolgenden Bearbeitungsschritten entstanden sind. Sowohl Spitzen als auch Löcher können bei Komprimierung der Bilddaten entstehen.

Wird Kontrast bei einem Bild erhöht, so werden die Balken (Intensitätswerte) im Histogramm auseinander gezogen. Dadurch wird die Verteilung breiter und es entstehen sogenannte Löcher. Mithilfe Interpolation besteht die Möglichkeit, Löcher bei der Kontrasterhöhung zu vermeiden.

Wird der Kontrast bei einem Bild verringert, so sind Spitzen sichtbar, da die unterschiedlichen Intensitätswerte zusammenfallen und die zugehörigen Histogrammeinträge erhöhen (vgl. [6, Seite 47]). Wenn beispielsweise durch die Kontrastverringerung zwei bisher verschiedene Intensitätswert-Balken im Histogramm zusammenfallen, dann ergibt sich nun ein Intensitätswert-Balken. Aus dem resultierenden Balken können die ursprünglichen Intensitätswerte nicht mehr herausgefiltert werden, da Dynamik und Bildinformation verloren gegangen sind.

Dynamik

Die Dynamik gibt die Anzahl verschiedener Intensitätswerte an, die in einem Bild vorkommen (vgl. [6, Seite 44-45]).

Für den Idealfall wird eine volle Dynamik bevorzugt. Diese ist vorhanden, wenn der gesamte Intensitätsbereich voll ausgeschöpft wird. Das bedeutet, dass die Dynamik der Anzahl von Intensitätswerten k entspricht und alle Intensitätswerte des Intensitätsbereich im Bild verwendet werden. Somit ist die volle Dynamik gleich dem Kontrast a und gleich dem Intensitätsbereich des Bildes, d. h. $[a_{min}; a_{max}] = [0; k-1]$.

Generell ist ein Bild mit hohen Kontrast und hoher Dynamik wünschenswert, da dann die meiste Bildinformation im Bild enthalten ist.

Bildrauschen

Beim Bildrauschen sind störende Bildpartikel (auch „Artefakte“ genannt) im Bild, welche die Qualität des Bildes verringern (siehe Abbildung 13). Dieses kann beispielsweise von Kamera-Sensoren oder unvorteilhafter Belichtung verursacht werden. Um das Bildrauschen zu vermeiden, gibt es Möglichkeiten, die beim Bildentstehungsprozess beachtet werden sollten (siehe [15, Seite 31-36]).

Sollte dennoch Rauschen ins Bild gelangen, wie z. B. durch verlustbehaftete Kompressionsverfahren, so kann dieses mittels Bildverarbeitung verringert werden. Im Histogramm allein ist das Bildrauschen in der Regel nicht erkennbar.

In dieser Bachelorarbeit wird bestehendes Bildrauschen, welches beispielsweise durch das Einlesen des Grundrissbildes mittels Scanner entstehen kann, durch die Anwendung von Filtern reduziert. Weitere Details folgen im Kapitel 2.5.

Abbildung 13: Links ist das Originalbild von Lena Forsén[2]. Auf dem rechten Bild ist ein Bildrauschen hinzugefügt worden. (Quelle für Bild links: [38]. Eigene Darstellung für Bild rechts frei nach: ebd.)

[2]Es ist ein Standardbild, welches in der Bildverarbeitung zum Testen häufig benutzt wird.

2.4 Punktoperation

Die Punktoperationen sind Operationen auf Bilder, die nur die Werte der einzelnen Bildpunkte betreffen, ohne die Größe, Geometrie noch die lokale Bildstruktur zu ändern.

Grundsätzlich gibt es bei Punktoperationen zwei Arten: homogene und nicht homogene Punktoperationen. Diese werden in den nachfolgenden Unterkapiteln genauer beschrieben.

2.4.1 Homogene Punktoperation

Eine homogene Punktoperation ist nur abhängig vom Intensitätswert und unabhängig von der Bildkoordinate (u, v). Es wird dabei eine Funktion f auf jedes Pixel $I(u, v)$ im Bild I angewendet und das Ergebnis von $f(I(u, v))$ wird in dem jeweiligen Pixel gespeichert.

Eine homogene Punktoperationen wird immer auf das gesamte Bild gleichermaßen angewendet. Die Beispiele hierfür sind:

- Farbtransformation,
- Schwellenwertoperation,
- Invertierung des Bildes,
- Kontrast- und Helligkeitsanpassung usw.

2.4.2 Nicht homogene Punktoperation

Nicht homogene Punktoperationen sind im Gegensatz zu homogenen Punktoperationen abhängig von der Bildkoordinate (u, v). Es wird dabei eine Funktion g auf ein bestimmtes Pixel $I(u, v)$ im Bild I angewendet und das Ergebnis von $g(I(u, v), u, v)$ wird in dem jeweiligen Pixel gespeichert.

Beispiele für die Anwendung von nicht homogenen Punktoperation sind:

- Farbtransformation eines bestimmten Bildbereichs,
- Verblassen oder Verdunkeln des Bildes bis zum Bildrand,
- Kontrast- oder Helligkeitsanpassung in einem bestimmten Bildbereich usw.

2.4.3 Anwendungsbeispiele

Für Punktoperationen gibt es zahlreiche Anwendungsbeispiele. Die für diese Bachelorarbeit verwendeten Punktoperationen werden nachfolgend aufgelistet (vgl. [6, Seite 60-62]).

Anpassung von Intensitätswerten
Hierbei kann Kontrast, Helligkeit und Farbraum vom Pixel verändert werden. Der Kontrast kann prozentual erhöht oder verringert werden, d. h. der Intensitätswert wird mit einem Faktor multipliziert. Das Ergebnis wird immer auf den nächsten ganzzahligen Wert gerundet.
Die Helligkeit kann beliebig erhöht bzw. verringert werden, d.h. der Intensitätswert wird mit einem Wert addiert bzw. subtrahiert.

Auch der Farbraum kann verändert werden, z. B. RGB-Bild wird mithilfe der Formel in ein Grauwertbild umgewandelt:

$$0.299 \cdot rot + 0.587 \cdot grün + 0.114 \cdot blau$$

Dies wird bei der Bildvorverarbeitung in dieser Bachelorarbeit verwendet, um das übergebene Grundrissbild in ein Grauwertbild zu transformieren (siehe Kapitel 3.1).

Beschränkung der Ergebniswerte
Bei der Beschränkung von den Ergebniswerten (engl. *Clamping*) werden die Unter- und Obergrenze des Intensitätsbereichs begrenzt. Das bedeutet konkret, dass der Maximalwert nicht die Obergrenze überschreiten und der Minimalwert nicht die Untergrenze unterschreiten darf. Bei einem 8-Bit-Grauwertbild lautet die Untergrenze null und die Obergrenze $255 (= K - 1)$. So muss beispielsweise bei Helligkeitserhöhung nur die Obergrenze sowie bei Helligkeitsverminderung nur die Untergrenze limitiert werden.

Schwellenwertoperation
Bei der Schwellenwertoperation (engl. *Thresholding*) werden Bildwerte in zwei Klassen a_0, a_1 aufgeteilt (vgl. [6, Seite 61-62]). Der Schwellenwert (engl. *Threshold*) q ist ein Grenzwert, der bestimmt, ob der Intensitätswert des Pixels $I(u, v)$ in die erste a_0 oder zweite a_1 Klasse kommt. Die Formel hierzu lautet wie folgt:

$$f_{threshold}(I(u,v)) = \begin{cases} a_0, & I(u,v) < q \\ a_1, & I(u,v) \geq q \end{cases}$$

Bei der Binarisierung von Grauwertbildern ergeben sich somit die Klassen $a_0 = 0$ und $a_1 = 1$. Wenn der Schwellenwert überschritten wird, so ändert der Intensitätswert sich in die Farbe schwarz, sonst bleibt er weiß (vgl. [26, Seite 43]).

Eine andere Art der Schwellenwertoperation wird vom Canny-Kantendetektor verwendet, welche im Kapitel 2.8.1 näher beschrieben wird.

2.5 Filter

Die Filter (auch „Nachbarschaftsoperatoren“ genannt) sind in der Bildverarbeitung ein viel genutztes Werkzeug. Sie werden für die Kantendetektion sowie Bildverbesserung verwendet (vgl. [12, Seite 144]). Die Filter werden in der Vorverarbeitung bei dieser Bachelorarbeit angewendet, um das Bild zu glätten. D. h. im Grundrissbild werden Beschriftungen verwaschen und das Bildrauschen reduziert. Ebenso werden Filter auch verwendet, um Kanten zu finden. Die Bestandteile eines Filters sind:

- Quellbild $I(u, v)$ (auch „ursprüngliche Bild“ oder „Eingangsbild“ genannt)
- Filtermatrix M (auch „Kernel“ oder „Filtermaske“ genannt)
- Filteroperation $O(i, j)$
- Zielbild $I'(u, v)$ (auch „Ergebnisbild“ genannt)

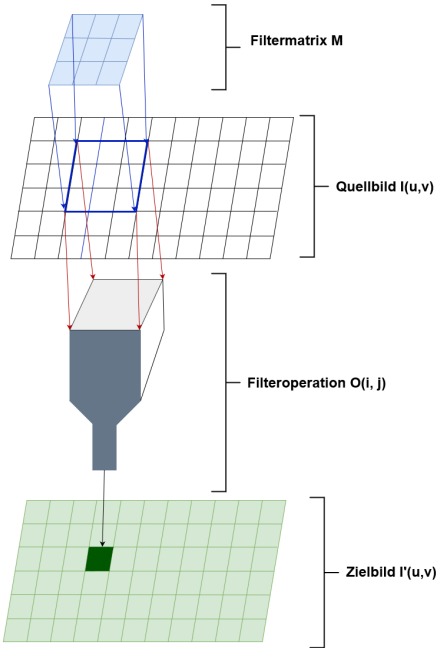

Abbildung 14: Funktionsweise eines Filters. Die Filtermatrix wird auf das Quellbild angewendet. Die überlappenden Werte (Intensitätswerte von Quellbild und Koeffizienten der Filtermatrix) werden mit der Filteroperation entsprechend der Filterart behandelt und das resultierende Zielpixel wird im Zielbild eingefügt.

Die Funktionsweise eines Filters wird in der Abbildung 14 grafisch dargestellt und nachfolgend textuell erklärt.

Das **Quellbild** $I(u, v)$ ist das ursprüngliche Bild, welches der Filteroperation übergeben wird.

Die **Filtermatrix** M bestimmt die Größe jener Region, die zum Berechnen eines neuen Intensitätswerts verwendet wird (siehe Abbildung 15). In der Filtermatrix selbst sind Koeffizienten (auch „Konstanten“ oder „Gewichte“ genannt) eingetragen, die positiv, negativ oder null sind. Die Filtermatrix hat ein eigenes Koordinatensystem (vgl. [6, Seite 96]). Im Mittelpunkt $M(0, 0)$ der Filtermatrix befindet sich der sogenannte Hotspot (auch „Filterkern“ genannt). Dieser ist sowohl vertikal als auch horizontal gesehen das mittlere Pixel, wenn die Größe der Filtermatrix ungerade ist.

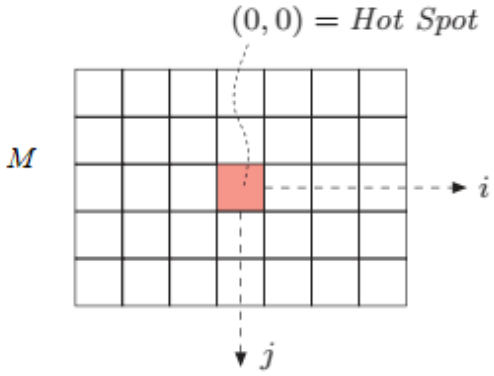

Abbildung 15: Eine allgemeine Filtermatrix M mit dem Hotspot in der Mitte.
(Quelle: [6, Seite 96])

Bei Größe der Filtermatrix wird die Anzahl Reihen i_{max} und Spalten j_{max} festgelegt (vgl. [6, Seite 101]). In der Praxis werden oft 3×3 (auch „3 kreuz 3“ oder „3 mal 3“ genannt) Filtermatrizen verwendet. Allerdings sind andere beliebige Filtermatrixgrößen auch möglich. Diese sollten idealerweise entsprechend dem Anwendungsfall ausgewählt werden und ungerade sein, um den Hotspot einfacher zu identifizieren. Je größer die Filtermatrix gewählt wird, desto mehr Werte werden in die Berechnung des Zielpixels miteinbezogen. In dieser Bachelorarbeit werden unterschiedliche Filtermatrixgrößen verwendet, welche dann in der Implementierung im Kapitel 3 genauer definiert werden.

Die Filtermatrix wird über das gesamte Quellbild gezogen und währenddessen wird die **Filteroperation** $O(i, j)$ durchgeführt. Diese Filteroperation führt entsprechend ihrem Typus Berechnungen auf den Intensitätswerten im Bild abhängig von den Koeffizienten, die sich in der Filtermatrix befinden, aus.

Das Ergebnis der Filteroperation wird ins **Zielbild** $I'(u, v)$ gespeichert. Dabei gilt es zu beachten, dass das Ergebnis auf den korrekten Intensitätsbereich begrenzt wird (Clamping, siehe Kapitel 2.4.3). Wenn die Filtermatrix über den Rand des Bildes hinaus platziert wird, dann wird bei der Filteroperation die Randbehandlung für die Berechnung des Ergebnisses angewendet. Auf die möglichen Randbehandlungen wird im Kapitel 2.5.3 näher eingegangen.

Eine Eigenschaft des Filters ist die Isotropie. Ein Filter wird als „isotrop“ bezeichnet, wenn er in alle Richtungen gleichförmig arbeitet und somit keine Richtung bevorzugt (vgl. [21, Seite 336]).

Die Unterschiede zwischen Filter und Punktoperation werden in der Tabelle 4 zusammengefasst dargestellt.

Merkmale	Filter	Punktoperation
Geometrie des Bildes wird verändert.	Ja, kann verändert werden.	Nein.
Glättung und Schärfung vom Bild wird ermöglicht.	Ja.	Nein.
Berechnung des Ergebnisses ist abhängig ...	... von der Filtermatrix (Hotspot und benachbarte Pixel).	... vom Pixel (Punkt).
Pixel im Quellbild werden während der Berechnung überschrieben und gespeichert.	Nein, es wird ein Zwischenschritt benötigt.	Ja.

Tabelle 4: Unterschied zwischen Filter und Punktoperation

Da die Bibliothek OpenCV selbstständig das Problem mit der Zwischenspeicherung des Ergebnisses löst, wird hier nicht weiter darauf eingegangen. Weitere Details hierzu können bei W. Burger und M. J. Burge (siehe [6, Seite 98]) oder in der OpenCV Dokumentation (siehe [28]) nachgelesen werden.

