GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

XX XXX XX
YYY YYY Y

Bachelorarbeit
im Studiengang „Angewandte Informatik“

SUSAN

Ein Ansatz zur Strukturerkennung in Bildern

Julian Lüken
julian.lueken@stud.uni-goettingen.de

Institut für Numerische und Angewandte
Mathematik

Bachelor und Masterarbeiten des Zentrums für
angewandte Informatik an der
Georg-August-Universität Göttingen

27. Oktober 2019

Inhaltsverzeichnis

Kapitel 1

Einführung

Kantendetektoren sind ein wichtiges Werkzeug der Bildverarbeitung...

Kapitel 2

Mathematische Grundlagen

2.1 Analytische Grundlagen

2.2 Varianzanalyse

2.3 Bildverarbeitung

In der digitalen Bildverarbeitung für Graustufenbilder betrachten wir Abbildungen der Form

$$I : [0, B-1]_{\mathbb{Z}} \times [0, H-1]_{\mathbb{Z}} \to [0, 255]_{\mathbb{Z}}.$$

Solche Abbildungen werden im weiteren Verlauf dieser Arbeit schlichtweg als Bilder bezeichnet. H steht für die Höhe und B für die Breite des Bildes. Ferner steht 0 für schwarz und 255 für weiß.

Eines der wichtigen Instrumente der Bildverarbeitung ist die Faltung. Da die Faltung allerdings nur für Funktionen definiert wird und die Abbildung I keine Funktion ist, müssen wir die Faltung für den diskreten Fall neu definieren. Diese Definition trägt den Namen Filtermaske. Eine Filtermaske ist eine Matrix $k \in \mathbb{R}^{n \times m}$, die folgendermaßen Anwendung findet:

$$O(x, y) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} k(i, j) \cdot I(x - i + a, y - j + b),$$

wobei (a, b) der Nucleus der Filtermaske ist, I das Eingangsbild und O das neu gewonnene Ausgangsbild. Wir schreiben, wie bei der Faltung,

$$O = k * I$$

Ein Beispiel für eine solche Filtermaske bietet der sogenannte Gaußsche Weichzeichner, der in [3] näher beschrieben wird. Im stetigen Fall würde man ihn anwenden, indem man eine Funktion mit der Funktion der Normalverteilung in zwei Dimensionen faltet. Die

Funktion der Normalverteilung lautet

$$g(x,y) := \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

Im Fall von digitalen Bildern kennen wir nur diskrete Bildpunkte, also muss eine Approximation genügen. Man wähle eine Größe $n = m$ der Filtermaske und summiere für $x, y \in [-n, n]_{\mathbb{Z}}$ jeweils

$$G(x,y) = \frac{1}{N} \sum_{i=\lceil \frac{N-1}{2} \rceil}^{\lfloor \frac{N-1}{2} \rfloor} \sum_{j=\lceil \frac{N-1}{2} \rceil}^{\lfloor \frac{N-1}{2} \rfloor} g(x + \frac{i}{N}, y + \frac{j}{N})$$

für ein passendes σ und ein großes N. Die Größe der Filtermaske bestimmt sich in diesem Fall über das gewählte σ. Eine Faustregel dafür lautet, dass falls $\sqrt{\hat{x}^2 + \hat{y}^2} \geq 3\sigma$ gilt, auch $g(\hat{x}, \hat{y}) \approx 0$ gilt. Ausgehend davon können wir $n = \lfloor 3\sigma \rfloor - 1$ wählen und erhalten somit die Filtermaske

$$G = \frac{1}{273} \begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}.$$

für $\sigma = 1$, $N = 1000$ und $n = 2$. Wenden wir nun unseren Operator G auf folgendes linkes Bild an, so erhalten wir das nachstehende rechte Bild:

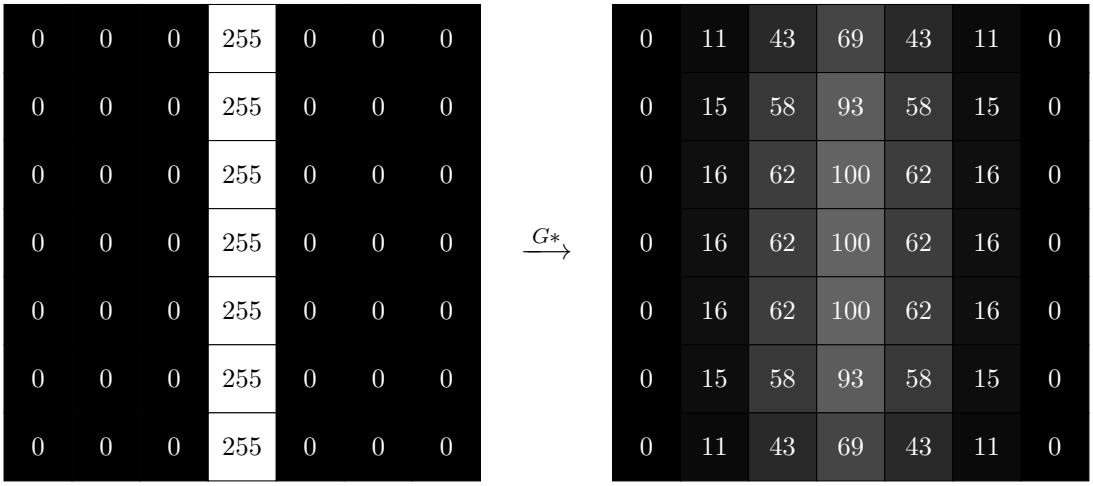

Abbildung 2.1: Die berechnete Gauß-Filtermaske in Aktion.

