# Verarbeitung von Grundrissen

**EDBV WS 2019/2020: AG_E5**

Leonhard Eder (00047514)
Raphael Schimmerl (00371366)
Florian Langeder (01527111)
Filip Hörtner (11808203)
Mark Alam (11808580)

6. Januar 2020

## 1 Gewählte Problemstellung

### 1.1 Ziel

Ziel des Projekts ist es, bestimmte Merkmale wie Wände, Türen, Fenster, Stiegen und Räume aus einem Grundrissplan zu erkennen und diese in einem Datensatz abzuspeichern.

### 1.2 Eingabe

Als Eingabe verwenden wir 2D-Grundrisspläne in MATLAB-kompatiblen Datentypen. Unser Projekt wird auf dem Dataset *cubicasa5k* (https://github.com/CubiCasa/CubiCasa5k) aufbauen, es können jedoch auch andere Grundrisse, die der in unseren Daten verwendeten Symbolik folgen, verarbeitet werden.

### 1.3 Ausgabe

Die Ausgabe ist eine Datei im JSON-Format, die die Auswertung der Analyse der Eingabedateien enthält. Enthalten sind: Anzahl der Räume, Fenster und Türen, Vorhandensein von Stiegen und die Größe der Räume (prozentuell).

### 1.4 Voraussetzungen und Bedingungen

Der Input besteht aus einem oder mehreren Graustufenbildern, die jeweils den Grundrissplan einer eingeschoßigen Wohnung darstellen. Die Notation der Symbole sollte der gängigen Symbolik entsprechen.

## 1.5 Methodik

Methodik-Pipeline

1. Umwandlung in ein Binärbild - Threshold nach Otsu
2. Entfernung von Details - Erosion
3. Erkennung der Türen und Abtrennung der Räume - Corner Detection/Mustererkennung
4. Flächenbestimmung - Connected Component Labeling
5. Erkennung der Fenster - Hit or Miss- Transformation
6. Erkennung der Stiegen - Threshold- Operationen, Erosion, Dilation

## 1.6 Evaluierungsfragen

- Stimmen die Anzahl der Räume, Türen, etc. überein?
- Werden alle Merkmale erkannt?
- Stimmen die Proportionen der Räume?
- Sind die berechneten Größen korrekt?
- Bei wie vielen Datensätzen wurde ein korrektes Ergebnis erzielt?

## 1.7 Zeitplan

| Meilenstein | abgeschlossen am | | Arbeitsaufwand in h | |
|---|---|---|---|---|
| | geplant | tatsächlich | geplant | tatsächlich |
| Erster Prototyp | 1.Dezember | 17.Dezember | 100h | 120h |
| Anzahl der Türen | 20.Dezember | 20.Dezember | 25h | 40h |
| Anzahl der Räume | 20.Dezember | 26.Dezember | 25h | 30h |
| Raumgrößen | 20.Dezember | 26.Dezember | 25h | 20h |
| Stiegenerkennung | 20.Dezember | 4.Jänner | 25h | 30h |
| Anzahl der Fenster | 20.Dezember | 5.Jänner | 25h | 20h |
| Tests, Evaluierung | 30.Dezember | 5.Jänner | 55h | 40h |
| Bericht | 3.Jänner | 6.Jänner | 20h | 15h |

# 2 Arbeitsteilung

| Name | Tätigkeiten |
|---|---|
| Leonhard Eder | findStairs<br>Bericht Abschnitt 3.7, 4.6 |
| Raphael Schimmerl | roomdetection, connectedComponentLabeling, hough_tr<br>Bericht Abschnitt 3.5, 4.4, 5 |
| Florian Langeder | windowdetection, hit_or_miss, remove_details, erosion, json output<br>Bericht Abschnitt 3.6, 4.5 |
| Filip Hörtner | doordetection, find_opposite, hough_tr<br>Bericht Abschnitt 3.1, 4.2, 4.3 |
| Mark Alam | dda, door_detection, corner_detection, wall_thickness, wall_thiccness<br>Bericht Abschnitt 3.2, 3.3, 3.4, 4.1 |

# 3 Methodik

## 3.1 Hough-Transformation

Die Hough-Transformation wird verwendet, um Linien oder andere Formen in einem Bild zu erkennen und zu lokalisieren. Da eine Linie mit zwei Parametern, zum Beispiel in der Form $y = k \cdot x + d$, beschrieben werden kann, kann für zwei Punkte $P(x_1, y_1)$ und $P(x_2, y_2)$ einfach überprüft werden, ob diese auf einer gemeinsamen Linie liegen, indem die beiden Gleichungen $y_1 = k \cdot x_1 + d$ und $y_2 = k \cdot x_2 + d$ erfüllt werden. Bei der Hough-Transformation sollen nun diejenigen Linien gefunden werden, auf denen möglichst viele Bildpunkte liegen. Es werden nun für alle Punkte $(x_i, y_i)$ die Parameter $k$ und $d$ variiert und für jede $k$-$d$ Kombination überprüft, wie viele Punkte auf dieser Geraden liegen (siehe Abbildung 1). [1]

Diese Übersetzung in den sogenannten Parameterraum führt zu einer neuen Form der Darstellung, bei der ein Punkt $(k_i, d_i)$ einer Geraden durch das Bild entspricht. Je mehr Punkte auf einer Linie liegen, desto höher ist der Wert an diesem Punkt, in der üblichen Darstellung wird dieser dann heller dargestellt. Punkte mit besonders hohen Werten entsprechen dann möglichen Linie im Ursprungsbild. Außerdem kann man durch die Positionen der hellen Punkte zueinander auf Verhältnisse der Linien schließen, zB sind parallele Linien auf einer $k$-Ebene, also horizontal nebeneinander (siehe auch Abbildung 2). [1]