Es gibt zwei Arten von Filtern (lineare und nichtlineare Filter), welche in den nachfolgenden Unterkapiteln beschrieben werden.

2.5.1 Lineare Filter

Beim linearen Filter werden die Intensitätswerte des Quellbildes und die Koeffizienten der Filtermatrix mit einer linearen Funktion verknüpft.
Dabei wird wie folgt vorgegangen (vgl. [6, Seite 96-97]):

1. Im ersten Schritt wird die Filtermatrix $M(i, j)$ mit der Filterfunktion über das Quellbild $I(u, v)$ gelegt, sodass der Hotspot von der Filtermatrix über der aktuellen Bildkoordinate $I(u, v)$ liegt.

2. Es werden alle Koeffizienten der Filtermatrix $M(i, j)$ mit jeweils dem korrespondierenden Pixel multipliziert.

3. Danach wird die Summe der Produkte aus vorherigen Schritt gebildet und in dem Pixel des Zielbilds $I'(u, v)$ gespeichert. Dieses Pixel hat dieselben Koordinaten wie jenes Pixel im Quellbild, das im Hotspot der Filtermatrix liegt.

Dieser Vorgang wird für jedes Pixel im Quellbild wiederholt. Die Formel für den beschriebenen Vorgang lautet wie folgt (vgl. ebd.):

$$I'(u, v) = \sum_{(i,j)\in R} I(u + i, v + j) \cdot M(i, j)$$

Die Variable R entspricht den Koordinaten der Filterregion. Bei einer Filtermatrixgröße von 3×3 ergibt sich die folgende Formel (vgl. ebd.):

$$I'(u, v) = \sum_{i,=-1}^{i=1} \sum_{j=-1}^{j=1} I(u + i, v + j) \cdot M(i, j)$$

Die linearen Filter werden u. a. in Tief- und Hochpassfilter kategorisiert.

Tiefpassfilter
Die Tiefpassfilter (auch „Glättungsfilter“ genannt, engl. *Blur*) vermindern Bildrauschen sowie die Schärfe von Grauwertkanten und -spitzen (vgl. [26, Seite 54]). Dadurch wird eine Glättung des Bildes herbeigeführt, was zum Verlust von Bilddetails führt.

Ein weiteres Merkmal von Tiefpassfiltern ist, dass sie nur positive Koeffizienten enthalten (vgl. [6, Seite 95]). Somit hat jedes Pixel positiven Einfluss auf die Berechnung und daher ist ein Ergebniswert unter null nicht erreichbar.

Für die Filtermatrix von Tiefpassfilter sollte eine möglichst runde Form bevorzugt werden, da diese isotrop wirkt.

Da der Box-Filter als der einfachste Tiefpassfilter gilt, wird zunächst dieser erläutert. Anschließend wird der Gauß-Filter beschrieben, da dieser eine höhere Komplexität aufweist.

Der ***Box-Filter*** (auch „Durchschnittsfilter“ oder „Mittelwertfilter“ genannt) ist nach seiner Form der Filtermatrix benannt. Für die Berechnung des Box-Filters wird das (einfache) arithmetische Mittel (auch „Mittelwert“ genannt, engl. *Mean*) unter der Filtermatrix M gebildet.

Die Formel für das (einfache) arithmetische Mittel lautet wie folgt (vgl. [32, Seite 52-53]):

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Die Variable n ist hier die Anzahl der Elemente $x_1, ..., xn$, die in die Berechnung des Mittelwerts mit einfließen.

Bei einem 3×3 Box-Filter sieht die zugehörige Filtermatrix M wie folgt aus (vgl. [12, Seite 145]):

$$M = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Die Formel mit relativen Koordinaten für einem 3×3 Box-Filter lautet daher:

$$\begin{aligned} I'(u,v) = \frac{1}{9} \cdot [&I(u-1, v-1) + I(u, v-1) + I(u+1, v-1) + \\ &I(u-1, v) + I(u, v) + I(u+1, v) + \\ &I(u-1, v+1) + I(u, v+1) + I(u+1, v+1)] \end{aligned}$$

Nach der Berechnung wird anschließend das Ergebnis im Zielpixel $I'(u, v)$ eingetragen.

Für den Box-Filter wird meist aufgrund der Symmetrie eine quadratische Form mit 3×3, 5×5, 7×7 oder etc. verwendet (vgl. ebd.). Er ist nicht isotrop, da er eine rechteckige Form hat (vgl. [6, Seite 103]). Zudem gewichtet der Box-Filter unabhängig von der Entfernung zum Hotspot. D. h. je größer die Filtermatrix ist, desto mehr beeinflussen weit entfernte Pixel das Ergebnis.

Dieser Filter vermindert kleine Störungen im Bild und hilft gegen Bildrauschen. Durch das Glätten verbreitert er scharfe Kanten über mehrere Pixel. Das bedeutet auch, dass bei geringem Abstand die Linien durch den Filter verschmolzen werden. Dadurch können beispielsweise Außenwände eines Gebäudes, die oft mit zwei Linien dargestellt sind, zu einer Linie zusammengefasst werden.

Der ***Gauß-Filter*** ist eine diskrete zweidimensionale Gauß-Funktion bzw. Gauß-Glocke (vgl. [6, Seite 104]).

Die Formel für diesen Filter lautet: $O^{G,\sigma}(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$.

Die Standardabweichung σ ist der durchschnittliche Abstand zum Mittelwert und bestimmt, wie breit diese Glockenkurve ist. Je höher die Standardabweichung ist, desto breiter ist die Glockenkurve. Je niedriger die Standardabweichung ist, desto schmaler ist die Glockenkurve.

Der Hauptunterschied zum Box-Filter dabei ist, dass der Gauß-Filter den Hotspot in der Filtermatrix priorisiert. Das bedeutet, dass der Hotspot den höchsten Wert der Filtermatrix zugewiesen wird. Die Ränder der Filtermatrix erhalten die Werte der Binomialkoeffizienten (vgl. [12, Seite 149-151]). Somit wird die Binomialverteilung als Näherung zur Gauß-Glocke verwendet (vgl. [34, Seite 168]). Das bedeutet, dass die Koeffizienten, die sich näher am Hotspot befinden, einen höheren Wert und auch mehr Einwirkung auf das Ergebnis als jene Koeffizienten haben, die weiter vom Hotspot entfernt sind. Daher verhält sich der Gauß-Filter etwas kontrollierter und kantenerhaltender als ein Box-Filter.

Bei der Anwendung des Gauß-Filters kann beispielsweise eine typische 3×3 Filtermatrix M verwendet werden, die wie folgt aussieht:

$$M = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Das gewichtete Mittel wird unter der Filtermatrix M berechnet. Dabei wird durch die Summe der Koeffizienten in der Filtermatrix dividiert, um zu vermeiden, dass das Ergebnis außerhalb des Intensitätsbereichs liegt. Danach wird das Ergebnis im Zielpixel $I'(u,v)$ gespeichert.

Die Gauß-Funktion ist isotrop, wenn eine Filtermatrixgröße für eine ausreichende Annäherung ausgewählt wird.

Die Kanten bleiben besser als beim Box-Filter erhalten. Die Linien mit geringem Abstand verschmelzen nicht, werden aber leicht verwischt. Außerdem vermindert er Bildrauschen und kleine Störungen im Bild.

Dieser Filter wird bei der Vorverarbeitung vom Canny-Kantendetektor (siehe Kapitel 2.8.1) eingesetzt, um nicht relevante Informationen, z. B. Beschriftungen aus den Grundrissbildern, zu verwischen.

Hochpassfilter

Der Hochpassfilter arbeitet konträr zum Tiefpassfilter. Das bedeutet, dass der Hochpassfilter Grauwertkanten und -spitzen im Bild verstärkt (vgl. [12, Seite 161]). Dadurch ist ein Detektieren von Kanten in bestimmten Richtungen im Quellbild ermöglicht. Ebenso reagiert dieser Filter auf Bildrauschen, weshalb vor dessen Anwendung das Bild meistens geglättet wird. Bei Hochpassfiltern kann zwischen verschiedenen Arten unterschieden werden (vgl. [12, Seite 154-161]). Eine Art davon sind die Kantenfilter, auf diese wird hier nun näher eingegangen.

Der Kantenfilter (auch „Differenzfilter“ oder „Gradientenfilter“ genannt) benutzt ebenfalls eine Filtermatrix M. Im Gegensatz zu Glättungsfiltern dürfen beim Kantenfilter einzelne Koeffizienten negativ sein (vgl. [6, Seite 104]). Für Kantenfilter gilt die Formel:

$$I'(u,v) = \sum_{(i,j)\in R^+} I(u+i, v+j) \cdot |M(i,j)| - \sum_{(i,j)\in R^-} I(u+i, v+j) \cdot |M(i,j)|$$

Dabei beschreibt R^+ die Filterregion mit positiven Koeffizienten $O(i,j) > 0$ und R^- die Filterregion mit negativen Koeffizienten $O(i,j) < 0$.

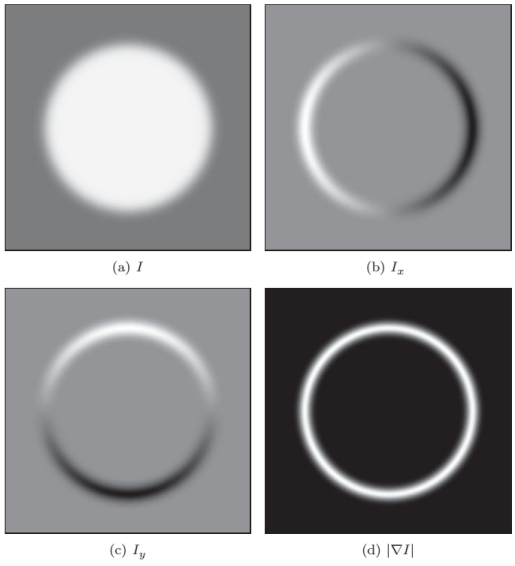

Abbildung 16: Beispielhafte Ergebnisse von Kantenfiltern. Bei Bild (a) ist eine Funktion I dargestellt. In Bild (b) wird die erste Ableitung horizontal entlang der x-Achse auf I angewandt. Bild (c) zeigt die erste Ableitung vertikal entlang der y-Achse auf I. Bild (d) veranschaulicht den Betrag von den Gradienten in I. (Quelle: [6, Seite 126])

Die Funktionsweise des Kantenfilters wird in der Abbildung 16 dargestellt. Durch die Anwendung des Kantenfilters werden Kanten mithilfe der diskreten Gradienten von der Intensität des Bildes gefunden. Bei jeder Art des Kantenfilters unterscheiden sich die Werte der Koeffizienten in der Filtermatrix (siehe Tabelle 5).

Operatoren	**Zur Detektion von vertikale Kanten**	**Zur Detektion von horizontale Kanten**
Prewitt-Operator	$M_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$M_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Sobel-Operator	$M_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$M_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Tabelle 5: Kantenoperatoren mit zugehörigen Filtermatrixen

Da der Sobel-Operator sowohl beim Canny-Kantendetektor als auch beim Harris-Eckendetektor eingesetzt wird, folgt dazu eine ausführliche Beschreibung.

Der ***Sobel-Operator*** berechnet den Gradienten der Intensität des Bildes (vgl. [12, Seite 154]).

Dabei gibt die Tangente die Steigung an einer x-Koordinate x_i einer Funktion $f(x)$ an. In der Abbildung 17 ist die Tangente als $f'(x_i)$ dargestellt. Zum Berechnen einer Tangente wird in der Regel auf die Ableitung $f'(x_i)$ zurückgegriffen.

Ist die Ableitungsfunktion nicht gegeben, so wird auf eine Näherung der Steigung zurückgegriffen. In der diskreten Mathematik wird hierzu der Differenzenquotient verwendet. Die Idee dahinter ist, dass durch die Steigung einer Geraden durch zwei Punkten nahe x_i – also einer Sekante – angenähert wird.

Die grüne Sekante in dieser Abbildung zeigt bereits eine ähnliche Steigung wie $f'(x_i)$. Die blaue Sekante ist aber eine bessere Näherung. Grundsätzlich gilt, je näher die Punkte zur Berechnung der Sekante an der gesuchten Stelle x_i liegen, desto näher ist die Steigung der Sekante der Steigung der Tangente.

In der Abbildung 17 sind die am nächsten gelegenen Funktionswerte, die benachbarten Pixel von x_i. Um die Steigung m der Sekante zu berechnen, wird der Differenzenquotient auf die beiden Punkte von dieser angewandt.

Die Formel lautet $m = \Delta y/\Delta x = (y_2 - y_1)/(x_2 - x_1)$. Das Δx beschreibt hierbei den Abstand der zwei Punkten in der x-Richtung. Dieser Abstand kann vereinfacht als eins angenommen werden. Somit ergibt sich die Formel: $m = \Delta y = (y_2 - y_1)$. Da die Funktionswerte, wie zuvor beschrieben, am besten die benachbarten Pixel von x_i sind, wird also vom Grauwert des rechten Nachbars x_{i+1} der Grauwert des linken Nachbars x_{i-1} abgezogen. Um die Reihenfolge im Bild zu beachten, kann die Formel umgeschrieben werden zu $m = -y_1 + y_2$. Oder um es in Bildpunkten auszudrücken: $-1 \cdot f(x_{i-1}) + 0 \cdot f(x_i) + 1 \cdot f(x_{i+1})$. Genau das ist nämlich auch das, was in der Matrix des Sobel-Operators steht: $[-1, 0, 1]$. Dabei liefert er eine relativ ungenaue Approximation, hebt Kanten stark hervor und lässt andere Artefakte verschwimmen.

In der Praxis wird oft ein Gauß-Filter vorher angewendet, um später mit dem Sobel-Operator nur die stark ausgeprägten Kanten zu detektieren.