Kapitel 3

Der SUSAN Kantendetektor

In diesem Kapitel stellen wir den Algorithmus vor, danach erläutern wir einige Details zur Implementation des Algorithmus in Python 3. Im dritten Abschnitt leiten wir her, warum der SUSAN-Kantendetektor funktioniert und im letzten Teil führen wir ein paar numerische Experimente durch.

3.1 Der Algorithmus

Der SUSAN Kantendetektor aus [1] besteht aus drei Großschritten:

Im ersten Schritt, den wir in Abschnitt 3.1.1 besprechen, legen wir eine Maske über jedes Pixel. Mithilfe dieser Maske berechnen wir für jedes Pixel eine Kennzahl A, die im weiteren Verlauf dieser Arbeit „Antwort“ heißt. Die lokalen Maxima von A liegen genau auf den lokalen Extrema der partiellen Ableitungen in horizontaler und vertikaler Richtung unseres Eingangsbildes I, wie wir später noch zeigen.

Der zweite Schritt in Abschnitt 3.1.2 ist die sogenannte Non-Maximum-Suppression, bei der wir mithilfe der Richtung der im ersten Schritt berechneten Antwort die Kanten im Eingangsbild I noch genauer lokalisieren, indem wir nur Züge der Kanten erhalten, die in der Antwort lokal maximal sind.

Falsch positive und falsch negative Ergebnisse sind auf mit Rauschen behafteten Digitalbildern keine Seltenheit. Aus diesem Grund gibt es noch einen dritten Ausdünnungsschritt, in welchem bei Bedarf räumlich isolierte Antworten entfernt werden und Kanten vervollständigt werden. Diesen Schritt behandeln wir näher im Abschnitt 3.1.3.

Anschließend wird im letzten Teil dieses Kapitels, Abschnitt 3.1.4 der Eckendetektor des SUSAN-Prinzips vorgestellt. Durch kleine Änderungen an den ersten zwei Schritten können so, zusätzlich zu Kanten, ebenfalls Ecken gefunden werden.

3.1.1 Das SUSAN-Prinzip

Sei I ein Eingangsbild. Um jedes Pixel im Bild legen wir eine Maske. Für unseren Zweck betrachten wir lediglich die Masken der Größe 3×3 beziehungsweise 7×7, wie sie in Abbildung 3.1 dargstellt sind.

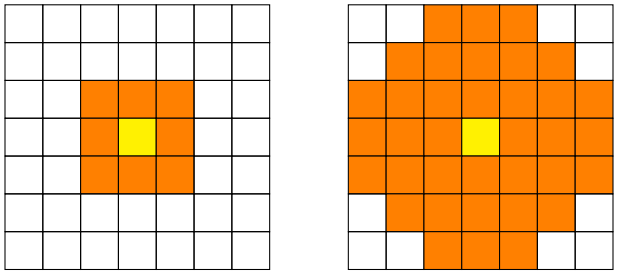

Abbildung 3.1: Die zwei Filtermasken. Links 3×3, rechts 7×7. Dabei ist das gelbe Pixel der Mittelpunkt oder Nucleus der Maske, die orangenen Pixel liegen in der Maske und die weißen Pixel außerhalb der Maske.

Die Masken sind Umgebungen eines Pixels, die einen Kreis approximieren. Die 3×3 Maske approximiert einen Kreis mit einem Radius von 1.4, die 7×7 Maske hingegen approximiert einen kreis mit einem Radius von 3.4.

Prinzipiell finden wir beim ersten Schritt des SUSAN-Verfahrens die Anzahl der zum Nucleus, also dem Mittelpunkt der Maske, ähnlicher Pixel $n(r)$ für jeden Nucleus r. Mit „ähnlich“ ist in diesem Sinne gemeint, dass die absolute Intensitätsdifferenz $|I(a) - I(b)|$ zwischen den beiden Pixeln a und b unter einer im Vorfeld festgelegten Grenze t liegt. Die ähnlichen Pixel bezeichnen wir ferner als USAN (*univalue segment assimilating nucleus*). Die maximale Größe einer USAN ist für die 7×7 Maske 36 und für die 3×3 Maske 8. Üblicherweise wählen wir g als $\frac{3}{4}$ der Maskengröße, in den ersten Beispielen werden wir allerdings g gleich der Maskengröße wählen.

Konkret führen wir folgende Rechnung durch. Für jedes Pixel r_0 in I, wobei $I(r_0)$ der Grauwert am Pixel r_0 ist, berechnen wir die Antwort

$$A(r_0) = \max\{0, g - n(r_0)\}.$$

Dabei ist n definiert als

$$n(r_0) = \sum_r c_t(r, r_0),$$

wobei wir über alle Pixel r in der Maske summieren und

$$c_t(r, r_0) = \exp\Big(- \Big(\frac{I(r) - I(r_0)}{t} \Big)^6 \Big)$$

eine Vergleichsfunktion für zwei Pixel ist. Statt der obigen Vergleichsfunktion kann auch

die Näherung

$$c_t(r, r_0) \approx \begin{cases} 1 & \text{falls } |I(r) - I(r_0)| \leq t \\ 0 & \text{sonst} \end{cases}$$

verwendet werden. In einer USAN liegen genau diejenigen Pixel r aus der Maske, für die $c_t(r, r_0) > 0$. Wir verbleiben mit der Antwort A, auf welchem wir schon sehr gut den Effekt der Kantendetektion beobachten können.