In der Praxis wird die Parametrisierung mit $k$ und $d$ nicht verwendet, da bei vertikalen Linien $k = \infty$ gilt. Hier kommt eine andere Beschreibung von Geraden zum Einsatz, die Hesse'sche Normalform $x \cdot \cos(\theta) + y \cdot \sin(\theta) = r$. Auch hier kann man alle Geraden in einem zweidimensionalen Parameterraum beschreiben, daher funktioniert diese Methode fast analog zur oben beschriebenen. Ein Beispiel findet sich in Abbildung 2. [1]

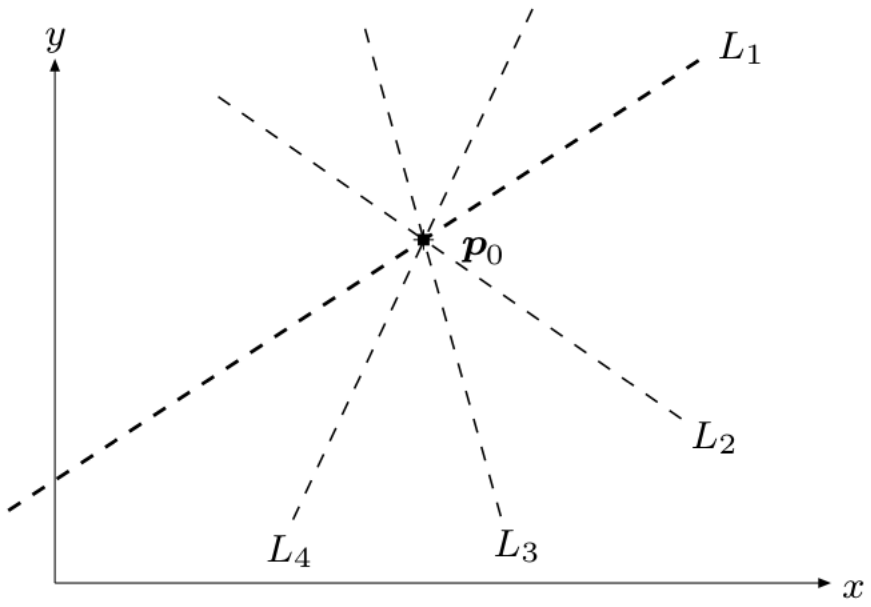
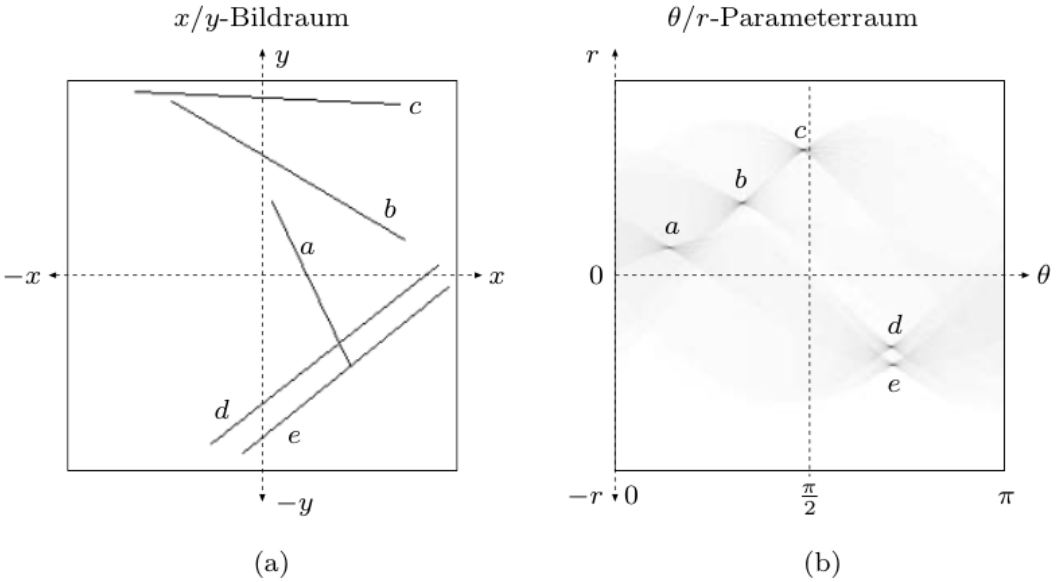



Abbildung 1: Geradenbündel durch einen Bildpunkt [1]




(a) (b)

Abbildung 2: (a): ein Bild mit Linien, (b): Repräsentation der Geraden im Parameterraum [1]

## 3.2 Threshold nach Otsu

Um ein digitales Bild in ein Binärbild umzuwandeln, muss das verwendete Material segmentiert werden. Hierfür kommen Schwellwertalgorithmen zu Gebrauch, die das Bild analysieren und die benötigten Bildpunkte von der Umgebung trennen.

Bei unserem Projekt haben wir dabei das Verfahren nach Otsu angewendet, bei dem erstmal ein RGB-Bild (img) übergeben wird, die Werte auf das Intervall [0;1] skaliert und nach an einem bestimmten, übergebenen Schwellenwert (x) die Segmentierung stattfindet. Hierbei werden alle Bildpixel mit dem Schwellenwert x verglichen, wobei alle Pixel größer dem Vergleichswert auf weiß gesetzt werden und kleiner gleich dem Wert auf schwarz. Außerdem gibt es einen zusätzlichen Parameter (invert) für die Invertierung des Bildes. Sofern der Parameter den Wert „1“ besitzt, wird das Bild invertiert.