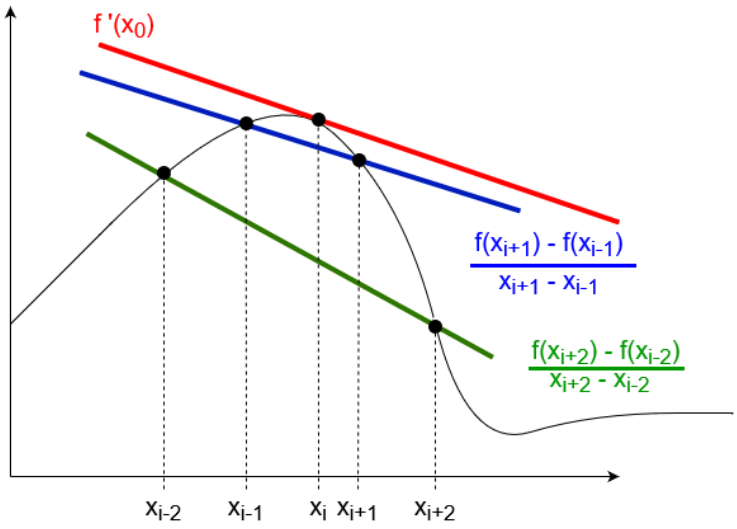

Abbildung 17: Annäherung der Steigung $f'(x_i)$ mittels Differenzenquotient

Es besteht die Möglichkeit, Operatoren zu erweitern (vgl. [12, Seite 157]). Diese entsteht, wenn beispielsweise der Sobel-Operator um 45 Grad zu rotiert wird. Durch diese Erweiterung können die diagonalen Kanten erkannt werden:

$$M_{x_2}^S = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} \quad M_{y_2}^S = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

2.5.2 Nichtlineare Filter

Die nichtlinearen Filter bilden eine nichtlineare Funktion ab (vgl. [6, Seite 111]). Im Gegensatz zum linearen Filter verwischen diese Filter die beabsichtigten Bildstrukturen – wie Punkte, Kanten und Linien. Dadurch vermindert sich die Bildqualität.

Bei nichtlinearen Filtern wird zwischen Richtungsfiltern, Relaxionsfiltern, morphologische Filtern und Rangordnungsfiltern unterschieden (vgl. [34, Seite 170-182]). Da morphologische Filter und Rangordnungsfilter für diese Bachelorarbeit relevant sind, werden diese nachfolgend genauer beschrieben.

Morphologische Filter

„Morphologie ist die Lehre von den Gestalten und Formen. Morphologischen Operationen verändern die Form von Objekten in einem Bild." [12, Seite 163] Für die Verwendung einer Filtermatrix sind beliebige Formen möglich (siehe Abbildung 18).

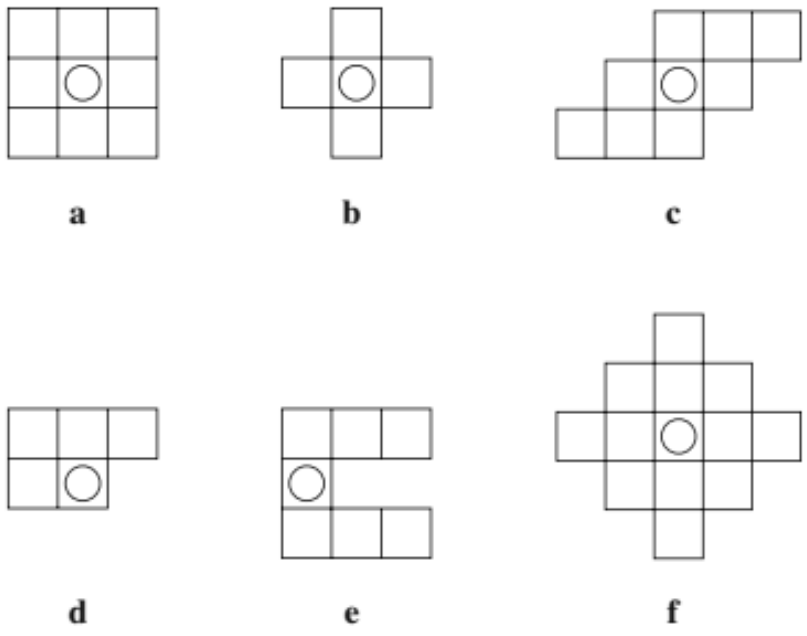

Abbildung 18: Morphologische Filtermatrizen. Der Hotspot ist als Kreis in den Filtermatrizen eingezeichnet. Dieser muss sich nicht immer in der Mitte befinden. (Quelle: [12, Seite 163])

Minimum-Operator - Erosion
Der Minimum-Operator (auch „Minimum-Filter“ genannt) verknüpft Intensitätswerte des Quellbilds mit der nichtlinearen Funktion (vgl. [6, Seite 112]): $I'(u,v) = min\{I(u+i, v+j)|i,j \in R\}$.

Durch diesen Operator wird eine Schrumpfung (engl. *Erosion*) der hellen Bildbereiche erzielt, da der Intensitätswert des Pixels durch den minimalen Wert ersetzt wird (vgl. [27, Seite 152]). Dadurch werden die dunklen Bildbereiche auf Kosten der hellen ausgedehnt.

Maximum-Operator - Dilation
Der Maximum-Operator (auch „Maximum-Filter“ genannt) verknüpft Intensitätswerte des Quellbilds mit der nichtlinearen Funktion (vgl. [6, Seite 112]): $I'(u,v) = max\{I(u+i, v+j)|i,j \in R\}$.

Dieser Operator hat genau den gegenteiligen Effekt wie der Minimum-Operator. Der Maximum-Operator bezweckt eine Ausdehnung (engl. *Dilation*) der hellen Bildbereiche (vgl. [27, Seite 152]). Hierbei werden die dunkleren Bildbereiche verdrängt.

Opening
Als Opening wird der Vorgang bezeichnet, wenn nach einer Erosion eine Dilatation ausgeführt wird (vgl. [26, Seite 148]). Diese Operation wird für das Separieren von miteinander verbundenen Teilchen verwendet (siehe Abbildung 19). Allerdings entstehen dadurch auch kleine Löcher.

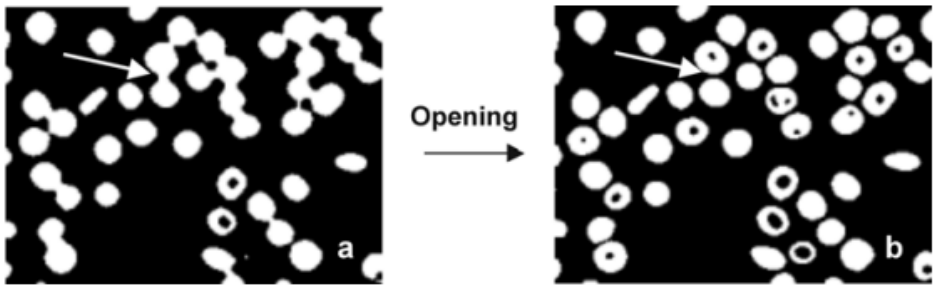

Abbildung 19: Opening (Quelle: [26, Seite 149])

Closing

Beim Closing wird eine Dilatation gefolgt von einer Erosion ausgeführt (vgl. [26, Seite 148]). Im Gegensatz zum Opening werden hier kleine Teilchen verbunden und Löcher geschlossen (siehe Abbildung 20).

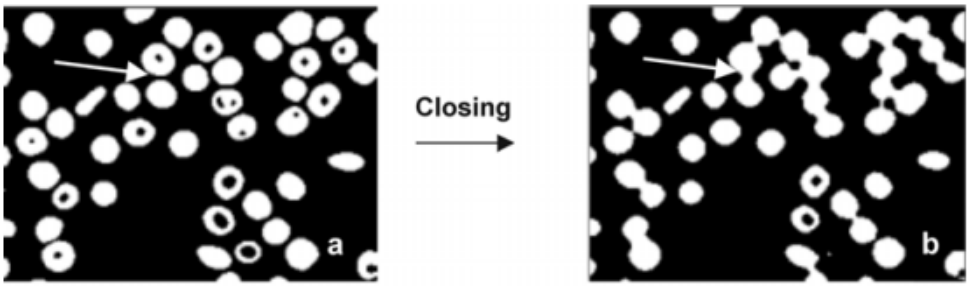

Abbildung 20: Closing (Quelle: [26, Seite 148])

Sowohl das Opening als auch das Closing sind kaskadierbar, d. h. sie können mehrfach hintereinander ausgeführt werden, ohne das Bild zu verändern (vgl. [34, Seite 178]).

Thinning, Thicking, Skeleton

„Die Thinning-Operation ist eine sehr häufige Anwendung morphologischer Verfahren mit dem Ziel, binäre Strukturen auf eine maximale Dicke von einem Pixel zu reduzieren, ohne sie dabei in mehrere Teile zu zerlegen.“ [6, Seite 205] Diese wird oft nacheinander ausgeführt und ist eine Erosion mit bestimmten Bedingungen.

Eine Bedingung könnte beispielsweise eine Hit-or-Miss Transformation sein. Dabei wird als Erstes eine Maske definiert, auf der Intensitätswerte angeordnet werden (vgl. [34, Seite 178-179]). Mithilfe dieser Maske können Pixel detektiert werden, die gewisse geometrische Eigenschaften aufweisen (wie z. B. Eckpunkte, Randpunkte, o. Ä.). Anschließend wird diese Maske über das Bild bewegt. Stimmt die Anordnung der Intensitätswerte auf der Maske mit dem aktuellen Bildbereich überein, so wird dies „Hit“ genannt und das Zielpixel wird schwarz markiert. Falls es keine Übereinstimmung gibt, so wird dies „Miss“ genannt und der Zielpixel weiß markiert.

„Diese wird an einer bestimmten Position nur dann angewandt, wenn auch nach erfolgter Operation eine ausreichend dicke Struktur verbleibt und keine Trennung in mehrere Teile erfolgt.“ [6, Seite 206] Die Operation wird so lange wiederholt, bis es ans Skelett (engl. *Skeleton*) des Bildes angelangt bzw. sich keine Änderungen im Ergebnisbild mehr ergeben.

Das Thinning wird beim Canny-Detektor in dieser Bachelorarbeit (siehe Kapitel 2.8.1) angewendet.

Rangordnungsfilter

Bei Rangordnungsfiltern werden die Intensitätswerte in einer Rangordnung (z. B. nach der Größe) sortiert. Je nach Art des Rangordnungsfilters wird dann der entsprechende Intensitätswert daraus ausgewählt. Unter dem Rang eines Wertes wird die Position innerhalb der geordneten Reihe aller Beobachtungswerte verstanden (vgl. [32, Seite 147]).

Der ***Medianfilter*** wird mit der nichtlinearen Funktion verknüpft (vgl. [6, Seite 112]): $I'(u,v) = median\{I(u+i, v+j)|i,j \in R\}$.

Die Formel für Median lautet (vgl. [3, Seite 73-4]):

$$\tilde{x} = \begin{cases} x_{[\frac{n+1}{2}]}, & n \text{ ist gerade} \\ \frac{1}{2}(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}), & n \text{ ist ungerade} \end{cases}$$

Die Variable $x_{[i]}$ ist das Element x, das sich an der Position i in der Rangordnung befindet. Der Median (auch „Zentralwert“ genannt) ist der Wert, welcher sich in der Mitte in der Folge von den sortierten Intensitätswerten befindet (siehe Abbildung 21).

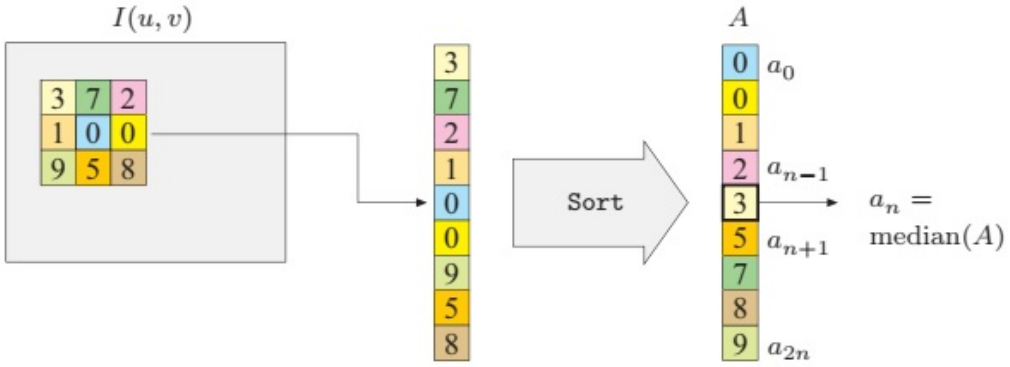

Abbildung 21: Funktionsweise des Medianfilters (Quelle: [6, Seite 114])

Beim Medianfilter werden die Intensitätswerte von der Filterregion sortiert. Anschließend wird der Median ausgewählt und im Zielpixel $I'(u,v)$ gespeichert.

Er ist ein robustes Lagemaß. Das bedeutet, dass er nicht anfällig von Ausreißern ist. Daher eignet er sich gut, um einzelne Störpixel zu entfernen (siehe Abbildung 22). Ebenso wird er verwendet, um sporadische Bildstörungen (z. B. einzelne Bildpunkte), Bildrauschen oder ganze Bildzeilen zu entfernen (vgl. [12, Seite 164-168]). Nach Anwendung des Medianfilters bleiben die wichtigen Strukturen (z. B. Linien mit geringem Abstand) und Kanten im Bild erhalten (vgl. [6, Seite 113]). Wenn allerdings ein großflächiges

Abbildung 22: Beispielhafte Darstellung von nichtlinearer Filter auf ein verrauschtes Bild. (In Anlehnung an: [6, Seite 113])

Bildrauschen im Bild vorhanden ist, so ist das Anwenden des Medianfilters eher weniger sinnvoll.

Der Medianfilter ist kaskadierbar. Manchmal kann die Bildqualität verbessert werden, indem auf das skalierte und hochpassgefilterte Bild anschließend ein Medianfilter angewendet wird (vgl. [27, Seite 155]).

Ein Nachteil beim Medianfilter ist, dass sehr hohe Rechenzeit aufgrund des Sortiervorgangs benötigt wird. Sowohl beim Minimum- als auch beim Maximum-Operator wird weniger Rechenzeit als beim Medianfilter benötigt, da nur der größte bzw. kleinste Wert in der Folge gefunden werden muss und dies keine Sortierung bedingt.

2.5.3 Randbehandlung

Die Randbehandlung beinhaltet die Entscheidung, die es zu treffen gilt, was geschehen soll, wenn die Filtermatrix über den Bildrand hinausragt und somit einige Filterkoeffizienten sich außerhalb des Bildes befinden (siehe Abbildung 23).

Für Randergänzung wird ein Rahmen (engl. *Padding*) um das Bild gelegt, der mit Intensitätswerten ausgefüllt wird. Dabei kann zwischen folgenden Möglichkeiten gewählt werden (vgl. [21, Seite 306-307], [6, Seite 118-119]):

1. **Randergänzung mit Nullen**: Da unbekannt ist, was sich außerhalb des Bildrandes befindet, wird den außerhalb liegenden Pixeln den Intensitätswert null (schwarz) zugewiesen. Dadurch entstehen scharfe schwarze Kanten und die eigentliche Bildgröße wird mit jeder Anwendung des Filters verkleinert.