255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255
255	255	255	255	255	0	255	255	255	255	255

170	85	85	85	170	255	170	85	85	85	170
85	0	0	0	85	212	85	0	0	0	85
85	0	0	0	85	212	85	0	0	0	85
85	0	0	0	85	212	85	0	0	0	85
85	0	0	0	85	212	85	0	0	0	85
85	0	0	0	85	212	85	0	0	0	85
85	0	0	0	85	212	85	0	0	0	85
85	0	0	0	85	212	85	0	0	0	85
85	0	0	0	85	212	85	0	0	0	85
85	0	0	0	85	212	85	0	0	0	85
170	85	85	85	170	255	170	85	85	85	170

Abbildung 3.2: Das SUSAN Prinzip (links Eingang, rechts Antwort, $t = 1$)

Nehmen wir nun zum Beispiel ein Eingangsbild wie in Abbildung 3.2. Hier wurde lediglich das SUSAN-Prinzip angewandt und danach wurde jeder Graustufenwert mit dem Faktor $\frac{255}{\max\{A(i,j)\}}$ multipliziert, um den Effekt hervorzuheben. Zunächst einmal finden wir, dass $A(i, j)$ genau da am größten ist, wo unser Eingangsbild eine Kante hat. Auch an den Rändern des Bilds findet das SUSAN-Prinzip eine Kante. Es besteht lediglich das Problem, dass um die Kante herum eine kleine Antwort dort ist, wo im originalen Bild nur die Nähe zu einer Kante besteht. Wir schließen daraus, dass die Lokalisation der Kanten zwar durchaus funktioniert, aber verbesserungswürdig ist.

255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255
255	255	255	0	0	0	0	0	255	255	255
255	255	255	0	0	0	0	0	255	255	255
255	255	255	0	0	0	0	0	255	255	255
255	255	255	0	0	0	0	0	255	255	255
255	255	255	0	0	0	0	0	255	255	255
255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255

255	127	127	127	127	127	127	127	127	127	255
127	0	0	0	0	0	0	0	0	0	127
127	0	0	63	127	127	127	63	0	0	127
127	0	63	255	127	127	127	255	63	0	127
127	0	127	127	0	0	0	127	127	0	127
127	0	127	127	0	0	0	127	127	0	127
127	0	127	127	0	0	0	127	127	0	127
127	0	63	255	127	127	127	255	63	0	127
127	0	0	63	127	127	127	63	0	0	127
127	0	0	0	0	0	0	0	0	0	127
255	127	127	127	127	127	127	127	127	127	255

Abbildung 3.3: Das SUSAN Prinzip findet Ecken und hebt diese sogar in der Antwort hervor

Im Beispiel auf Abbildung 3.3 können wir beobachten, dass das SUSAN-Prinzip Ecken besonders hervorhebt. Dieser Effekt spielt eine große Rolle im SUSAN-Eckendetektor, auf den im Abschnitt 3.1.4 dieser Arbeit weiter eingegangen wird.

Außer auf Testbildern ist es laut [1] vorteilhaft, g so groß wie $\frac{3}{4}$ der Maskengröße zu wählen, um die Lokalisation der Kanten durch das Verfahren zu verbessern. Die Wirksamkeit dieser Maßnahme ist beispielhaft an Abbildung 3.4 sichtbar.

Abbildung 3.4: Ein Testdurchlauf mit $t = 15$ und der 7×7 Maske. Ganz links ist das Originalbild. In der Mitte ist g die Maskengröße und rechts $\frac{3}{4}$ der Maskengröße.

Da dieser Algorithmus von den lokalen Bedingungen jedes Pixels abhängt, kann dieser Algorithmus nicht durch eine Faltung ausgedrückt werden, wie etwa eine Reihe anderer Filter, darunter der Gaussche Filter aus 2.3, aber auch der Kantendetektor von Canny (siehe [x]).

3.1.2 Non-Maximum-Suppression

Das oben genannte Prinzip aus Abschnitt 3.1.1 findet Kanten innerhalb von Bildern, aber die Lokalisation der Kanten könnte besser sein, wie wir in der Abbildung 3.2 bereits erkennen konnten. Im Bereich der eigentlichen Kante stellen wir fest, dass die Antwort A ungleich 0 ist. Um die Kante genauer zu lokalisieren, verwenden wir das Prinzip der Non-Maximum-Suppression, bei der nur die maximale Antwort entlang einer Kante erhalten bleibt. Zu diesem Zweck berechnen wir die Richtung eines jeden Pixels $r_0 = (x_0, y_0)$, für welches $A(x_0, y_0) \neq 0$ gilt, durch eine Fallunterscheidung. Wir unterscheiden dabei verschiedene Arten von Kanten.

Der Inter-Pixel-Fall liegt vor, wenn eine Kante zwischen zwei Pixeln liegt. Der Intra-Pixel-Fall liegt vor, wenn ein Pixel genau auf der Kante liegt. In den Abbildungen stehen die grün markierten Pixel jeweils für Pixel, die in der USAN liegen. Die roten Pixel liegen außerhalb der USAN. Die Mitte ist der Nucleus der Maske.