Am Ende wird dann das resultierende Binärbild zurückgegeben.

## 3.3 Corner Detection

Bei der Corner Detection geht es vor allem um die Eckenerkennung. Die Türerkennung baut sich darauf auf.

Die Funktion nimmt zu aller Erst ein Binärbild (bin_img) entgegen. Dann wird jedes Pixel durchgegangen. Wenn das Pixel im Binärbild den Wert „0“ (schwarz) enthält, wird es ignoriert, da eher die Wände als potenzielle äußere Ecken dienen und den Wert „1“ (weiß) besitzen (Siehe Abbildung 3: Grüner Punkt) und schwarze Pixel eher als potenzielle innere Ecken (Siehe Abbildung 3: Roter Punkt).

Unter der Annahme, dass ein Pixel weiß ist, wird eine weitere Bedingung überprüft, und zwar folgende: Wenn das Pixel horizontal und vertikal jeweils ein „weißes“ und ein „schwarzes“ Nachbarpixel besitzt, dann ist das Pixel eine Ecke (Siehe Abbildung 3: Grüner Punkt), denn wenn das Pixel horizontal oder vertikal zwei „weiße“ Nachbarn haben würde, dann ist er innerhalb der Wand und kann kein Randpixel bzw. keine Ecke sein. Alle erkannten Ecken werden dann in einer M mal 2 Matrix (M ist die Anzahl der Ecken) gespeichert, wobei der erste Wert die y-Koordinate des Pixels angibt und der zweite die x-Koordinate. Am Ende wird die Matrix retourniert.

## 3.4 DDA-Algorithmus

Der DDA-Algorithmus dient zur Rasterung der Linien. Dabei werden folgende Parameter übergeben: matrix (das Binärbild), x1 und y1 (Koordinaten des ersten Pixels) und x2 und y2 (Koordinaten des zweiten Pixels). Nach dem gängigen Algorithmus wird die Linie Pixel für Pixel in die Bildmatrix eingeschrieben und schlussendlich die Matrix retourniert.

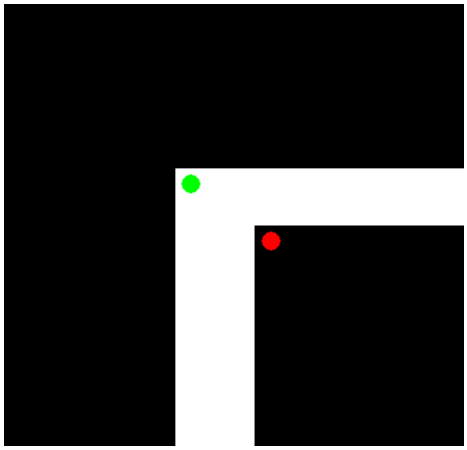


Abbildung 3: Grün: Akzeptierte Ecke (äußere Ecke), Rot: Abgelehnte Ecke (innere Ecke)

### 3.5 Connected Component Labeling

Für die Klassen roomdetection und wall_thickness haben wir die Methodik Connected Component Labeling (auch region labeling) verwendet. Diese vergleicht die benachbarten Pixelwerte eines Pixels und erstellt für jede Region für alle zusammenhängenden Pixel jeweils ein Lable. Dabei wird zwischen Vierer- und Achter-Nachbarschaft unterschieden. Die Vierer-Nachbarschaft vergleicht das obere, linke, untere und rechte Pixel mit dem momentanen ausgewählten Pixel. Die Achter-Nachbarschaft beschreibt zusätzlich die diagonal-liegenden Pixel. [1] In unserem Programm haben wir ausschließlich die Achter-Nachbarschaft verwendet. Da wir als Eingabe immer ein Binärbild haben, muss nur zwischen den Werten „0“ und „1“ unterschieden werden. Für Connected Component Labeling gibt es zum Beispiel folgende zwei Algorithmen:
Wenn das erste Pixel von einer verbundenen Komponente gefunden wird, dann werden alle verbundenen Pixel gelabelt, bevor es zum nächsten Pixel geht. Die Indices der Pixel werden in einer Datenstruktur gespeichert. Außerdem ist die Eingabe ein Binärbild und die Regionen mit weißen Pixeln („1“) werden jeweils gelabelt. Die Schritte des Algorithmus sind folgende:

1. Starte bei dem ersten Pixel des Bildes. Setze das momentane Label auf „1“ und geh zu Schritt 2.

2. Wenn das Pixel weiß und noch nicht gelabelt ist, label dieses Pixel, füge es in einer Queue als erstes Element ein und geh dann zum dritten Schritt. Wenn das momentane Pixel schon gelabelt ist oder schwarz ist, dann wiederhole den zweiten Schritt für das nächste Pixel.

3. Pop ein Element aus der Queue und vergleiche die Nachbarn anhand der Nachbarschaft. Wenn ein Pixel weiß ist und noch nicht gelabelt ist, gebe es dem aktuellen Label und füge es der Queue hinzu. Der dritte Schritt wird solange wiederholt bis in der Queue keine Elemente mehr vorhanden sind.

4. Geh zum zweiten Schritt für den nächsten Pixel im Bild und erhöhe die Labelnummer um eins.

Die Queue wird verwendet um die Nachbarn zu überprüfen und hinzuzufügen falls diese weiß sind.