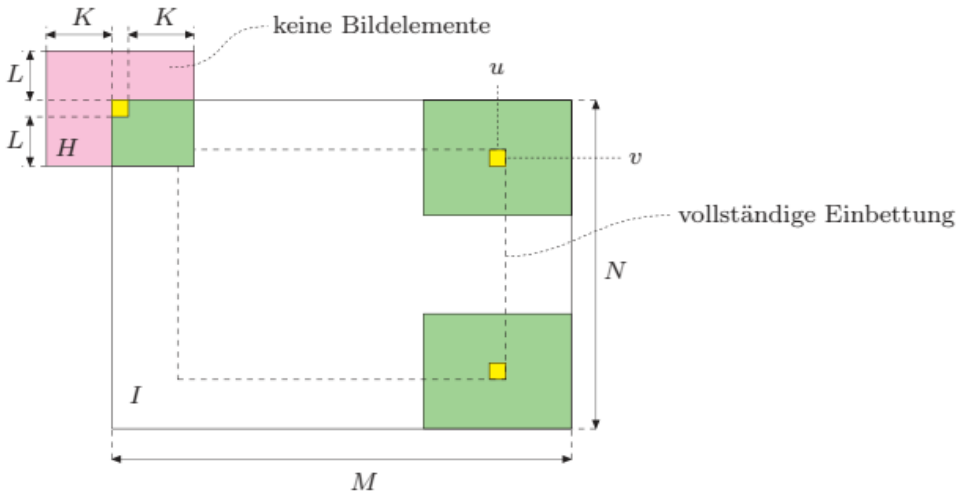

Abbildung 23: Randbehandlung für Filter. (Quelle: [6, Seite 103])

2. **Randergänzung mit fixem Wert**: Diese ist eine Verallgemeinerung von der Randergänzung mit Nullen. Je nach Bild und Anwendungsszenario bieten sich andere Werte als null eher an. Hierbei wird ein fixer Wert für die Pixel außerhalb des Bildes angenommen (siehe Abbildung 24/(b)). Bei großen Filtern ist diese Randbehandlung nicht empfehlenswert, da dies zu Artefakten (Störungen) im Bild führt.

3. **Randergänzung mittels Signalextrapolation**: Der Intensitätswert des am nächsten liegenden Randpixels wird übernommen. Dadurch erlangen die Pixel am Rand hohes Gewicht bei der Filteroperation (siehe Abbildung 24/(c)).

4. **Randergänzung mittels Spiegelung**: Die Intensitätswerte der nächstgelegenen Bildkante werden nach außen gespiegelt dargestellt (siehe Abbildung 24/(d)).

5. **Periodische Randergänzung**: Die Intensitätswerte von den außerhalb liegenden Pixeln wiederholen sich zyklisch in allen Richtungen (siehe Abbildung 24/(e)). Dies wird oft für die diskrete Spektralanalyse angewendet.

Für die Implementierung dieser Bachelorarbeit bietet sich eine Randergänzung mit fixem Wert 255 an, da der Hintergrund des Grundrissbildes meist weiß ist.

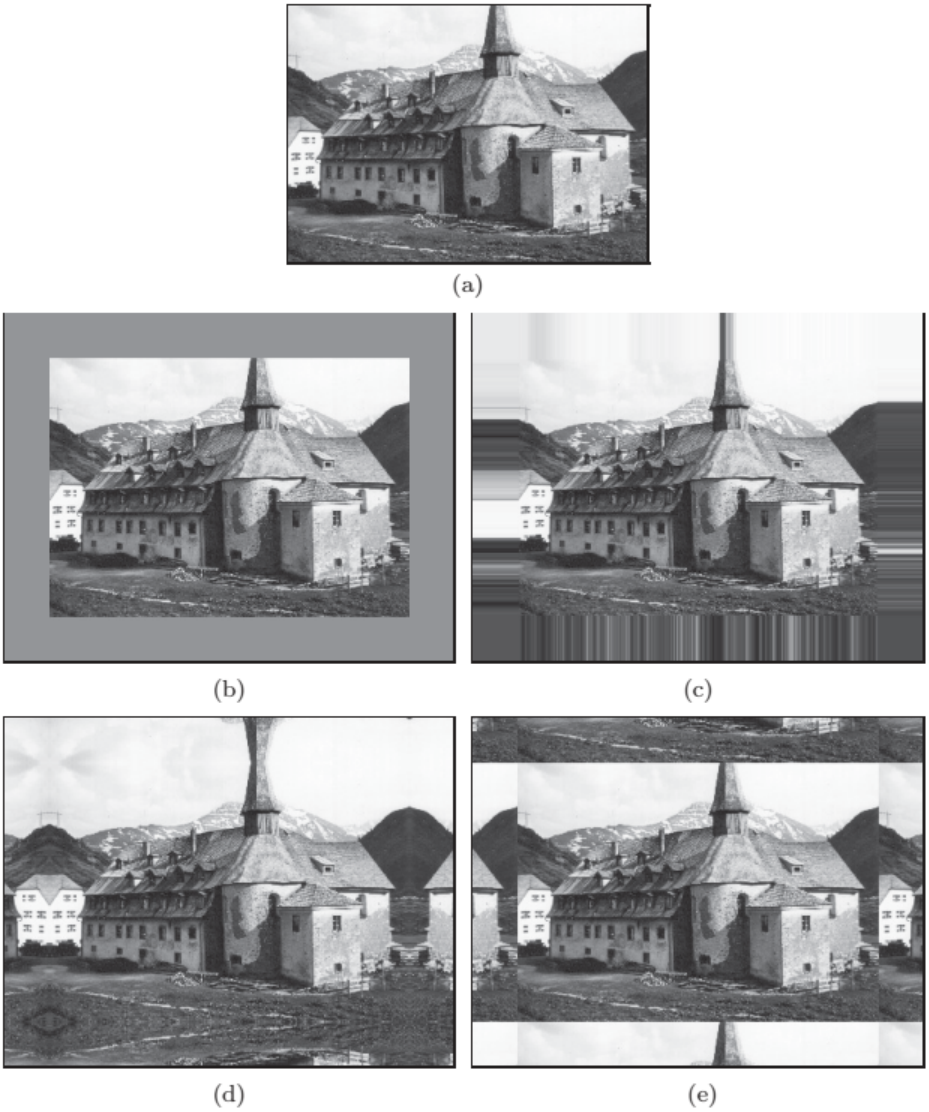

Abbildung 24: Randbehandlungsmöglichkeiten. (a) zeigt das Originalbild. In den weiteren Bildern werden die Randergänzung mit fixem Wert (b), mit Signalextrapolation (c), mittels Spiegelung (d) und die periodische Randergänzung (e) dargestellt. (Quelle: [6, Seite 120])

2.6 Polygon

Der Begriff „Polygon“ bedeutet Vieleck (vgl. [11]). Ein Polygon ist eine geometrische Figur, die durch einen geschlossenen Streckenzug verbunden wird (vgl. [39]). Dieser Streckenzug wird Polygonzug genannt. Das Polygon ist geschlossen, wenn der Startknoten und Endknoten mit einer Kante verbunden ist.

Die Polygonzüge werden in dieser Bachelorarbeit verwendet, um Wände und Hindernisse einzuzeichnen. Dabei werden die Eckpunkte des Polygonzugs in einer Datei abgespeichert, welche von pFlow eingelesen wird.

2.7 Graph

„Ein Graph $G(V, E)$ besteht aus einer endlichen Menge V von Knoten (engl. *Vertex*) und einer Menge E von Kanten (engl. *Edge*) a, b mit $a, b \in V$, $a \neq b$.“[35, Seite 416] Ein Graph kann mithilfe eines Diagramms dargestellt werden (vgl. [33, Seite 158]). Dabei werden **Knoten** (Synonym: Ecke) als Punkte eingezeichnet. Eine **Kante** verbindet zwei Knoten miteinander mittels einer Linie. Sind zwei Knoten a, b mit einer Kante verbunden a, b, so werden diese Knoten als Endknoten bezeichnet (vgl. [35, Seite 416]). In dieser Bachelorarbeit wird ein Graph benutzt, um Polygonzüge darzustellen. Die Ecken der Polygonzüge entsprechen den Knoten des Graphen. Zum Speichern wird eine Liste von Knoten mit Koordinaten in entsprechender Reihenfolge verwendet. Es wird auf eine Inzidenzmatrix verzichtet, da zu jedem Knoten nur zwei Kanten existieren.

2.8 Detektor

„Ein Detektor (auch Feature- oder Merkmalsdetektor genannt) hat die Aufgabe, charakteristische Merkmale oder markante Punkte in einem Bild zu finden.“ [27, Seite 346] Er bedient sich an verschiedenen vorher definierten Methoden bzw. Befehlssätzen.

Die unten angeführten Detektoren werden für die Detektion von Ecken und Kanten in der Implementierung benötigt (siehe Kapitel 3).

2.8.1 Canny-Kantendetektor

Der Canny-Kantendetektor (engl. *Canny Edge Detector*) ist sehr weit verbreitet und sehr robust. Er wurde von J. F. Canny 1983 vorgestellt und berechnet eine möglichst genaue Lokalisierung der Kanten (vgl. [27, Seite 179-180]).

Die Vorgangsweise des Canny-Kantendetektors kann in 3 Phasen eingeteilt werden (vgl. [6, Seite 137-145]):

1. Vorverarbeitung:
   (a) Das Bild wird in ein Grauwertbild konvertiert.
   (b) Der Gauß-Filter wird auf das Grauwertbild angewendet.
   (c) Der Sobel-Operator wird in x- und y-Richtung auf das Bild ausgeführt, um den Gradienten zu berechnen und die Richtung zu bestimmen.
2. Lokalisierung der Kanten:
   Mithilfe des Ergebnisses vom Sobel-Operator werden die Kanten im Bild lokalisiert. Bei hoher Bildauflösung breitet sich der Gradient über viele Pixel aus. Ideal wäre eine geringe Bildauflösung, denn dort ist ein scharfer Gradient am einfachsten zu identifizieren.
3. Verfolgung der Kanten:
   (a) Es wird eine Non-Maximum-Suppression durchgeführt.
   Dabei wird jedes Pixel im Bild geprüft, ob dieses ein lokales Maximum ist. Das bedeutet, es wird überprüft, ob dessen Intensitätswert größer ist als der Intensitätswert von den direkten benachbarten Pixeln. Alle Werte, die ungleich dem lokalen maximalen Wert sind, werden auf null gesetzt. Dadurch werden die dominanten Kanten identifiziert und beispielsweise Bildrauschen reduziert. In diesem Schritt erfolgt ein Thinning, da die Kanten auf den Pixeln mit maximaler Ausprägung reduziert werden.

(b) Es wird die Hysterese Schwellenwertoperation (engl. *Hysteresis Thresholding*) angewendet.

Dabei wird ein unterer und ein oberer Schwellenwert dem Canny-Kantendetektor übergeben (siehe Abbildung 25). Diese Werte müssen bei einem Grauwertbild im Intervall $[0; 255]$ liegen.

Ist der Betrag des Gradienten unterhalb des unteren Schwellenwerts, so wird diese Kante ignoriert. Wenn die Ausprägung der detektierten Kante oberhalb des oberen Schwellenwerts liegt, so wird diese detektiert. Liegt die Ausprägung der detektierten Kante zwischen den beiden Schwellenwerten, dann muss beachtet werden, ob sie mit einer bereits detektierten Kante in Verbindung steht. Ist dies der Fall, so wird diese Kante detektiert, sonst wird sie ignoriert.

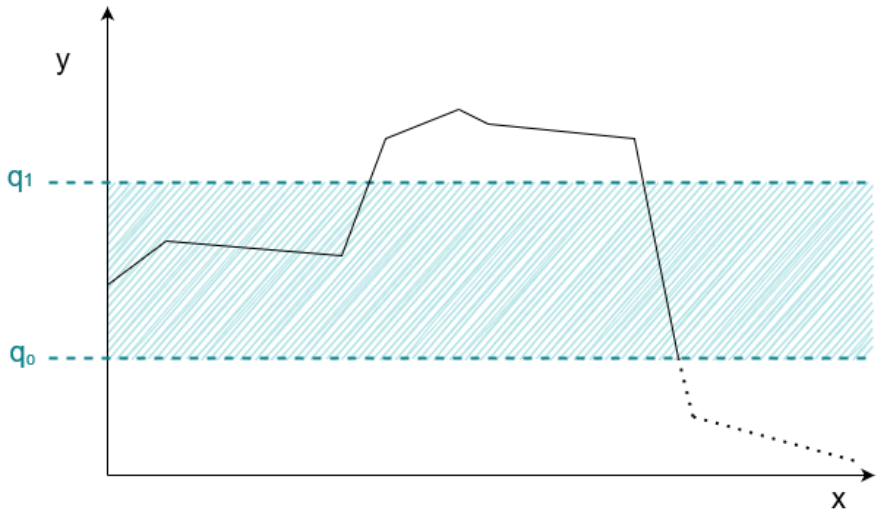

Abbildung 25: Hysterese Schwellenwertoperation. Die abgebildete Funktion stellt die Ausprägung der gefundenen Kanten dar. Die gepunkteten Funktionswerte werden ignoriert, da diese unter dem Schwellenwert q_0 liegen. Für Werte zwischen q_0 und q_1 wird geprüft, ob diese mit einer stark ausgeprägten Kante verbunden sind. Wenn dem so ist, dann werden diese detektiert.

Diesem Kantendetektor werden das Grauwertbild, den unteren und oberen Schwellenwert sowie die Größe der Filtermatrix als Eingabe-Parameter übergeben.

2.8.2 Harris-Eckendetektor

Der Harris-Eckendetektor (auch „Plessey feature point detector“ genannt, engl. *Harris Corner Detector*) wurde von C. G. Harris und M. Stephens 1988 entwickelt (vgl. [6, Seite 156-158]). Dieser Detektor erkennt Eckpunkte.

Dabei wird wie folgt vorgegangen (vgl. [27, Seite 346-349]):

1. Der Sobel-Operator wird angewendet, um Kanten entlang der x- und y-Richtung zu lokalisieren.

2. Anschließend wird für jedes Pixel die Strukturmatrix S bestimmt:

$$S = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} s_x^2 & s_x s_y \\ s_x s_y & s_y^2 \end{bmatrix}$$

   Hierbei entspricht s_x der Ableitung in x-Richtung und s_y der Ableitung in y-Richtung. Die Strukturmatrix S „[...] hat eine gewisse Ähnlichkeit zur Hesse-Matrix“. [23, Seite 1]

3. Die Ausprägung der Ecke wird mithilfe der folgenden Formel berechnet: $C(x, y) = A * B - C^2 - k(A + B)^2$. Eine Ecke entsteht, wenn zwei Kanten sich treffen. Da A und B die Kantenstärke in unterschiedliche Richtungen repräsentieren, müssen diese größer null sein, falls entsprechende Kante in diesem Bereich existiert. Ist dies nicht der Fall, so gilt $A * B = 0$. Somit existiert keine Kante in $C(x, y)$. Mit der Variable `k` wird die Sensibilität gesteuert.