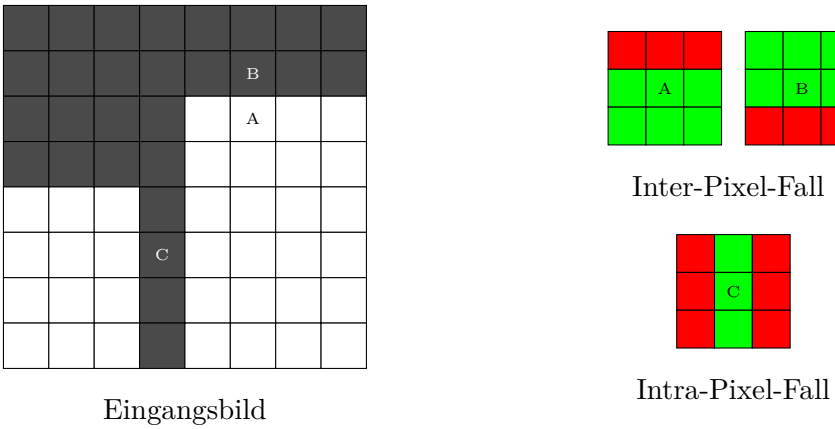

Abbildung 3.5: Inter-Pixel-Fall und Intra-Pixel-Fall für die 3×3 Maske: Die drei rechts abgebildeten USAN wurden durch wählen des Nucleus an den mit A, B und C markierten Stellen im Eingangsbild ausgewertet. Die grün markierten Pixel stehen für Teile der USAN.

Wir werden zunächst jede Antwort in einen der zwei Fälle einsortieren, danach können wir die Richtung bestimmen.

1. **Inter-Pixel:**

   Falls die Größe der USAN größer ist als der Maskendurchmesser und die Distanz zwischen $\mathrm{COG}(r_0)$ und r_0 größer als 1 Pixel ist, so ist die Richtung $D(r_0)$ gegeben durch

$$D(r_0) = \begin{cases} \arctan\left(\frac{x_0 - \mathrm{COG}(x_0)}{y_0 - \mathrm{COG}(y_0)}\right) & \text{falls } \mathrm{COG}(y_0) \neq y_0 \\ \frac{\pi}{2} & \text{sonst} \end{cases},$$

wobei

$$\mathrm{COG}(r_0) := \frac{\sum_r r\, c(r, r_0)}{\sum_r c(r, r_0)} = \frac{\sum_r r\, c(r, r_0)}{n(r_0)}$$

das sogenannte Gravitationszentrum (*center of gravity*) ist.

2. **Intra-Pixel:**

   Andernfalls müssen wir die zweiten Momente der USAN folgendermaßen berechnen:

$$d_{x_0} := \sum_r (x - x_0)^2\, c_t(r, r_0)$$
$$d_{y_0} := \sum_r (y - y_0)^2\, c_t(r, r_0)$$
$$\sigma := -\mathrm{sgn}\Big(\sum_r (x - x_0)\,(y - y_0)\, c_t(r, r_0) \Big)$$

   Dabei ergibt sich die Kantenrichtung als

$$D(r_0) = \begin{cases} \sigma \arctan \frac{d_{y_0}}{d_{x_0}} & \text{falls } d_{x_0} \neq 0 \\ \frac{\pi}{2} & \text{sonst} \end{cases}$$

   Falls allerdings $d_{x_0} = 0$, so ist $D(r_0) = \frac{\pi}{2}$

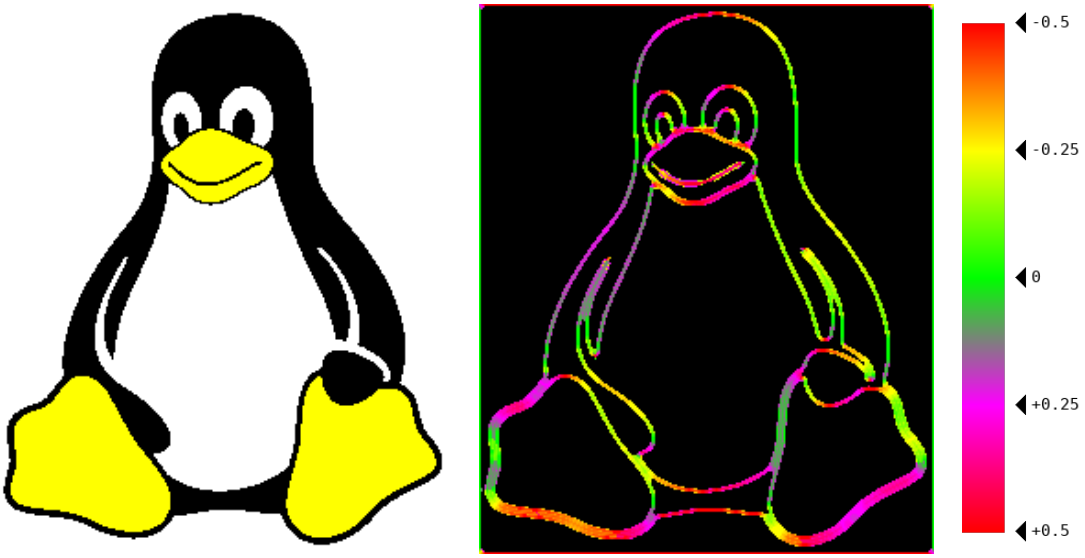

Abbildung 3.6: Ein Testdurchlauf für das Berechnen der Richtungen mit $t = 15$. Links ist das Eingangsbild. Rechts ist die Kantenrichtung für jedes Pixel abgebildet. Die Zahlenwerte können an der Skala abgelesen werden und sind als Vielfache von π angegeben.