Ein anderer Algorithmus ist folgender:
Dieser iteriert in einem zweidimensionalen Array, welches die jeweiligen Werte des Binärbildes speichert. Dabei gibt es zwei Durchläufe. Beim ersten werden temporäre Labels und Äquivalenzen gespeichert und beim zweiten werden jeweils die temporären Labels von den kleinsten Label von deren äquivalenten Klassen gespeichert. Dabei wird für jeden Pixel seine Achter-Nachabrschaft überprüft. [2]
Diesen haben wir auch als Methode in connectedComponentLabeling implementiert.

## 3.6 Hit-or-Miss Transformation

Die Erkennung der Fenster nimmt als Grundlage die morphologische Hit-or-Miss Transformation. Diese wird angewandt um Muster innerhalb eines Bildes zu erkennen. Als Vorverarbeitung wird das Bild mittels des Threshold durch Otsu in ein Binärbild umgewandelt.

Für die Hit-or-Miss Implementation werden 2 Strukturelemente in Form von Binärmatrizen herbeigenommen. Das erste beschreibt mit 1er Werten wo ein weißer Pixel vorhanden sein muss. Das zweite Strukturelement besitzt die exakt gleichen Dimensionen, nur definieren 1er wo sich kein Pixel befinden darf. Alle Koordinaten, bei denen in beiden Strukturelementen eine 0 ist werden als "Don't care"bezeichnet und geben keine weitere Aussage über den vorhandenen Pixelwert aus.

## 3.7 findStairs

Schwellenwert-Operationen: Einem Pixel wird abhängig davon, ob sich der Wert dieses Pixels unterhalb oder oberhalb eines bestimmten Schwellenwertes befindet, ein neuer Wert zugewiesen. Bei der Stiegenfindung kommen Schwellenwertoperationen vor allem über ganze Cluster von Pixel zum Einsatz: Sämtlichen Pixeln in einem Nachbarschaftscluster wird ein bestimmter Wert abhängig von der Summe über eben diesen Cluster zugewiesen.
Dilatation: Hierfür wird für jedes Pixel, das einen bestimmten Wert aufweist, in unserem Fall 0 oder eins, entsprechend einer Maske den Pixeln in der Nachbarschaft (in diesem Fall die 8er-Nachbarschaft) ebenfalls der entsprechende Wert zugewiesen. Dadurch ergibt sich eine Vergrößerung Fläche mit den entsprechenden Farbwerten.

# 4 Implementierung

## 4.1 Bestimmung der Wandstärke

Zur Bestimmung der durchschnittlichen Wandstärke wird zunächst das Binärbild übergeben. Zwei Matrizen hx, hy mit der Größe und den Daten des binären Originalbildes werden erzeugt, die für die Manipulation der Daten dienen, damit das Originalbild nicht überschrieben wird. Außerdem werden zwei weitere Matrizen Ax, Ay erstellt, die jeweils die Größe des Originalbildes besitzen und mit dem größten Integer (in MATLAB `intmax`) befüllt werden. Sie sind für das Speichern der Wanddicke jedes Pixels notwendig. Die Antwort auf die Frage, warum immer zwei Matrizen erzeugt werden, ist, dass wir für jede Operation jeweils horizontal und vertikal durchgehen.

Das Prinzip dieser Funktion beruht nun darauf, dass jedes Pixel einmal horizontal und einmal vertikal durchgegangen wird. Da die Wand in unserem Fall die Farbe "weiß" enthält, werden "weiße"Pixel bevorzugt und die anderen Pixel ignoriert. Wenn wir jetzt in der Schleife auf ein Wandpixel stoßen, erstellen wir einen Counter für die Wand (er zählt die folgenden "weißen"Pixel vom jetzigen Pixel weg, Wert am Anfang ist eins) und gehen alle darauf folgenden "weißen"Nachbarpixel durch bis wir auf ein ßchwarzesßtoßen. Dabei wird bei einem folgenden Wandpixel der Counter erhöht und das Pixel schwarz eingefärbt (in hx bei horizontaler Durchführung bzw. in hy bei vertikaler), damit es nicht noch einmal durchgegangen wird, da nur das Randpixel wichtig ist (Vom Rand beginnend bis zum anderen Rand). Am Ende wird dann der Wert statt dem intmax auf der jeweiligen Koordinate der jeweilgen Matrix gespeichert (in Ax bei horizontaler, in Ay bei vertikaler Durchführung).

Zum Schluss wird von den beiden Matrizen Ax und Ay das Minimum genommen und die intmax Werte herausgefiltert. (Sie besitzen keine Größe der Wanddicke). Da bei unserem Bildmaterial keine Wand eine Größe von mehr als 50 Pixel besitzt, werden nur kleinere Werte gefiltert in Betracht gezogen. Zu guter Letzt wird von allen übrigen Werten der Durchschnitt berechnet und der Wert dann für andere Funktionen zur Verfügung gestellt.

## 4.2 Bestimmung der Wandstärke (Prototyp)

In einer ersten Version der Wandstärkenbestimmung wurde versucht, mithilfe einer Hough-Transformation die Wände als Geraden zu erkennen und die Größen der Hotspots im Parameterraum abzumessen. Die durchschnittliche Wandstärke sollte sich laut unseren Überlegungen ungefähr proportional zur Größe dieser Felder verhalten. Nach einer Threshold-Operation verbleiben in der Matrix nur einige Felder, die aus 1-Werten bestehen. Allerdings war hier die Schwierigkeit, den richtigen Schwellwert zu erhalten und die Größe der Felder entsprechend umzurechnen. Da außerdem die Wandstärken innerhalb eines Bildes oft stark variieren, war diese Methode äußerst fehleranfällig und

wir sind auf die oben beschriebene Methode (4.1) umgestiegen.