4. Mithilfe eines vorgegebenen Schwellenwerts werden zu schwach ausgeprägte Ecken aussortiert. Die verbleibenden Ecken werden der Ausprägung nach sortiert und entsprechend zwischengespeichert.

5. Das Bild wird in Kacheln aufgeteilt. Pro Kachel werden die schwächsten Ecken entfernt, sodass nur eine festgelegte Anzahl von stärksten Ecken verbleiben.

Der Harris-Eckendetektor erwartet als Eingabeparameter ein Grauwertbild, die Größe der Filtermatrix für den Sobel-Operator, die Anzahl von benachbarten Pixeln, welche in die Berechnung eingehen sollen, und den Wert `k` für die Gleichung.

3 Implementierung

Bei der Implementierung wird nach Teile und Herrsche-Ansatz (engl. *Divide and Conquer*) vorgegangen. Dabei wird die Aufgabenstellung (siehe 1.2) in kleine Arbeitspakete geschnürt und nacheinander abgearbeitet (vgl. [30, Seite 147]). Jedes Unterkapitel entspricht einem kleinen Arbeitspaket. Für die Implementierung werden die erwähnten Werkzeuge aus dem Kapitel 1.4 verwendet. Die Methoden werden an Grundrissbildern mit unterschiedlicher Komplexität getestet (siehe Abbildung 31). Folgende Arbeitspakete wurden definiert:

- Bildvorverarbeitung
- Kantendetektion
- Eckendetektion
- Erstellung der Polygonzüge

3.1 Bildvorverarbeitung

Im ersten Schritt soll herausgefunden werden, welche zwei Filter sich potenziell für die Vorverarbeitung (Bereinigung) von Grundrissbildern eignen. Dafür werden die von OpenCV zur Verfügung stehenden Filter mit verschiedenen Größen der Filtermatrix (engl. *Kernel Size (ksize)*) getestet. Da OpenCV die Filter „Edge Preserving Filter“ und „Bilateral Blur“ zur Verfügung stellt, wurden diese auch für den Test eingesetzt. Diese beiden Filter verfolgen das Ziel, sowohl unerwünschte Texturen zu glätten als auch die Kanten zu erhalten (vgl. [14]). Da diese keine konfigurierbaren Größen der Filtermatrix als Eingabeparameter benötigten, werden sie in der Abbildung 26 separat aufgelistet.

Vor der Anwendung des Filters werden die verschiedenen Grundrissbilder als Grauwertbild konvertiert, da der Canny-Kantendetektor ein Grauwertbild als Eingabe erwartet. Anschließend werden die Auswirkungen der verschiedenen Filter bewertet. Ziel hierbei ist es, einen Filter zu finden, welcher die Kanten von den Wänden erhält, aber die Texte glättet, sodass eine möglichst genaue Detektion der Kanten erfolgt.

Aus der subjektiven Bewertung der Filter wurden folgende Erkenntnisse gewonnen:

	ksize = 3			ksize = 5			ksize = 7			ksize = 9			ksize = 11			ksize = 21				
Images	Gaussian Filter	Average Blur	Median Blur	Gaussian Filter	Average Blur	Median Blur	Gaussian Filter	Average Blur	Median Blur	Gaussian Filter	Average Blur	Median Blur	Gaussian Filter	Average Blur	Median Blur	**Gaussian Filter**	Average Blur	Median Blur	**Edge Preserving Filter**	Bilateral Blur
001.PNG	-	-	-	-	-	-	-	2	3	-	2	2	2	3	2	2	4	3	**1**	5
002.PNG	-	-	-	-	-	-	-	-	2	-	2	3	2	3	3	2	4	4	**1**	5
003.PNG	-	-	-	-	-	3	-	2	3	-	3	4	-	3	4	2	4	4	2	5
004.PNG	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	4	**1**	2	5
005_wry.png	-	-	-	-	-	-	-	3	2	2	3	3	2	4	3	2	4	4	2	5
floor_plan - Page 65.PNG	-	-	-	-	-	-	-	-	-	3	4	4	3	4	4	3	5	5	4	5
O_-1.PNG	-	-	-	-	-	-	-	-	2	-	3	-	3	3	4	3	4	5	4	5

Legende	
1 = Sehr gut	Kanten bleiben erhalten, Text ist verwaschen
2 = Gut	Wände sind sichtbar, Strukturen vom Text sind erkennbar
3 = Befriedigend	Alles verwaschen (Wände nicht mehr gut sichtbar)
4 = Genügend	Keine Struktur erkennbar
5 = Nicht genügend	Keine Änderung oder leeres Bild
-	Ergebnis bringt keine Verbesserung gegenüber der nächstgrößeren Kernelgröße

Abbildung 26: Bewertung der gefilterten Bilder. Die Spalten beziehen sich auf die entsprechenden Filter mit verschiedenen Größen der Filtermatrix. Die Zeilen zeigen unterschiedliche Grundrissbilder. In der Legende werden die Bewertungskriterien aufgeschlüsselt dargestellt. Die Grundrissbilder sind in Abbildung 31 eingefügt.
(Bildschirmabbildung von in Microsoft Excel erstellt.)

- Generell fällt auf, dass eingezeichnete Türen, Treppen, Sprechblasen und Texte für die Filter schwer zu verwischen sind.
- Die Filter haben kaum eine Auswirkung auf Bilder mit einer großen Bildauflösung. Um auch solche detailreichen Bilder noch sinnvoll zu verarbeiten, sollten diese vor Anwendung des Filters auf eine Maximalgröße verkleinert werden.
- Sowohl der Gauß-Filter mit ksize=21 als auch der Edge Preserving Filter erbringen solide Ergebnisse auf den getesteten Bildern. Das bedeutet, beide Filter erhalten die relevanten Kanten und glätten den Text. Daher werden diese in der nächsten Implementierung genauer miteinander untersucht und miteinander verglichen.

3.2 Kantendetektion

Im zweiten Schritt wird ein Kantendetektor ausgewählt und implementiert. Ebenso soll festgelegt werden, in welcher Reihenfolge die Filter bei der Vor- und Nachverarbeitung des Bildes angewendet werden. Als Eingabeparameter wird das Grundrissbild vom Keller des O-Gebäudes der HTWG übergeben (siehe Abbildung 32 / (a)).

Der Canny-Kantendetektor erfordert bei der Vorverarbeitung, dass ein Bild als Grauwertbild transformiert und danach ein Gauß-Filter darauf angewendet wird. Aufgrund der zuvor gewonnenen Erkenntnisse (siehe Kapitel 3.1) wird hier der Gauß-Filter mit einer `ksize=21` verwendet.

Um Edge Preserving Filter und Gauß-Filter miteinander zu vergleichen, wird auch eine Vorverarbeitung nur mit Edge Preserving durchgeführt.

Anschließend wird über die zwei Grundriss-Grauwertbilder – eins davon ist mit Gauß-Filter (siehe Abbildung 32 / (b)) und das andere mit Edge Preserving Filter (siehe Abbildung 32 / (c)) vorverarbeitet – dem Canny-Kantendetektor übergeben.

Bei den resultierenden Bildern ist ersichtlich, dass durch die Vorverarbeitung mit dem Gauß-Filter ein besseres Ergebnis erzielt wird. Beim Verwenden des Edge Preserving Filters wird der Text im resultierenden Bild schlechter geglättet, was zu schlechteren Detektionen führt und deswegen bei der Kantenlokalisierung kein Vorteil bringt. Daher wird der Gauß-Filter für die Vorverarbeitung vom Canny-Kantendetektor bei dieser Bachelorarbeit genutzt.

Um den Text in der Vorverarbeitung besser verwaschen zu können, kam die Idee auf, ein Closing (siehe Kapitel 2.5.2) zu verwenden. Eine Bestätigung für diese Annahme ist bei näherer Betrachtung des Raums K12 (siehe Abbildung 32 / (d) und (e)) ersichtlich, denn dort tritt der gewünschte Effekt ein, weil die Raumbezeichnung fast vollständig verschwindet. Für die Vorverarbeitung beim Closing bewährt sich ksize=5 und als Filtermatrix-Form eine Ellipse.

Da Wände im Grundrissbild zwei parallel verlaufende Kanten aufweisen, sollen diese miteinander verschmolzen werden. Einerseits gibt dies optisch die Wände besser wieder. Andererseits ist dies auch für weitere Algorithmen hilfreich, die auf den gefundenen Wänden aufbauen (siehe Kapitel 3.4). Daher wird in der Nachbearbeitung noch mal ein Closing auf das Bild angewendet. Dabei wird anfangs als Filtermatrix-Form eine Ellipse verwendet und sich an die `ksize` herangetastet (siehe Abbildung (32 / (e), (f) und (g)). Diese liefert mit `ksize=7` ein brauchbares Ergebnis.

Allerdings sind in den Ecken des Grundrissbildes noch schwarze Artefakte vom vorherigen Closing (siehe Abbildung 32 / (g)). Um diese zu entfernen, wird die Filtermatrix-Form auf ein Rechteck umgestellt (siehe Abbildung 32 / (h)).

Da Türen, Treppen, Sprechblasen, Texte etc. auf diesem Grundrissbild enthalten sind, bringt dies Artefakte mit sich. Um diese Artefakte zu entfernen, reicht die Vorverarbeitung aus dem vorherigen Kapitel nicht aus. Daher wird die Komplexität der Grundrissbilder vereinfacht, sodass mit den nächsten Schritten fortgefahren werden kann. Um die Komplexität zu verringern, wird in den weiteren Implementierungen mit vereinfachten Grundrissbildern, die die oben genannten Eigenschaften nicht enthalten, gearbeitet.

Der verwendete Code für diesen Schritt sieht wie folgt aus (siehe Programmcode 1):

```
1  def apply_customized_canny(image_path):
2    #1. Schritt: Bild einlesen und als Grauwertbild speichert
3    gray_image = img_read(filename=image_path,mode=gray)
4    #2. Schritt: Gauss-Filter mit ksize=(21,21)
5    blur_image = add_gaussian_blur(image=image_gray,ksize=(21, 21))
6    #3. Schritt: Closing mit ksize=(5,5) und Ellipsen-Form
7    closing_image = closing(image=blur_image,form=ELLIPSE,
8                            ksize=(5, 5))
9    #4. Schritt: Canny-Kantendetektor
10   canny_image = canny(image=closing_image)
11   #5. Schritt: Closing mit ksize=(12,12) und Rechteck-Form
12   return closing(image=image_canny,form=RECTANGLE,ksize=(12, 12))
```

Programmcode 1: Kantendetektion

3.3 Eckendetektion

Nachdem die Kanten detektiert sind, müssen die Eckpunkte gefunden werden. Damit in späterer Folge der Polygonzug erstellt werden kann. Um die Eckpunkte zu detektieren, wird der Harris-Eckendetektor implementiert.

Dem Harris-Eckendetektor werden folgende Übergabeparameter mitgegeben:

- Bild ist ein simples Grundrissbild, welches zuvor mit Closing, Canny-Kantendetektor anschließend wieder mit Closing bearbeitet wurde (siehe z. B. Abbildung 33/(b))
- Der Parameter `blocksize` gibt die Anzahl von benachbarten Pixeln an, die bei der Detektierung der Ecken berücksichtigt werden. Dieser Parameter muss größer als eins sein.
- Die `ksize` definiert die Größe der Filtermatrix, welche für die Ausführung des Sobel-Operators benötigt wird. Dieser muss größer als die `blocksize` gewählt werden.
- Der Parameter `k` gibt an, wie stark die Ecke ausgeprägt ist, um sie zu detektieren. Je kleiner der Wert für diesen Parameter gewählt wird, desto mehr Ecken werden erkannt.

Als Ergebnis liefert der Harris-Eckendetektor ein zweidimensionales Array mit `True`- und `False`-Werten zurück. Dieses Array ist so groß wie das Bild, das dem Harris-Eckendetektor übergeben wurde. Jedes Feld im Array entspricht einem Pixel im Bild. Ist der Wert des Feldes `True`, so ist hier eine Ecke detektiert worden. Ein `False`-Wert bedeutet keine Ecke. Da der Harris-Detektor viele Detektionen pro tatsächlicher Ecke liefert, werden diese anschließend zusammengefasst. Dabei wurden folgende Ansätze getestet:

Erste Ecke pro Cluster
Im ersten Ansatz werden die folgenden Übergabeparameter mitgegeben:
`blocksize = 2, ksize= 3, k=0.04`.

Der Harris-Eckendetektor zeichnet viele Ecken im Bereich ein, in welchen die tatsächliche Ecke liegt. Die Ecken werden innerhalb einer Distanz in einem bestimmten Bereich (engl. *Cluster*) zusammengefasst. Die Distanz wird mit der Konstante namens `MAX_DISTANCE_CORNER` angegeben und mittels euklidischen Abstands berechnet. Das Resultat zeigt, dass die 200 gefundenen Ecken vom Harris-Eckendetektor erfolgreich auf 15 reduziert werden konnten. Hierbei wird auch ersichtlich, dass der Harris-Eckendetektor sehr sensibel reagiert, da eine Ecke zu viel eingezeichnet wurde, weil ein Pixel (siehe Abbildung 33/(c)) falsch eingezeichnet ist.

Gemittelte Ecke pro Cluster
Da die erste gefundene Ecke nicht immer auf der tatsächlichen Ecke liegt, wird das Ziel verfolgt, sich der tatsächlichen Ecke besser anzunähern. Im zweiten Ansatz wird aus den Koordinaten der Ecken innerhalb eines Clusters jeweils der Mittelwert berechnet und als neuer Eckpunkt abgespeichert. Die Annahme dahinter ist, dass eine Ecke im Bild durch den Mittelwert des zugehörigen Clusters besser als durch die Koordinate der ersten Detektion in diesem Cluster lokalisiert wird.

Davor werden die folgenden Übergabeparameter mitgegeben: `blockSize=2, ksize=5, k=0.05`. Die `ksize` wurde erhöht, damit weniger Kanten detektiert werden. Der Grund hierfür ist, dass nun mehr Pixel in die Berechnung der Kante einfließen und somit die störenden Pixel diese weniger beeinflussen. Infolge wird `k` erhöht, da weniger Ecken lokalisiert werden.