Die Berechnungen für die jeweiligen Kantenrichtungen an jedem Pixel werden in der Implementation gleichzeitig mit den Berechnungen für das SUSAN-Prinzip durchgeführt. Auf diese Weise müssen c_t und n an jeder Stelle nur einmal berechnet werden.

147	40	40	40	147	255	147	40	40	40	147
40	0	0	0	0	201	0	0	0	0	40
40	0	0	0	0	201	0	0	0	0	40
40	0	0	0	0	201	0	0	0	0	40
40	0	0	0	0	201	0	0	0	0	40
40	0	0	0	0	201	0	0	0	0	40
40	0	0	0	0	201	0	0	0	0	40
40	0	0	0	0	201	0	0	0	0	40
40	0	0	0	0	201	0	0	0	0	40
40	0	0	0	0	201	0	0	0	0	40
147	40	40	40	147	255	147	40	40	40	147

Abbildung 3.7: Eingangsbild wie in Abbildung 3.2. Durch die Non-Maximum-Suppression wurde die Kante pixelgenau lokalisiert

Die Richtung lohnt sich nur für diejenigen Pixel (i, j) zu bestimmen, für die $A(i, j) > 0$ gilt. Ist die Richtung der Kante bestimmt, so können wir die lokalen Maxima von A entlang der Richtung, die senkrecht zur Kantenrichtung steht, erhalten. Alles, was kein lokales Maximum entlang dieser Richtung ist, wird verworfen. Wir erhalten so ein neues Bild. In Kapitel 3.2 wird darauf eingegangen, wie genau dieser Prozess implementiert wurde.

Die Non-Maximum-Suppression unterdrückt in unserem Beispiel tatsächlich alle diejenigen Pixel, die keine lokalen Maxima in der Richtung senkrecht zur Kante sind. Auch für größere Bilder erzielt die Non-Maximum-Suppression den erwünschten Effekt, siehe Abbildung 3.8.

Abbildung 3.8: Wie Abbildung 3.4. Links das Bild mit den Kantenrichtungen (Skala wie in Abbildung 3.6). In der Mitte das Bild nach dem prinzipiellen SUSAN-Schritt. Das Bild rechts ist nach der Non-Maximum-Suppression.

3.1.3 Ausdünnen

Da viele digitale Bilder eingangs mit Rauschen behaftet sind, ist es manchmal hilfreich, einzelne Antworten zu entfernen und dafür andere hinzuzufügen. Zu diesem Zwecke empfiehlt [1], dass man einige der Kanten ausdünnt. Dieser Vorgang wird genauer in [2] beschrieben. Es wird erneut über das ganze Bild iteriert. Jedes Pixel (i, j) mit einer Antwort $A(i, j) > 0$ wird auf die Anzahl seiner direkten Nachbarn mit $A(x, y) > 0$ überprüft. Als direkte Nachbarschaft werden die acht nächsten Pixel bezeichnet, siehe dazu die 3×3 Maske in Abbildung 3.1. Angenommen, das Pixel hat...

- **0 Nachbarn:** Entferne die Antwort des Pixels.
- **1 Nachbar:** Überprüfe, ob in einer Reichweite von 3 Pixeln eine Linie mit der gleichen Richtung existiert. Falls ja, verbinde die Pixel miteinander.
- **2 Nachbarn:** Falls das Pixel adjazent zu einer diagonalen Linie ist, entferne es. Falls das Pixel außerhalb einer sonst horizontalen oder vertikalen Linie liegt, verschiebe die Antwort des Pixels in die Lücke der vertikalen oder horizontalen Linie
- **3 Nachbarn:** Falls die drei Nachbarn in einer Linie liegen, entferne die Antwort des Pixels.

Im Testbild der Abbildung 3.9 erkennt man die Funktionsweise in allen vier oben genannten Fällen.