## 4.3 Türenerkennung

Nach Entfernung der Details wie Beschriftungen und Mobiliar verbleiben nur mehr die Wände, welche keine Fenster mehr enthalten. Um die Türerkennung von der Symbolik unabhängig zu machen, und im weiteren Verlauf auch Räume zu erkennen, die ohne Türen, aber mit türähnlichen Öffnungen verbunden sind, wurde in Absprache mit den Kollegen folgende Definition einer Tür beschlossen: Jede Öffnung in einer Wand, die eine Breite von $2 \cdot w\_t$ bis $6 \cdot w\_t$ hat (wobei $w\_t$ der durchschnittlichen Wanddicke entspricht), und sich zwischen zwei Enden einer Wand befindet. Somit werden oft mehr Türen erkannt, als am Plan eingezeichnet werden, allerdings wird dies für die Raumabtrennung benötigt und definiert eine Tür als "Raumtrenner".
Die Türenerkennung setzt als zentrale Komponente auf die Eckenerkennung (siehe 3.3), die alle Ecken im Bild in einer 2×Anzahl großen Matrix speichert. Hier wird aus Performancegründen die in MATLAB implementierte Funktion `corner()` der selbstimplementierten vorgezogen, welche jedoch ähnliche Ergebnisse liefern würde. In Abbildung 4 werden die erkannten Ecken in rot eingezeichnet.

Die erkannten Ecken werden nun in mehreren Stufen gefiltert, sodass am Ende nur mehr Wandenden übrig bleiben. Die erste Stufe sortiert zunächst alle Ecken nach der X-Koordinate und filtert dann alle Punkte aus, die keinen Eckpunkt in einer Entfernung von $1,5 \cdot$w_t in X-Richtung haben. So werden schon alle false positives aussortiert, die irgendwo an einer Wand oder am Rand des Bildes aufgrund von Artefakten nach der Bereinigung erkannt werden.

In einer nächsten Filterungsstufe werden die euklidischen Abstände zwischen allen Punkte errechnet und jene Punkte, die in einem Umkreis von $0,003 \cdot$w_t bis $2 \cdot$w_t keinen anderen Punkt haben, aussortiert. Die Werte für die Entfernung wurden durch empirische Tests am Datenset festgelegt. Die untere Grenze ist zur Vermeidung von false positives aufgrund des Matchings mit sich selbst, die obere Grenze entspricht der durchschnittlichen höchsten Abweichung von der durchschnittlichen Wanddicke. Hier ist anzumerken, dass die Wandstärke teilweise sehr stark variiert und er hier zu fehlerhaften Messungen kommen kann. Es wurde ein Wert gewählt, der ein balanciertes Verhältnis zwischen Nichterkennung von sehr breiten Türen und Erkennung von anderen engen Öffnungen als Türen gewährleistet. Es ist aus unserer Sicht hier nicht möglich, alle Türen korrekt zu erkennen, da der Übergang von Tür zu Wandöffnung fließend ist und die Entscheidung oft auch der subjektiven Wahrnehmung des Betrachters obliegt.

Nachdem nun nur mehr Eckpunkte vorhanden sind, die einen weiteren Eckpunkt in näherer Umgebung haben, wird nun der Mittelpunkt zwischen den beiden errechnet (in Abbildung 4 grün eingezeichnet). Normal zur Verbindung der zwei Punkte wird im Mittelpunkt rund 3 Pixel in beiden Richtungen jeweils ein weiteren Testpunkt festgelegt. Diese beiden Testpunkte dienen der Eliminierung von false positives an Wandecken. Bei diesen werden die äußere und die innere Ecke erkannt und diese befinden sich oft in einem Abstand von weniger als $2 \cdot$w_t. Diese werden aussortiert, indem kontrolliert wird, ob jeweils einer der Punkte schwarz und der andere weiß ist. In Abbildung 4 ist der

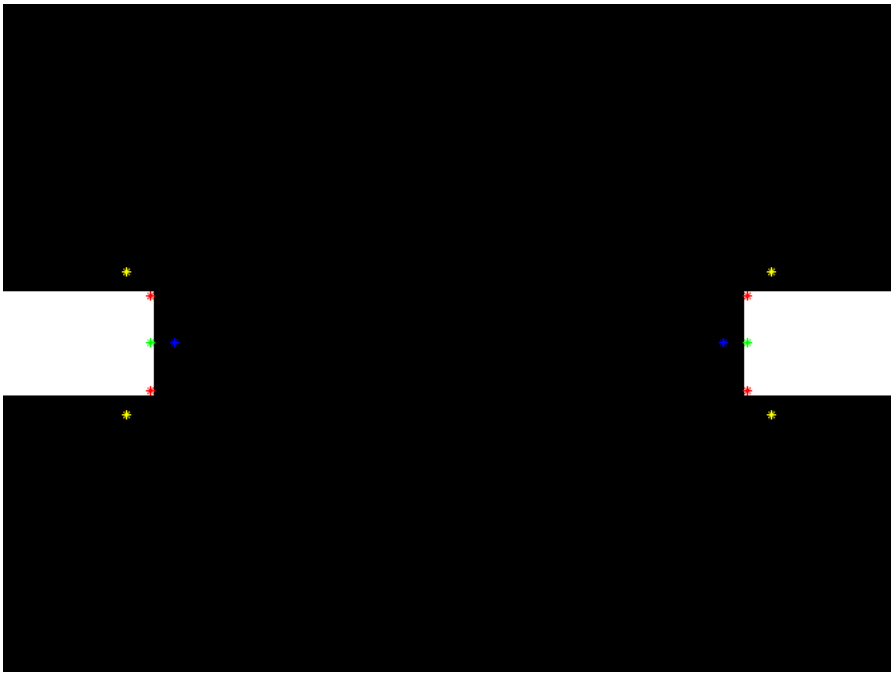


Abbildung 4: Zwei detektierte Wandenden, die eine Tür bilden

schwarze der beiden Punkte in blau eingezeichnet.