Das Resultat des zweiten Ansatzes gibt die Positionen der tatsächlichen Ecken besser als im ersten Versuch wieder. Die Implementierung des zweiten Versuch sieht wie folgt aus (siehe Programmcode 2):

```
1 def apply_customized_harris(image_path):
2   #1. Schritt: Kantenbild generieren
3   canny_image=apply_customized_canny(image_path)
4   #2. Schritt: Harris-Eckendetector
5   corners[] = harris(canny_image)
6   #3. Schritt: Cluster bilden
7   clusters[] = get_clusters(corners, MAX_DISTANCE_CORNER)
8   #4. Schritt: bereinigte Ecken-Liste
9   return get_mean_corner_of_cluster(clusters)
```

Programmcode 2: Eckendetektion

Generell wurden folgende Erkenntnisse aus dieses Ansatzes gewonnen:

- Wenn die Ecken in Cluster zusammengefasst werden, dann kann sich dies suboptimal auf Türen auswirken. Es kann nämlich passieren, dass nahe liegende Ecken, die beispielsweise die beiden Seiten einer Türe kennzeichnen, als ein Cluster erkannt und entsprechend zusammengefasst werden. Durch die Mittlung der Ecken ist dann die resultierende Ecke in der Mitte von der Tür eingezeichnet und gibt dadurch nicht mehr den Grundriss wieder.

  Eine kleinere `MAX_DISTANCE_CORNER` löst das Problem zwar, weil diese kleiner als die Türbreite ist und dadurch auch kleinere Cluster gebildet werden, aber dann könnten die Außenwandecken in zwei Cluster zerfallen. Daher muss ein Trade-Off mit diesem Parameter eingegangen werden.

- Treppen haben viele Ecken. Diese werden oft bei der Bildung von Cluster aufgrund `MAX_DISTANCE_CORNER` verworfen. Daher ist es nicht empfehlenswert, wenn diese im vereinfachten Grundrissbild enthalten sind.
- Die erhaltenen Resultate vom zweiten Ansatz liefern eine gute Ausgangsbasis (siehe Abbildung 33), um im nächsten Schritt aus den gefundenen Ecken die Polygonzüge zu erstellen.

3.4 Erstellung der Polygonzüge

Um nun Polygonzüge aus den zuvor gefundenen Ecken zu erstellen, wird eine abgewandelte Form des Dijkstra Algorithmus verwendet. Der Dijkstra Algorithmus wird in der Praxis verwendet, um kürzeste Pfade in Graphen herauszufinden (vgl. [35, Seite 454-458], [24, Seite 324]).

Als Eingabeparameter wird das Kantenbild vom angepassten Canny-Kantendetektor verwendet. Bei diesem Bild handelt es sich um ein Binärbild. In diesem Fall sind die Räume schwarz und die Wände weiß darauf dargestellt. Ein weiterer Eingabeparameter ist eine Liste vom Harris-Eckendetektor mit den bereinigten Ecken.

Im ersten Schritt wird eine Kostenmatrix aufgestellt, welche die Distanz vom Startpixel bis zum Zielpixel repräsentiert (vgl. [34, Seite 226-228], siehe Abbildung 28). Als Startpixel wird eine beliebige Ecke der äußersten Wand verwendet. In der Implementierung ist dies die oberste rechte Ecke. Das Zielpixel hingegen repräsentiert jene Ecke, die dem Startpixel die am nächsten gelegene Ecke entlang der Wand ist.

Der Dijkstra Algorithmus läuft nur über die zuvor lokalisierten Wände im Bild. Für eine bessere Distanzdarstellung in der Kostenmatrix wird eine vierer statt achter Nachbarschaft gewählt (siehe Abbildung 27). Jedes Pixel hat somit vier direkte Nachbarn, welche vom Dijkstra Algorithmus geprüft werden.

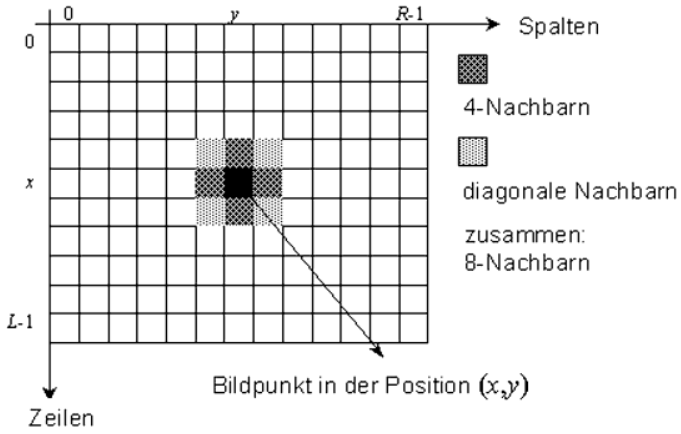

Abbildung 27: Auf dem Bild befindet sich eine vierer und achter Nachbarschaft. Eine vierer Nachbarschaft besteht aus vier Pixeln, die sich um einem bestimmten Pixel horizontal und vertikal anordnen. Hingegen besteht eine achter Nachbarschaft aus der vierer Nachbarschaft und den vier diagonal angrenzenden Pixeln.
(Quelle: [27, Seite 34])

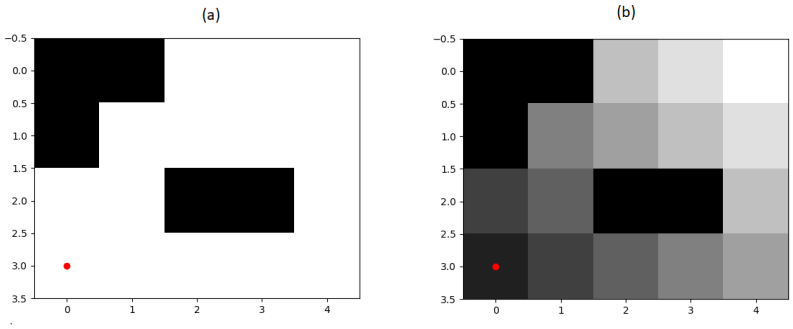

Abbildung 28: Aufstellung der Kostenmatrix vom abgewandelten Dijkstra Algorithmus wird anhand eines vereinfachten Beispiels dargestellt. (a) zeigt das Startbild. Der rote Punkt ist das aktuelle Startpixel. Die Wände sind im Bild weiß eingezeichnet. Die Räume sind schwarz dargestellt. (b) zeigt die Kostenmatrix, welche zum Startpixel gehört. Die schwarzen Pixel bleiben unverändert, da die Kosten nur auf den Wand-Pixeln berechnet werden. Aus dem Intensitätswert ist die Entfernung der einzelnen Pixel abzulesen. Das weiße Pixel oben rechts hat die größte Distanz zum Startpixel. (Bildschirmabbildung wurde in Python3 erstellt.)

Das Startpixel in dieser Kostenmatrix hat initiale Kosten in Höhe von eins. Bei den Kosten von den Nachbarn wird eins auf die aktuellen Kosten addiert. Selbiges gilt für deren Nachbarn und alle weiteren gefunden Nachbarn. Während der Berechnung der Kostenmatrix werden mehrere Überprüfungen durchgeführt:

- Befindet sich das aktuelle Pixel nicht auf der Wand, wird dieses Pixel verworfen.
- Falls bereits Kosten für das aktuelle Pixel eingetragen sind, ist das Pixel bereits besucht worden und wird nun verworfen.
- Ist das aktuelle Pixel eine tatsächliche Ecke, so wird diese als nächste Ecke im Polygonzug erkannt und der Algorithmus beendet.
- Ist keine der vorherigen Bedingungen eingetreten, wird jedes benachbartes Pixel von diesem Pixel in die Job-Liste eingetragen und in der nächsten Iteration bearbeitet.

Wurde eine Ecke gefunden, wird eine Kante zwischen Startpixel und gefundenen Ecke in dem Polygonzug gespeichert. Nun wird die eben gefundene Ecke aus der Liste aller Ecken entfernt und als neues Startpixel gesetzt. Dieser Vorgang wird so oft wiederholt, bis keine weiteren Ecken auf den Wänden zu finden sind. Der Pseudocode zum Erstellen eines Polygonzugs kann aus Programmcode 3 entnommen werden.

```
def apply_customized_dijkstra(image_path):
  #1. Schritt: Kantenbild generieren
  canny_image=apply_customized_canny(image_path)
  #2. Schritt: Harris-Eckendetector
  corners[]=apply_customized_harris(image_path)
  #3. Schritt: Polygonzug erstellen
  polygon = new Polygon()
  #4. Schritt: Startecke festlegen und im Polygonzug hinzufügen
  start_corner = get_start(corners[])
  polygon.add_point(start_corner)
  #5. Schritt: Polygonzug vervollständigen
  while polygon not finished:
    neighbor= apply_dijkstra(start_corner)
    polygon.add_point(neighbor)
    start_corner = neighbor
  return polygon
```

Programmcode 3: Polygonzugerstellung

Beim Testlauf hat sich gezeigt, dass der Algorithmus grundsätzlich in der Lage ist, Polygonzüge aus den Grundrissbildern zu extrahieren (siehe Abbildung 29 und 34). Dennoch sind mehrere Probleme aufgetreten, welche noch zu beheben sind:

- Die Polygonzüge können nur dann richtig erstellt werden, wenn alle relevanten Wände zuvor mit dem Canny-Kantendetektor detektiert wurden. Wurden Wände übersehen, dann können ebenso dessen Ecken nicht gefunden werden. Dies hat zur Folge, dass die resultierenden Polygonzüge den Grundriss (teilweise) nicht korrekt wiedergeben. Die Güte des Polygonzugs ist somit stark davon abhängig, ob alle relevanten Ecken zuvor detektiert wurden.

- Ebenfalls entstehen Probleme, wenn die gefundene Ecke nicht auf einer Wand liegt. Ist dies der Fall, so wird der Algorithmus keine benachbarten Ecken mehr finden und somit den Polygonzug hier abschließen.

- Zusätzlich muss ergänzt werden, dass sich auf dem vereinfachten Grundrissbild nur ein Grundriss von einem Gebäude befinden darf, da sonst beispielsweise das zweite Gebäude als innerer Polygonzug in pFlow angezeigt wird, weil pFlow für die Darstellung nur ein einzigen äußeren Polygonzug zulässt.

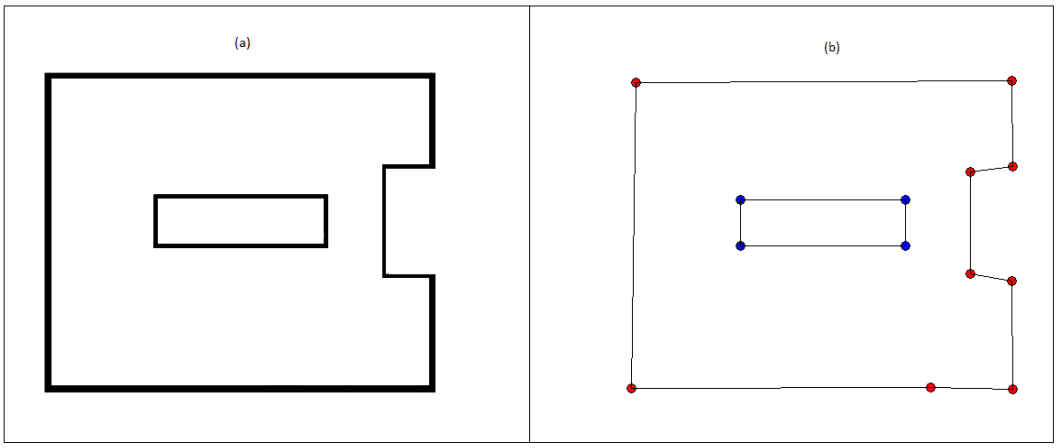

Abbildung 29: (a) zeigt ein vereinfachtes Grundrissbild. (b) stellt die vom Algorithmus daraus extrahierte Polygonzüge dar. Die roten Punkte definieren den äußeren Polygonzug, wohingegen die blauen Punkte den Umriss des Hindernisses wiedergibt. (Bildschirmabbildung wurde in Python3 erstellt.)

Damit der Algorithmus ein gutes Ergebnis liefert, soll das Eingabe-Bild bestimmte Bedingungen erfüllen:

- Es darf sich nur ein Grundriss von einem Gebäude auf dem Bild befinden, da ansonsten die Erstellung der Polygonzüge fehlerhaft wird.
- Es dürfen keine Türen, Treppen und Hindernisse im Bild enthalten sein.
- Es dürfen keine Beschriftungen (also: kein Text und Sprechblasen, keine Schilder mit Notausgang etc.) eingezeichnet sein.
- Die Ecken müssen mit jeweils zwei Linien abgebildet werden. Diese stehen im besten Fall in einem 90 Grad Winkel zueinander. Dadurch können die Kanten sowie Ecken entsprechend detektiert werden.
- Die eingezeichneten Wände im Grundrissbild müssen horizontal oder vertikal verlaufen. Dies ist z. B. nicht der Fall, wenn das Bild schief eingescannt wurde. Dann müsste die Schiefe des Bild vorher korrigiert werden, sodass die eben genannte Bedingung gegeben ist.

Sind die Bedingungen für ein vereinfachtes Grundrissbild nicht gegeben, so werden unterschiedliche Probleme auftreten. Die Abbildung 30 zeigt den tatsächlichen Grundriss des O-Gebäudes. Im Gegensatz zur vorherigen Abbildung 29 ist das Ergebnis sehr fehlerhaft. Als äußerer Polygonzug wurde das Treppenhaus angenommen (siehe Abbildung 30/(a)). Somit liegen alle inneren Polygonzüge außerhalb des äußeren Polygonzugs, was der Logik von „innen“ und „außen“ widerspricht. Ebenso sind die Außenwände schlecht detektiert, da Fenster Artefakte hinterlassen. Diese Artefakte führen zum Zerfall der Polygonzüge, was deutlich in der oberen rechten Ecke zu sehen ist. Auch die Raum-Bezeichnungen verursachen Eckpunkte, welche der Algorithmus verbindet. Dies ist unten links in der Abbildung ersichtlich. Wie zu erwarten, wird in der Mitte von den Türen eine Ecke detektiert. Diese befindet sich nicht auf der Wand und bleibt somit unter Umständen alleine stehen (siehe Tür von Raum O004). Zusammenfassend sind die Polygonzüge aus Abbildung 30/(b) für eine Simulation unbrauchbar.