Abbildung 3.10: test

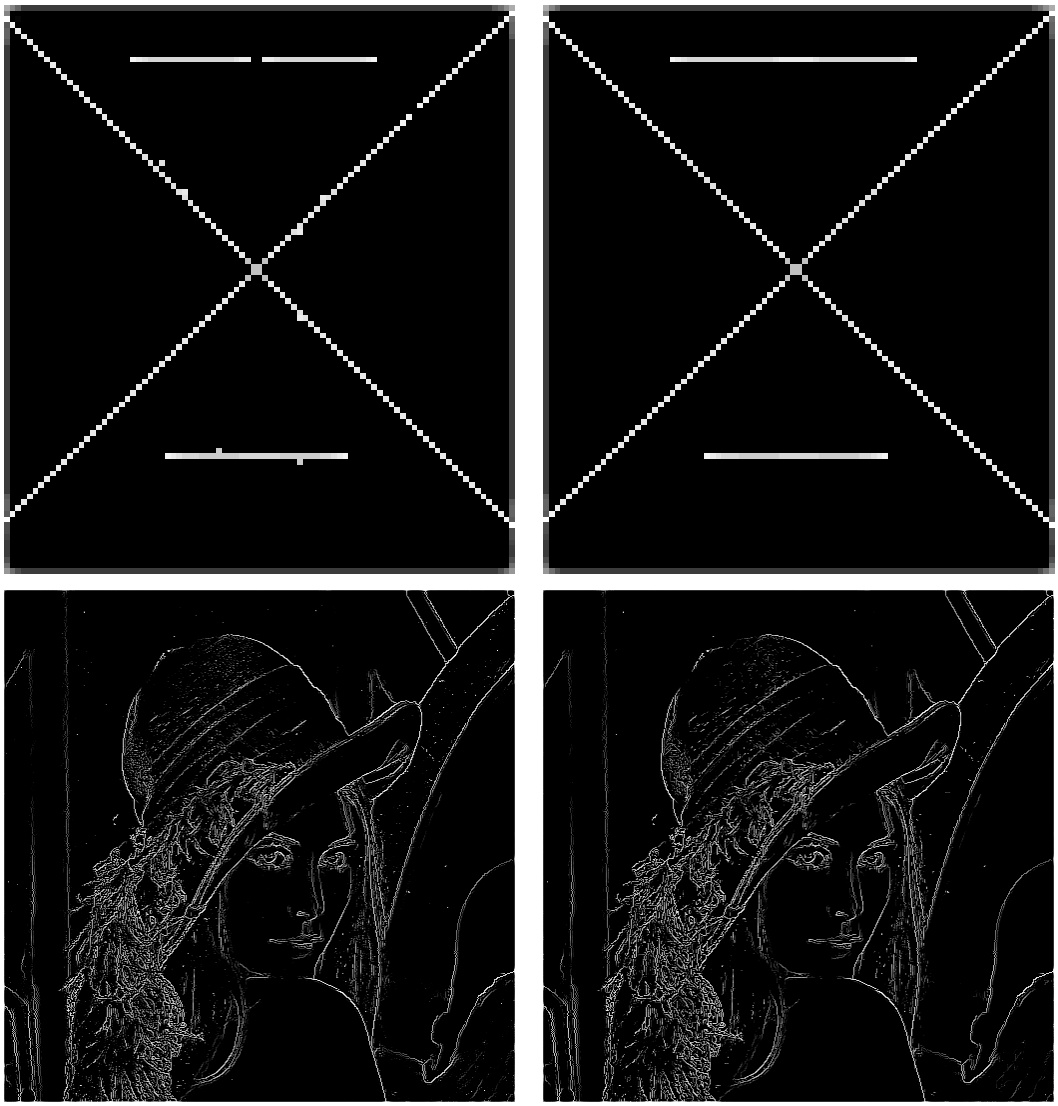

Abbildung 3.9: Links: Kantenbilder nach der Non-Maximum-Suppression. Rechts: Kantenbilder nach dem Ausdünnen.

Laut der zitierten Quelle [2] soll eigentlich auch für die Fälle mit mehr als 3 Nachbarn gesorgt werden. Allerdings ist eine Vorhersage über Nachbarschaften mit 4 oder mehr Nachbarn relativ schwierig. In Abbildung 3.10 sind ein paar Beispielhafte Fälle abgebildet.

3.1.4 Der SUSAN-Eckendetektor

In [1] wird zusätzlich ein Verfahren beschrieben, mit welchem sich durch das SUSAN-Prinzip Ecken finden lassen. Dieser Eckendetektor basiert darauf, dass der SUSAN Kantendetektor an Ecken stärkere Antworten liefert als an Kanten. Man verwendet also im

ersten Schritt des Eckendetektors lediglich einen niederigeren Wert für g in 3.1. Danach finden sich immer noch viele falsch positive Antworten. Um diese zu vermeiden, überprüft man, ob die jeweiligen Kandidaten mit einer Kante zusammenhängen. Dies gelingt durch das Erzwingen folgender Regeln für jede Ecke:

1. Die Entfernung zwischen Gravitationszentrum und Nucleus muss groß genug sein
2. Jedes Pixel in der Maske, welches sich auf gerader Strecke zwischen Nucleus und Gravitationszentrum befindet, muss in der USAN des Nucleus enthalten sein.
3. Alles, was kein lokales Maximum in einer 5×5 Maske ist, soll entfernt werden.

Die Entfernung zwischen Gravitationszentrum und Nucleus versichert, dass die Richtung der Kante sich an der Ecke ändert. Punkt garantiert den räumlichen Zusammenhang von Ecken und Kanten. Der dritte Punkt ist vom Prinzip gleich der Non-Maximum-Suppression.

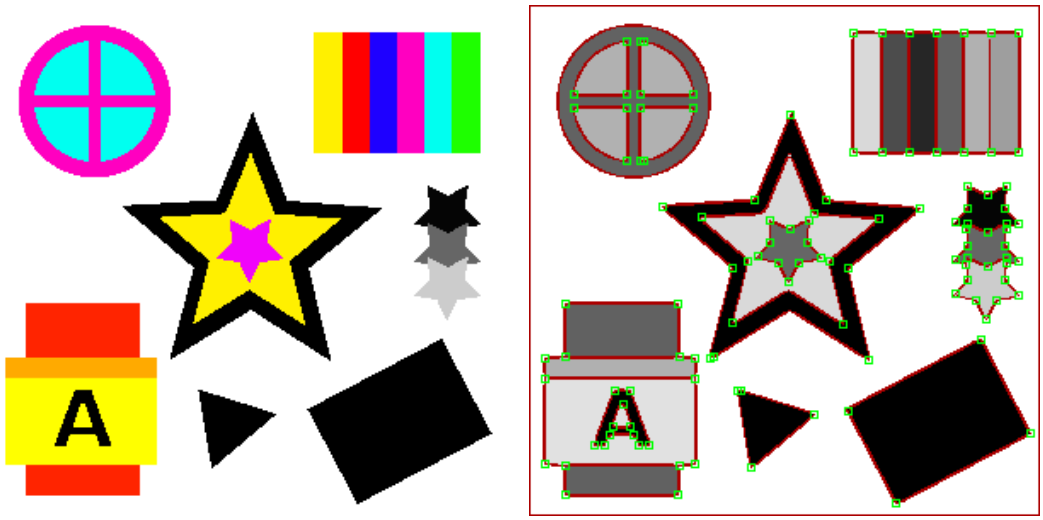

Abbildung 3.11: Die Ecken sind grün umrahmt wohingegen die Kanten rot übergezeichnet sind.