Hier bleiben weitere false positives, die entstehen, wenn die Entfernung der Details Ecken an den Wänden hinterlässt. So wird noch eine weitere Prüfung vorgenommen. Wenn an einer Seite der Kante schwarz und an der anderen weiß erkannt wurde, wird auf der weißen Seite (Wand) wieder im rechten Winkel $0,75 \cdot$w_t auf beiden Seiten überprüft, ob dort ein schwarzer Pixel ist. Das sorgt dafür, das nur Wandenden zurückbleiben. Die gelben Markierungen in Abbildung 4 zeigen diese Punkte.

Die Wandenden können natürlich auch alleine vorkommen, daher wird im letzten Schritt in einem Suchbereich, der $2 \cdot$w_t normal zur Kante entfernt ist und $4 \cdot$w_t$\times \sim$ 40Pixel groß ist, nach einem weiteren Punktepaar gesucht. Aufgrund der vorgegangenen Feststellung der schwarzen Seite der Kante (Abbildung 4 in blau) ist die Suchrichtung vorgegeben. Das gegenüber liegende Punktepaar muss natürlich parallel zum ersten Paar liegen, um einen Türstock oder eine Wandöffnung zu bilden. Daher wird die Methode zur Suche eines zweiten Punktpaares erneut aufgerufen, allerdings diesmal umgekehrt, sodass vom zweiten Paar aus nach dem ersten Paar gesucht wird. Wenn sich zwei Punktpaare gegenseitig gefunden haben, dann bedeutet das, dass diese in einem Abstand zueinander liegen, der einer Wandöffnung wie oben definiert entspricht und außerdem die Wandenden zu einander stehen.

## 4.4 Raumerkennung und Flächenbestimmung

Diese Methode gibt ein zweidimensionales Array zurück. In der ersten Spalte sind die jeweilige Raumnummer und in der zwei Spalte die jeweiligen Flächen in Prozent gespeichert. Der Parameter ist ein Binärbild. In unserem Fall ist dies der Plan in einem Binärbild mit den entfernten Details.
Zuerst wird das Komplement von dem Eingabebild erzeugt. Dies ist wichtig, da durch connectedComponentLabeling nur weiße Pixel gelabelt werden. Da allerdings connectedComponentLabeling eine wesentlich höhere Laufzeit aufweist (37 Minuten im Gegensatz zu 1 Minute bei einem Datensatz von fünf Bildern), wird die vorgefertigte Funktion bwlabel() verwendet. Dann wird die Matrix L anhand der Label gefiltert und sortiert. Die Sortierung ist notwendig, um das Label mit der größten Fläche, welches die Umgebung des Hauses ist, herauszufiltern. In der Schleife berechnen wir zum einem die jeweiligen Flächen der Räume und zum anderen die insgesamte Pixelgröße. Dadurch können wir in der zweiten Schleife die jeweiligen Prozente im Array abspeichern.// Die Raumgrößen werden als eigenes JSON-File ausgegeben, da durch die unterschiedlichen Anzahlen der Räume das Integrieren in das erste File durch MATLAB nicht möglich war.

## 4.5 Fenstererkennung

Zuerst wird das Eingabebild mittels Threshold nach Otsu in ein Binärbild umgewandelt. Als Threshold wurde dabei nach Herumprobieren ein Wert von 0.7 gewählt, der nach mehrfachem Testen graue Pixel der Fenster einheitlich eliminiert hat (oder in schwarze Pixel umgewandelt).

Für die Hit-or-miss Transformation werden insgesamt 4 Patterns verwendet. Eines für den linken Teil eines Fensters, eines für den rechten Teil und dann jeweils noch einmal für oben und unten bei vertikalen Fenstern. Durch die verschiedenen Größen der Fenster wird ein relativ simples Pattern verwendet, damit auch leicht abweichende Fenster erkannt werden. Aus Ausgangspunkt wurden 2 Strukturelemente definiert um ein wichtiges Merkmal auf der linken Seite eines horizontalen Fensters zu finden (Abbildung 5, Abbildung 6).

Diese wurden dann gespiegelt um den entsprechenden Teil auf der rechten Seite des Fensters zu finden und dann für vertikale Fenster entsprechend rotiert. Insgesamt werden also 4 Hit-or-miss Transformationen ausgeführt die jeweils eine Binärmatrix mit den Treffern produzieren.

Die Hit-or-miss Transformation wird folgendermaßen definiert:

$$A \circledast B = (A \ominus B1) \cap (A^c \ominus B2)$$ [1]

Zuerst wird eine Erosionsoperation durchgeführt mit dem Binärbild und dem ersten Strukturelement. Als Ergebnis kommt ein Binärbild mit allen Treffern bei denen die 1er übereinstimmen. Danach wird das Komplement von dem Original-Binärbild genommen und dieses mit dem zweiten Strukturelement erodiert. Aus den Resultaten von Schritt 1 und 2 wird dann die Schnittmenge gebildet für das Endergebnis.