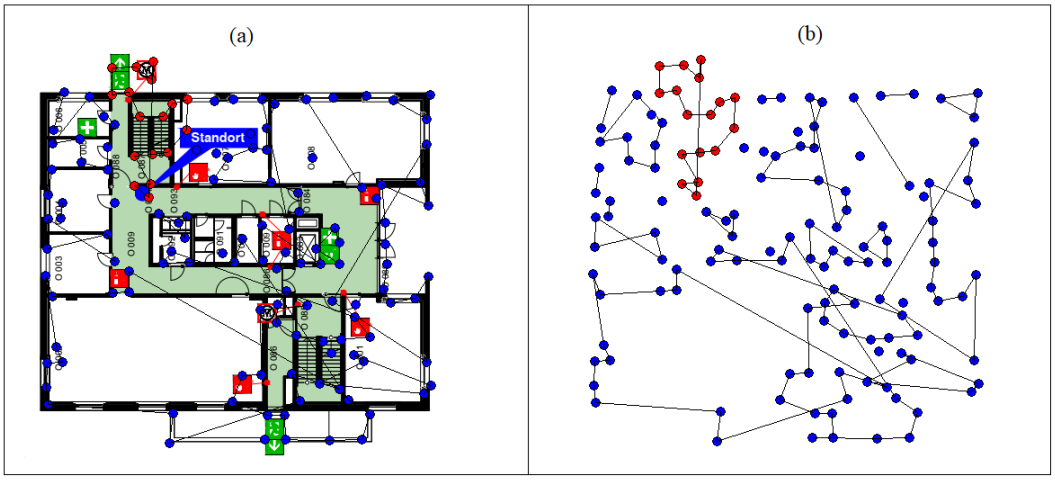

Abbildung 30: (a) stellt ein komplexes Grundrissbild mit den gefundenen Polygonzügen dar. (b) zeigt nur die vom Algorithmus daraus extrahierte Polygonzüge. (Bildschirmabbildung wurde in Python3 erstellt.)

3.5 Weitere aufgetretene Probleme

Bei der Implementierung sind einige Probleme aufgetreten. Um dem Leser diese zu ersparen, sind die Wichtigsten nachfolgend erklärt.

Farbtransformation
Beim Einlesen des Bildes wird das Bild standardmäßig von OpenCV mit BGR (Blau - Grün - Rot) statt RGB (Rot-Grün- Blau) eingelesen. Dadurch hatten alle eingelesen Farbbilder plötzlich einen Blaustich und bei den RGB Histogrammen wurden Blau- und Rot-Werte vertauscht. Abhilfe schafft die Zeile `cv2.cvtColor(self.image, cv2.COLOR_BGR2RGB)`, damit wird das BGR-Bild in ein RGB-Bild transformiert.

Da in dieser Bachelorarbeit mit Grauwertbildern gearbeitet wird, wird beim Einlesen des Bildes der Parameter `flags = 0` beim Einlesen des Bildes `cv2.imread(filename=image_path, flags = 0 )` gesetzt werden.

Beim Darstellen von Grauwertbildern muss bei matplotlib der Parameter `cmap="gray"` bei `plt.imshow(X=image, cmap="gray")` mit übergeben werden, um das Bild richtig darzustellen. matplot macht bei der Methode `imshow()` immer eine automatische Kontrastanpassung, d.h. der kleinste Intensitätswert wird als schwarz und der größte als weiß dargestellt. Um dies zu verhindern, werden die Parameter `vmin` für schwarz, `vmax` für weiß.

x- und y-Achse vertauscht
Beim Zugriff auf ein Element eines numpy-Arrays erfolgt dieser über y, x statt, wie bei Koordinaten üblich, x, y. Zudem wird beim Darstellen von Bildern via matplotlib die y-Achse häufig invertiert (vgl. [16, Seite 1]). Somit sind größere y-Werte weiter unten im Bild. Der Ursprung $(0, 0)$ liegt oben links.

4 Fazit und Ausblick

Die zum Eingang formulierte Forschungsfrage ist, wie Polygonzüge aus einem Grundrissbild automatisiert ermittelt werden. Um dies herauszufinden, wurden zunächst Grundlagen der Bildverarbeitung im Kapitel 2 erläutert. Danach wurden verschiedene Implementierungsversuche im Kapitel 3 durchgeführt und dokumentiert.

Ein automatisiertes Erkennen von Polygonzügen wurde wie folgt erreicht:

Für die Vorverarbeitung wurde ein Closing, ein Canny-Kantendetektor und danach nochmal ein Closing durchgeführt. Das erste Closing dient zum Schließen eventueller Lücken bzw. Störungen in den Wänden. Der Canny-Kantendetektor wird für Kantenlokalisation eingesetzt. Das zweite Closing wird zum Schließen der Bereiche zwischen den gefundenen Kanten verwendet.

Anschließend wurden auf dem Kantenbild die Ecken mittels Harris-Eckendetektor detektiert. Jene Ecken, die innerhalb einer vordefinierten maximalen Distanz liegen, wurden in Cluster zusammengefasst. Aus den jeweiligen Clustern wurde eine resultierende Ecke via Mittelwertbildung berechnet.

Diese Ecken wurden einer abgewandelten Form des Dijkstra Algorithmus übergeben, welcher anhand des Kantenbildes und den gemittelten Ecken Polygonzüge generiert. Die generierten Polygonzüge können anschließend im vorgesehenen Dateiformat für pFlow abgespeichert werden.

In dieser Bachelorarbeit wurde gezeigt, dass es grundsätzlich möglich ist, aus einem vereinfachten Grundrissbildes Polygonzüge zu erstellen. Je komplexer allerdings das übergebene Grundrissbild ist, desto mehr muss der Algorithmus entsprechend angepasst werden, um mit Störungen besser umgehen zu können. Auch sollte es ermöglicht werden, in der Graphical User Interface (GUI) grundlegende Einstellungen – z. B. die Stärke des Glättens – abhängig vom gegebenen Grundriss einstellen zu können.

Abgesehen von den Punkten, die N. Steinbrügge bereits erwähnt hat (siehe [31, Seite 21-24]), wären noch folgende Features für die pFlow-Anwendung sehr vorteilhaft:

Bildgröße ändern
Aktuell werden zu große geladene Grundrissbilder abgeschnitten angezeigt. Abhilfe könnte durch entsprechendes Skalieren des Bildes geschaffen werden. Die Logik ist bereits im `ImageCleaner.py` implementiert.

Bild rotieren
Wenn ein Grundrissbild schief oder auf dem Kopf eingescannt wird, kann dies durch entsprechende Rotation des Bildes behoben werden. Die Logik zum Rotieren ist bereits im `ImageCleaner.py` implementiert.

Sensitivitätsregler zur Erstellung der Polygonzüge
Beim Erkennen der Wände ist die Güte des Ergebnisses stark von den Parametern abhängig. Beispielsweise sollte ein Grundbild, je detailreicher es ist, desto stärker geglättet werden. Oder es weist unterschiedlich dicke Wände auf.

Um den Algorithmus möglichst zielführend anzuwenden, würde es sich anbieten, einige Parameter durch Slider je nach Grundrissbild anzupassen. Der Hintergrund hierfür ist, dass Türen, Treppen sowie Beschriftungen schwer zu extrahieren und stark vom Bild abhängig sind.

Auch beim Erstellen der Polygonzüge wird das Ergebnis unterschiedlich je nach Wandstärke und Bildauflösung ausfallen. Ebenfalls wäre denkbar, Texte durch ein gehäuftes Auftreten von Ecken zu erkennen. Diese Texte könnten dann ignoriert werden.

Scharfes Bildformat
Es gibt Ansätze, um eine Pixelgrafik zur Vektorgrafik zu konvertieren (siehe [2, Seite 192-193]). Dadurch entsteht der Vorteil, dass die Bilder deutlich schärfere Kanten haben. Abhängig wie gut dies auf Grundrissbildern funktioniert, könnten hier hilfreiche Informationen gewonnen werden.

Objekterkennung
In der Bildvorverarbeitung könnte mithilfe von Machine Learning Treppen, Hindernisse oder Beschriftungen, die ein erhöhtes Vorkommen an Ecken in einem Bereich aufweisen, identifiziert und entsprechend markiert werden. Abhängig von der Art des Hindernisses könnte angegeben werden, wie viele Personen sich auf den Hindernis befinden dürfen.

Literaturverzeichnis

[1] Anja. Bauplan - Grundriss - Architektenplan - Architektenentwurf. https://cdn.pixabay.com/photo/2014/05/26/07/53/building-plan-354233_960_720.jpg, 2014. Stand: 13.12.2021.

[2] J. Böhringer, P. Bühler, P. Schlaich, and D. Sinner. *Kompendium der Mediengestaltung*. Springer Berlin Heidelberg, 2014.

[3] G. Bourier. *Beschreibende Statistik praxisorientierte Einführung ; mit Aufgaben und Lösungen*. Springer Gabler, Wiesbaden, 2012.

[4] J. Browning. *Pro Python*. Apress, New York, 2014.

[5] P. Bühler, P. Schlaich, and D. Sinner. *Digitales Bild*. Springer Berlin Heidelberg, 2017.

[6] W. Burger and M. J. Burge. *Digitale Bildverarbeitung*. Springer Berlin Heidelberg, 3., vollständig überarbeitete und erweiterteauflage edition, 2015.

[7] Dudenredaktion. Bild. https://www.duden.de/node/22513/revision/611515, (o.J). Stand: 30.11.2021.

[8] Dudenredaktion. Grundriss. https://www.duden.de/node/60880/revision/567901, (o.J). Stand: 5.12.2021.

[9] Dudenredaktion. Kontur. https://www.duden.de/node/82536/revision/600996, (o.J). Stand: 22.01.2022.

[10] Dudenredaktion. Pixel. https://www.duden.de/node/111961/revision/533557, (o.J). Stand: 30.11.2021.

[11] Dudenredaktion. Polygon. https://www.duden.de/node/113113/revision/530366, (o.J). Stand: 22.11.2021.

[12] A. Erhardt. *Einführung in die digitale Bildverarbeitung Grundlagen, Systeme und Anwendungen ; mit 35 Beispielen und 44 Aufgaben*. Vieweg + Teubner, Wiesbaden, 2008.

[13] M. O. Franz. Industrielle Bildverarbeitung - Einleitung. http://www-home.fh-konstanz.de/~mfranz/ibv_files/lect00_Einleitung.pdf, Oct. 2007. Stand: 25.11.2021.

[14] E. Gastal and M. Oliveira. Domain transform for edge-aware image and video processing. *ACM Trans. Graph.*, 30:69, 07 2011.

[15] T. Gockel. *Kompendium digitale Fotografie Von der Theorie zur erfolgreichen Fotopraxis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[16] T. Haenselmann. Bildverarbeitung WS2016/17 - CCD-Sensoren. https://www.cb.hs-mittweida.de/fileadmin/verzeichnisfreigaben/haenselm/dokumente/dbv_2016w_color.pdf, Oct. 2016. Stand: 09.02.2022.

[17] M. L. Hetland. *Beginning Python*. Apress, 2017.

[18] H. Hirsch-Kreinsen and A. Karacic, editors. *Autonome Systeme und Arbeit*. transcript Verlag, Jan. 2019.

[19] Hochschule Konstanz. Von Personenströmen und Ansteckungswahrscheinlichkeiten. https://www.htwg-konstanz.de/hochschule/fakultaeten/informatik/uebersicht/informatik-news/news/von-personenstroemen-und-ansteckungswahrscheinlichkeiten/, 2021. Stand: 24.01.2022.

[20] A. Huamán, L. García, T. Tsesmelis, and K. Vladislav. Opencv: Image processing. https://docs.opencv.org/3.4/d7/da8/tutorial_table_of_content_imgproc.html, 2022. Stand: 13.01.2022.

[21] B. Jähne. *Digitale Bildverarbeitung*. Springer Berlin Heidelberg, 2012.

[22] S. Kapil. *Clean Python*. Apress, 2019.

[23] W. Konen. Der harris-eckendetektor und die strukturmatrix. http://www.gm.fh-koeln.de/~konen/WPF-BV/TR-Harris-Strukturmatrix.pdf, 2006. Stand: 10.02.2022.

[24] B. Kreussler and G. Pfister. *Mathematik für Informatiker*. Springer Berlin Heidelberg, 2009.

[25] C. Musel. Zu meinem hausbau-entwürfen. http://www.digizeitschriften.de/dms/img/?PID=urn%3Anbn%3Ade%3Absz%3A16-diglit-85387%7Clog00010, oJ. Stand: 10.02.2022.

[26] B. Neumann. *Bildverarbeitung für Einsteiger Programmbeispiele mit Mathcad ; mit 45 Tabellen*. Springer, BerlinHeidelbergNew York, 2005.

[27] A. Nischwitz. *Bildverarbeitung*. Springer, 4. edition, 2020.

[28] OpenCV. OpenCV. https://docs.opencv.org/3.4/index.html, (o.J). Stand: 24.01.2022.

[29] T. Peters. Pep 20 – the zen of python. https://www.python.org/dev/peps/pep-0020/, 2004. Stand: 13.01.2022.

[30] S. S. Skiena. *The Algorithm Design Manual*. Springer International Publishing, 2020.

[31] N. Steinbrügge. Entwicklung einer Software zur 2D-Modellierung von Räumen und Berechnung einer Delaunay-Triangulierung, Aug. 2021.

[32] T. C. Stocker and I. Steinke. *Statistik*. De Gruyter Oldenbourg, 11 2016.

[33] W. Struckmann and D. Wätjen. *Mathematik für Informatiker*. Springer Berlin Heidelberg, 2016.

[34] H. Süße and E. Rodner. *Bildverarbeitung und Objekterkennung*. Springer Fachmedien Wiesbaden, 2014.

[35] G. Teschl and S. Teschl. *Mathematik für Informatiker*. Springer Berlin Heidelberg, 2013.

[36] L. W. A panda at the zoo in copenhagen, denmark. https://unsplash.com/photos/e3mu-MTj7Xk, 2020. Stand: 18.01.2022.

[37] M. Wäger. *Adobe Photoshop*. Rheinwerk Verlag GmbH, Aug. 2021.

[38] Wikipedia. Lenna. https://en.wikipedia.org/wiki/Lenna, 2021. Stand: 05.11.2021.

[39] Wikipedia. Polygon. https://de.wikipedia.org/wiki/Polygon, 2021. Stand: 22.11.2021.