3.2 Implementation

Meine persönliche Implementation ist in Python 3 erfolgt. Die Software ist abhängig von den folgenden Python 3 Paketen:

- `pillow`, eine Bibiliothek für Bildverarbeitung [4]
- `numpy`, eine Bibliothek für wissenschaftliches Rechnen [5]
- `multiprocessing` eine Bibiliothek für prozessbasierte Parallelisierung [6]

Die Software ist verfügbar unter dem Link `https://gitlab.gwdg.de/julian.lueken/susan/`. In der Klasse `Susan.py` befindet sich der vollständige Kantendetektor. Er lässt sich per `import Susan` nutzen. Dazu erzeugt man ein Objekt der `Susan`-Klasse und übergibt eine Bilddatei. Danach ruft man auf dem Objekt die Funktion `detect_edges_mp` mit dem Parameter t, dem Grenzwert aus Abschnitt 3.1.1, auf.

```
import Susan

t = 15
S = Susan("lena.png")
S.detect_edges_mp(t)
```

Abbildung 3.12: Nutzen der Python 3 Applikation

Es können weiterhin folgende Parameter übergeben werden:

Name	**Beschreibung**	**Datentyp**	**Standardwert**
`t`	Grenzwert für c_t	Ganzzahl, $t \in [1, 255]_{\mathbb{N}}$	-
`filename`	Ausgabepfad	Zeichenkette	`"out.png"`
`nms`	Non-Maximum-Suppression	Boolesch	`True`
`heatmap`	Ausgabe der Richtungen	Boolesch	`False`
`geometric`	Schwache Antworten unterdrücken	Boolesch	`True`
`corners`	Ecken finden	Boolesch	`False`
`overlay`	Überlagerungsbild erzeugen	Boolesch	`True`

Beim Paramater `heatmap` werden die Richtungen als Bild ausgegeben, dabei korrespondieren die Richtungen zu Farben, genau so wie wir es schon in den Abbildungen zu Abschnitt 3.1.2 sehen konnten. Falls `geometric` auf `True` gesetzt ist, so wird g als $\frac{3}{4}$ der Maskengröße gewählt. Andernfalls ist g gleich der Maskengröße, wie im Abschnitt 3.1.1. Die Option `overlay` erzeugt ein Bild, auf dem Kanten in rot nachgezeichnet und Ecken von grünen Kästen umrahmt werden, wie in Abbildung 3.11.

3.2.1 Vergleichsfunktion

Zu Gunsten der Effizienz habe ich für die Vergleichsfunktion eine Lookup-Tabelle implementiert, wie es auch in [quelle] nahgelegt wurde. Beim Initialisieren des `Susan`-Objekts wird ein Feld erzeugt. Das Ziel ist es, alle möglichen Werte, die c_t (siehe 3.1.1) annehmen kann, zu speichern. In unserem Fall ist c_t folgendermaßen definiert:

$$c_t(a,b) := \exp\left(-\left(\frac{I(a)-I(b)}{t}\right)^6\right)$$

Ein Pixel in einem 8-bit Graustufenbild kann $2^8 = 256$ mögliche Intensitäten annehmen, demnach braucht das Feld für die Differenz zweier solcher Graustufenintensitäten 512 Plätze. Regulär werden Felder entweder ab 0 oder 1 indiziert. In Python werden Felder ab 0 indiziert, allerdings bietet Python den Vorteil, dass man mit Index $-i$ das i-te Element von hinten abrufen kann. In der Implementation ist diese Lookup-Tabelle dank dieser speziellen Art der Indizierung einfach und elegant: Der Index $a-b$ des Feldes steht für $c_t(a,b)$.

3.2.2 Non-Maximum-Suppression

Die Non-Maximum-Suppression ist ein Vorgang, bei der jede Kante genauer lokalisiert wird. Entlang einer Kante soll immer nur die maximale Antwort erhalten bleiben. Um die Non-Maximum-Suppression durchzuführen, wird zunächst für jedes Pixel (i,j) mit $A(i,j) > 0$ die Kantenrichtung $D(i,j)$ bestimmt. Die möglichen Kantenrichtungen werden dann für einen Zwischenschritt kategorisiert. Die Kategorien sind *negativ diagonal*, *vertikal*, *positiv diagonal* und *horizontal*. Die Kategorie bestimmt, welche zwei adjazenten Pixel C für die Non-Maximum-Suppression interessant sind.