[1] `https://docs.opencv.org/master/db/d06/tutorial_hitOrMiss.html` (Accessed 06-January-2020)

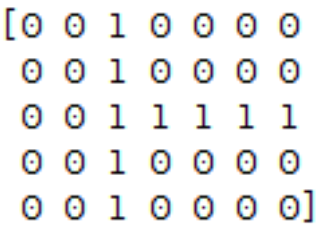



Abbildung 5: Strukturelement 1 - Horizontales Fenster - links

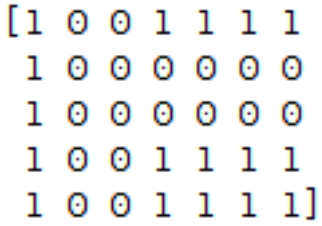



Abbildung 6: Strukturelement 2 - Horizontales Fenster - links

Nun stehen 4 Matrizen zur Verfügung mit den Treffern der Patterns. Mit einer Schleife wird nun über das gesamte Bild iteriert und Verbindungen zwischen den Fensterteilen gebildet, um falsche Treffer zu eliminieren. Bei vorhandenen Pixel im Ergebnis von der linken Seite des Fensters wird ab dem Pixel so lange nach rechts weitere Pixel abgefragt bis entweder im Orignalbild kein weißer Pixel mehr gefunden wird (keine Verbindung vorhanden), oder ein Pixel im rechten Gegenstück gefunden wird. Gibt es eine solche Verbindung wird noch zusätzlich überprüft, ob diese Distanz mindestens das doppelte der durchschnittlichen Wandbreite lang ist. Eine weitere Überprüfung wird noch durchgeführt bei allen Verbindungspixeln, und zwar ob die Nachbarpixel 2 Pixel weiter oben und 2 weiter unten schwarz sind, da normalerweise eine Strichdicke von 1-3 Pixel bei Fenstern besteht. Die selben Überprüfungen werden jeweils auch rotiert für vertikale Fenster mit den anderen beiden Ergebnis-Matrizen durchgeführt.

## 4.6 Stiegenerkennung

Bei der Findung der Stiegen ergaben sich zwei Grundprobleme:

1. Stiegen haben keine einheitliche Form, sie bestehen im Wesentlichen aus dünnen Linien, auch die Winkel sind nur relativ zu den Rändern einigermaßen einheitlich. Die Findung der Umrandung vom Stiegen wiederum erscheint unmöglich, da hierbei fast die gesamte Palette an Strukturen möglich ist (dicke Wände, dünne Wände, keine Wand,...).

2. Sämtliche Einzelteile (also Linien) kommen in verschiedensten Kombinationen auch abseits der Stiegen in großer Menge vor, auch die Einzelteile lassen sich daher schwer zuordnen.

Meine ersten Schritte zur Findung der Stiegen waren die Umwandlung in Graustufen, die Entfernung von grauen Bereichen mittels Schwellenwertoperation und die Entfernung des Bereichs außerhalb der zu betrachtenden Fläche. Dafür habe ich die am weitesten außen gelegenen schwarzen Pixel gesucht und damit eine Bounding Box geschaffen. Der nächste Schritt war die Entfernung dicker Linien (also Wände, Schrift, etc.), da Stiegen nur aus dünnen Linien bestehen. Im resultierenden Bild wurden die verbliebenen dünnen Linien durch Dilatation des weißen Bereichs zwischen diesen Linien klarer voneinander getrennt. Hauptzweck der klaren Auftrennung in dünne Linien wäre die darauf folgende Anwendung der Hough-Transformation für gerade Linien gewesen.
Aufgrund der vielen Linien verschiedenster Längen und Richtungen ist es mir allerdings nicht gelungen, eine Wahl der Buckets und Grenzwerte zu finden, die es mir ermöglicht hätte, zu Treppen gehörige gerade Linien von solchen zu trennen, die zu anderen Strukturen gehören, weshalb ich diesen Ansatz letztlich aufgeben musste.
Mein nächster Ansatz war ein rekursiver Algorithmus, der auf Basis von Connected Components gerade Linien finden sollte. Auch hier ist es mir nicht gelungen ein zufriedenstellendes Ergebnis zu erzielen, da die Linien dafür nicht gleichförmig genug waren - vor allem die schrägen Linien stellten hier eine zu hohe Herausforderung dar.

Letztlich hab ich mich dann dazu entschlossen, die Bereiche mit starker Dichte dünner Linien zu finden und als Kandidaten für Stiegen zu bestimmen. Hierfür habe ich Dilatation des schwarzen Bereichs auf das Bild angewendet. Um große Cluster zu erhalten, habe ich dafür eine 17x17-Umgebung gewählt.
Als letzter Schritt erfolgt Anwendung einer Threshold-Operation auf in ihrer Größe vom Maßstab (also von der in wall_thiccness bestimmten Wanddicke) abhängige Nachbarschaftscluster: Nur dort, wo in einem großen Bereich fast ausschließlich schwarze Pixel vorhanden sind, wird eine Stiege erkannt. Der Rückgabewert ist 1 für 'Stiegen vorhanden' und 0 für 'keine Stiegen vorhanden'.