Ehrenwörtliche Erklärung

Hiermit erkläre ich, Julia Hansi, geboren am 23. März 1993 in Hainburg a. d. Donau,

1. dass ich meine Bachelorarbeit mit dem Titel:

   „Automatisiertes Erkennen von Polygonzügen aus Grundrissbildern“

   in der Fakultät Informatik unter Anleitung von Professor Bohnet selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als die angeführten Hilfen benutzt habe;

2. dass ich die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

3. dass die eingereichten Abgabe-Exemplare in Papierform und im PDF-Format vollständig übereinstimmen.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 21. Februar 2022

Anhang

Link zum Git-Repository: `https://github.com/ju851han/pFlow-EdgeDetector`

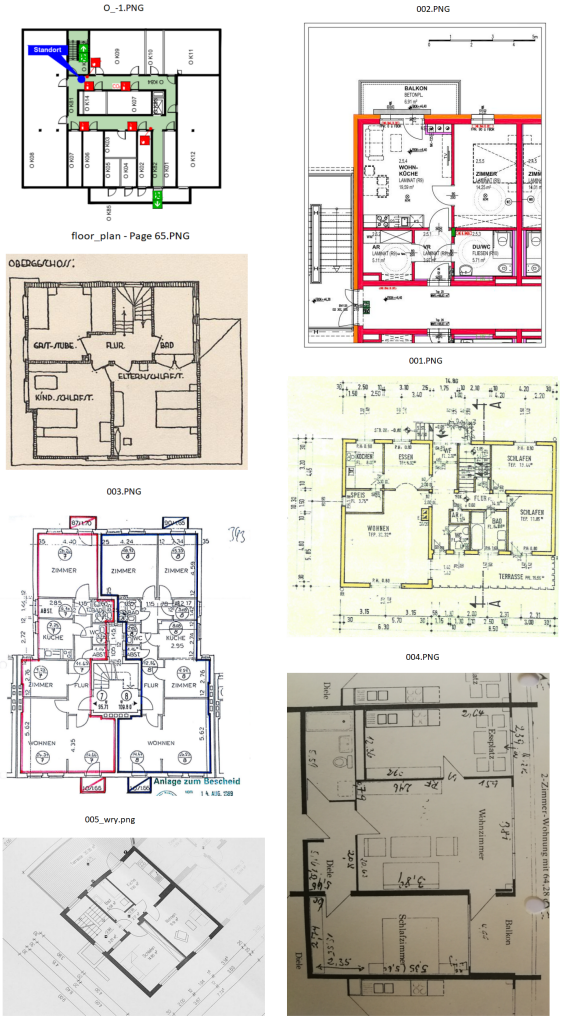

Abbildung 31: Diese unterschiedlichen Grundrissbilder wurden als Testbilder verwendet. (`O_-1.PNG` ist der Keller des O-Gebäudes von HTWG. Quelle von `floor_plan - Page 65.PNG` : [25]. Quelle von `005_wry.png`: [1])

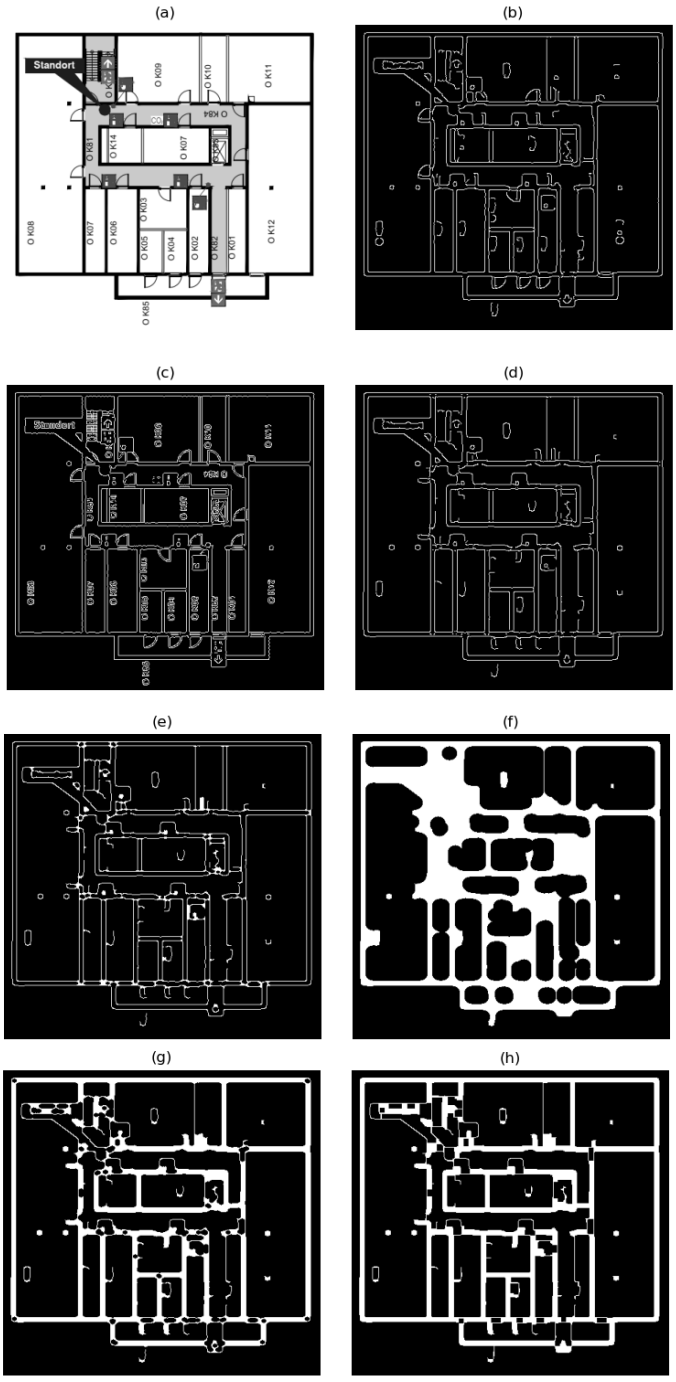

Abbildung 32: (a) ist das Eingangsbild vom Grundriss des Erdgeschosses vom O-Gebäude der HTWG. Die Bilder (b) bis (h) wurden mit der Kombination von Canny-Kantendetektor und diversen Filter vor- und nachverarbeitet. (Bildschirmabbildung wurde in Python3 erstellt.)

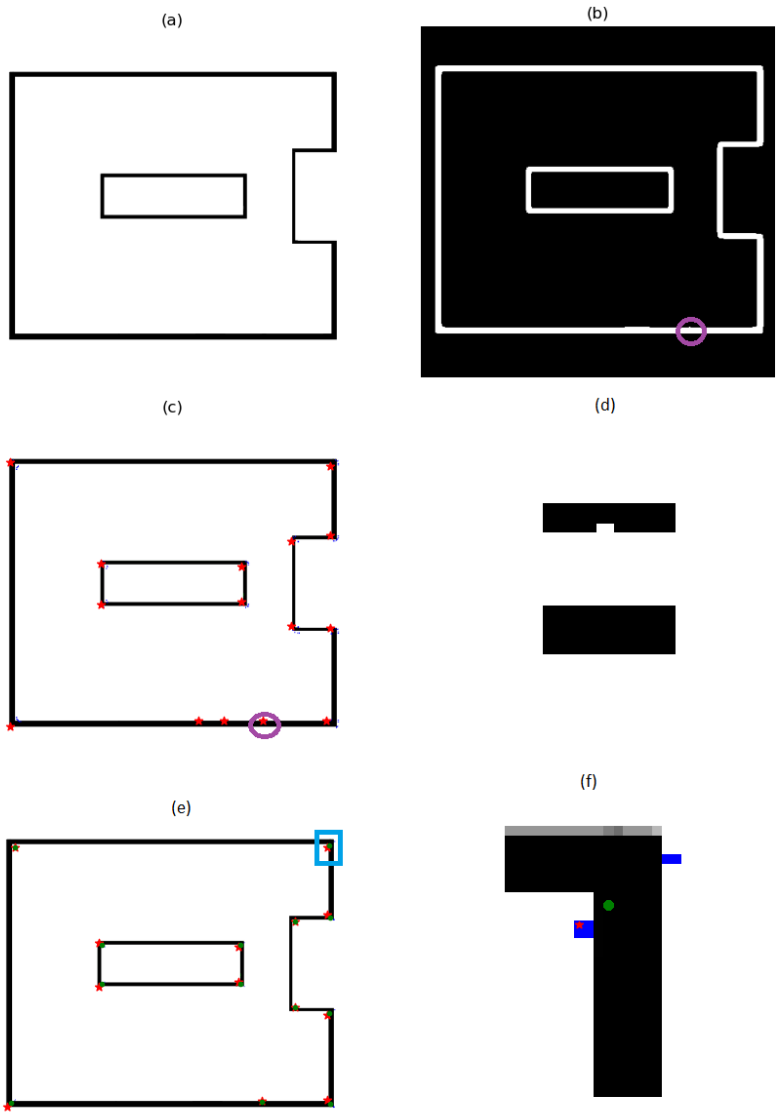

Abbildung 33: (a) ist das Originalbild vom vereinfachten Grundriss des Erdgeschosses vom O-Gebäude der HTWG. (b) ist (a), auf dem die definierten Vor- und Nachverarbeitung sowie Canny-Kantendetektor angewendet wird (siehe Kapitel 3.1). In (c) sind alle gefundenen Ecken vom 1.Versuch (siehe Kapitel 3.3) rot eingezeichnet. (d) stellt den violett eingekreisten Bereich von (b) vergrößert dar. (e) zeigt das Ergebnis von Versuch 2 mit angepassten Parametern. Blau markiert sind die gefundenen Ecken vom Harris-Eckendetektor. Rot markiert sind die ausgewählten Ecken vom 1. Versuch und grün markiert ist die ausgewählte Ecke vom 2. Versuch. (f) wird der hellblau umrandete Bildausschnitt vergrößert dargestellt. (Bildschirmabbildung wurde in Python3 erstellt.)

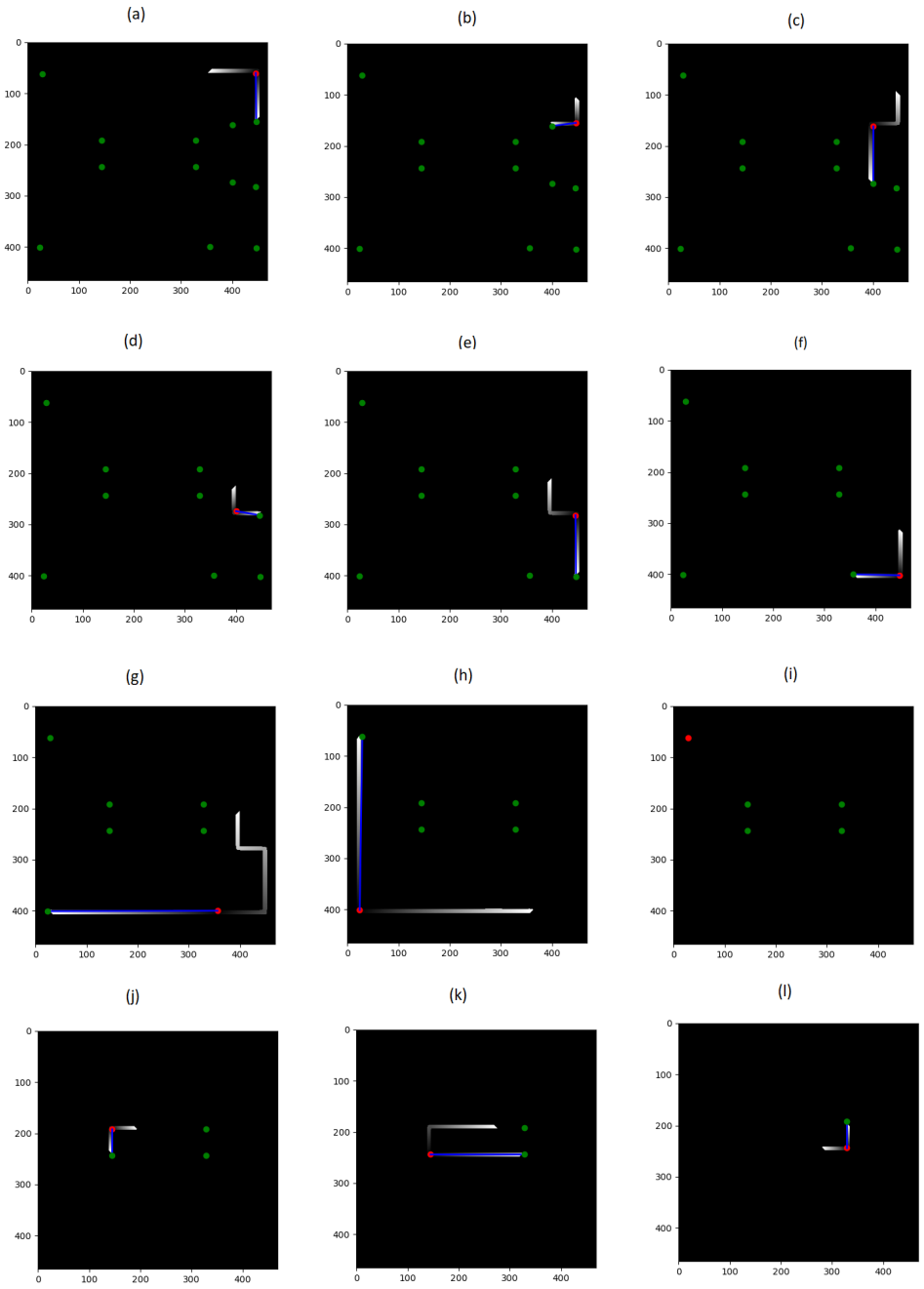

Abbildung 34: Schrittweiser Aufbau der Polygone. Der rote Punkt ist der aktueller Startpixel. Grüne Punkte geben noch nicht verbundene Ecken wieder. Weiß bis graue Linien stellen die Kosten der Matrix dar. Die blaue Linie ist die neu gefundene Kante.
(Bildschirmabbildung wurde in Python3 erstellt.)

Abkürzungsverzeichnis

GUI	Graphical User Interface
HTWG	Hochschule Konstanz Technik, Wirtschaft und Gestaltung
IDE	Integrated Development Environment
JPEG	Joint Photographic Experts Group
KI	Künstliche Intelligenz
ksize	Kernel Size
OCR	Optical Character Recognition
OpenCV	Open Source Computer Vision
PEP	Python Enhancement Proposals
PNG	Portable Network Graphics
SVG	Scalable Vector Graphics

Variablenverzeichnis

n	Anzahl Bits
k	Anzahl der Histogrammeinträge (Bins)
I	(Quell-)Bild
I'	Zielbild
$I(u, v)$	(Quell-)Pixel bzw. Bildkoordinate
u, v	u ist die Koordinate auf x-Achse; v ist die Koordinate auf y-Achse
$I'(u, v)$	Zielpixel
h	Histogramm
i, j	Index bzw. Stelle (im Histogramm (Bin))
$h(i)$	Anzahl der vorkommenden Pixel mit dem Intensitätswert i
a	Kontrastumfang
a_{min}	Untergrenze vom Kontrastumfang
a_{max}	Obergrenze vom Kontrastumfang
a_0	Schwellenwert-Klasse 1
a_1	Schwellenwert-Klasse 2
$f_{threshold}(I(u, v))$	Funktion für Schwellenwertoperation
q	Schwellenwert
q_0	unterer Schwellenwert für Hysterese-Schwellenwertoperation
q_1	oberer Schwellenwert für Hysterese-Schwellenwertoperation
M	Filtermatrix
$O(i, j)$	Filteroperation
R	Filterregion
S	Strukturmatrix

Abbildungsverzeichnis

Tabellenverzeichnis

Programmcodeverzeichnis