# 5 Evaluierung

Beim Testen des Datensets mit den 20 Bildern konnten mehrere Schlüsse gezogen werden. Erstens hatten sowohl die Fenstererkennung, als auch die Tür- beziehungsweise Raumerkennung eine durchschnittliche Abweichung von höchstens 0.5. Bei der Fenstererkennung hatten wir eine absolute Abweichung von 0.4, bei der Türerkennung von 0.5 und bei der Raumerkennung von 0.4. Wir haben hier auf die absolute statt der prozentuellen Abweichung gesetzt, da die prozentuelle Abweichung in unseren Augen eine falsche Sicht der Akkuratheit liefert. So wären Fehler bei hohen Anzahlen weniger gewichtet als bei niedrigen.

Es gibt aber auch Bilder, bei welchen sowohl eine korrekte Anzahl der Fenster, Türen und Räume möglich war, siehe zum Beispiel Abbildung 7.

Die Überprüfung der korrekten Flächen ist abhängig von der Anzahl der Räume. Die An-

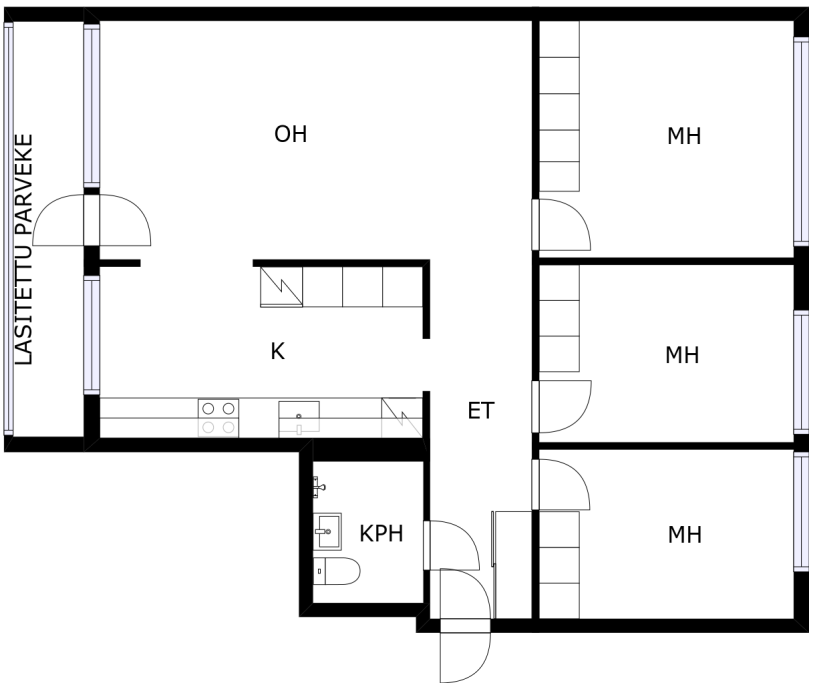



Abbildung 7: Datei09 in unserem Datenset

zahl wiederum ist abhängig von der Türerkennung. Wenn nämlich Türen fälschlicherweise erkannt werden, dann ist folglicherweise die Raumerkennung auch falsch. Dies muss allerdings nicht zwangsweise die Ausgabe der Proportionen verändern. In manchen Fällen werden aufgrund hartnäckiger Artefakte Türen gezeichnet, welche dann ganz kleine Räume bilden. Diese haben dann einen Flächenanteil von 0% und sind daher zu vernachlässigen. Die tatsächliche Fläche zu erkennen erwies sich als unmöglich, da leider weder ein Maßstab angegeben war und zum anderem dieser offensichtlich sehr unterschiedlich war.
Bei der Auswertung des Datensatz sind wir häufig auf die zu erwartenden Werte gekommen. Bei der Fensterauswertung konnte das Programm sechsmal, bei der Türerkennung fünfmal und bei der Raumerkennung zehnmal die korrekte Anzahl auslesen. Die Anzahl der Räume stimmt immer dann, wenn auch die Anzahl der Türen stimmt. Allerdings gibt es auch Fälle, wo es zu viele Türen gibt, aber die Anzahl der Räume stimmt. Das passiert immer dann, wenn durch die überflüssigen Türen kein neuer Raum fälschlicherweise gebildet wird.

Die Stiegenerkennung funktioniert schlecht bis gar nicht. Zwar lassen sich bei richtiger Einstellung der Parameter viele Stiegen erkennen, allerdings ist die Erkennung stark abhängig von den Abständen zwischen den Linien, die die Stufen markieren, und der Kräftigkeit der Striche. Hinzu kommt, dass die durchschnittliche Wanddicke als Referenzfaktor für den Maßstab eher schlecht als recht funktioniert. Einige False Positives ergeben sich durch in der Karte eingezeichnete Strukturen, vor allem durch Herde.

## 6 Schlusswort

Unser Projekt funktioniert in einem Großteil der Fälle sehr akkurat, vereinzelt kann es zu Ausreißern kommen. In den meisten Fällen hat ein Fehler in einer Methode eine Auswirkung auf eine andere Methode. Eine große Schwierigkeit war unter anderem die Ermittlung der durchschnittlichen Wanddicke. Diese ist nämlich auch ein wichtiger Teil der Türerkennung und hat dadurch auch einen essentiellen Einfluss auf die Raumerkennung.

Allerdings funktioniert das Programm, wenn es nicht durch hartnäckige Artefakte oder eine stark schwankende Wanddicke beeinflusst wird, einwandfrei.

## Literatur

[1] Wilhelm Burger and Mark James Burge. *Digitale Bildverarbeitung.* Springer, 2005, 2006.

[2] G Shapiro, L.; Stockman. *Computer Vision.* Prentice Hall, 2002.