Bedingung	**Kategorie**	**Adjazente** C
$D(i,j) \leq -\frac{3}{8}\pi$	negativ diagonal	$\{(i+1,j-1), \quad (i-1,j+1)\}$
$D(i,j) > -\frac{3}{8}\pi, \quad D(i,j) \leq -\frac{1}{8}\pi$	horizontal	$\{(i-1,j), \quad (i+1,j)\}$
$D(i,j) > -\frac{1}{8}\pi, \quad D(i,j) \leq \frac{1}{8}\pi$	positiv diagonal	$\{(i+1,j+1), \quad (i-1,j-1)\}$
$D(i,j) > -\frac{3}{8}\pi$	vertikal	$\{(i,j-1), \quad (i,j+1)\}$

Für jedes Pixel (i,j) wird nun überprüft, ob $(i,j) = \max(C \cup \{(i,j)\})$ gilt. Gilt es nicht, so wird $A(i,j)$ unterdrückt.

3.2.3 Parallelisierung

Das SUSAN-Prinzip sieht vor, die Antwort $A(i,j)$ und die Kantenrichtung $D(i,j)$ aus immer nur mithilfe von Pixeln aus der Maske auf (i,j) zu berechnen. An dieser Stelle kann die Berechnung von A und D parallelisiert werden. In meiner Implementation liefert das Paket `multiprocessing` die nötigen Werkzeuge, um die vorhandenen Ressourcen zusammenzuschließen und die Berechnung von A und D parallel abzuarbeiten.

Der Einfachheit halber habe ich das Bild nur der Höhe nach partitioniert. Eine Partitionierung $\mathcal{S}$ ist eine Menge von nichtnegativen ganzen Zahlen $\{S_1, S_2, ..., S_n, S_{n+1}\}$. Für alle $i \in \{1, 2, ..., n\}$ arbeitet der Job J_i die Partition $(S_i, S_{i+1}]_{\mathbb{Z}}$ ab.

Unter der stark vereinfachten Annahme, dass alle Kerne gleich viele Fließkommazahloperationen pro Zeiteinheit abarbeiten können und gleichgroße Partitionen etwa gleichviel Rechenzeit benötigen, gelte folgende Aussage: Eine Partitionierung $\mathcal{S}$ für die gilt $\#\mathcal{S} = n + 1$ ist genau dann optimal, wenn für alle Partitionen gilt

$$|\#(S_i, S_{i+1}]_{\mathbb{Z}} - \#(S_j, S_{j+1}]_{\mathbb{Z}}| \leq 1, \quad \forall i \neq j, \quad i, j \in \{1, 2, ..., n\}$$

Angenommen der Computer, auf dem die Software läuft, besitzt n Kerne, kann also n Jobs $J_1, J_2, ..., J_n$ gleichzeitig abarbeiten, und ein Bild der Höhe H wird eingegeben. Sei $k := H \bmod n$ und $z := \left\lfloor \frac{H}{n} \right\rfloor$. Dann ist die optimale Partitionierung $\mathcal{S}$ nach der Höhe gegeben durch:

$$\begin{aligned}
S_1 &= 0 \\
S_2 &= S_1 + z + 1 \\
S_3 &= S_2 + z + 1 \\
&\vdots \\
S_{k-1} &= S_{k-2} + z + 1 \\
S_k &= S_{k-1} + z + 1 \\
S_{k+1} &= S_k + z \\
&\vdots \\
S_n &= S_{n-1} + z \\
S_{n+1} &= S_n + z = H
\end{aligned}$$

Die Jobs $J_1, J_2, ..., J_n$ arbeiten das komplette Bild ab, denn es gilt

$$\bigcup_{i=1}^{n} (S_i, S_{i+1}]_{\mathbb{Z}} = (0, H]_{\mathbb{Z}} = [1, H]_{\mathbb{Z}}$$

3.3 Numerische Experimente

3.3.1 Variation des Grenzwerts

3.3.2 Vergleich mit dem Kantendetektor von Canny

In diesem Abschnitt wird der Kantendetektor von Canny [x] kurz vorgestellt und anschließend mit dem SUSAN-Kantendetektor verglichen. Der Kantendetektor von Canny basiert auf dem Prinzip der Ableitung: Ziel ist es, die Stellen der größten Änderung zu markie-

ren. In der stetigen Analysis sind diese Stellen die Nullstellen der zweiten Ableitung. Da wir allerdings keine stetige Funktion vorliegen haben, sondern lediglich ein digitales Bild, müssen wir eine Approximation finden.

Zunächst aber wird das Bild geglättet, meist durch einen Gausschen Weichzeichner, wie er bereits in [x] besprochen wurde.

Unter den Begriff Sobel-Operator fallen Filtermasken, die eine Approximation der ersten Ableitung in eine Richtung berechnen und orthogonal dazu das Bild glätten.

Mit den Sobel-Operatoren S_x und S_y lassen sich die geglätteten Approximationen der partiellen Ableitungen folgendermaßen berechnen

$$g_x = S_x * I \qquad g_y = S_y * I,$$

wobei

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Danach werden die beiden beiden gefalteten Bilder folgendermaßen addiert

$$O(i,j) = |g_x(i,j) + g_y(i,j)|.$$

Weiterhin wird, ähnlich wie beim SUSAN-Kantendetektor, eine Non-Maximum-Suppression durchgeführt. Die Kantenrichtung ist durch

$$\varphi(i,j) = \begin{cases} \arctan\left(\frac{g_y(i,j)}{g_x(i,j)}\right) & \text{falls } g_x(i,j) \neq 0 \\ \frac{\pi}{2} & \text{sonst} \end{cases},$$

bestimmt.

Zum Vergleich mit meiner Implementation des SUSAN-Kantendetektors ziehen wir nun die OpenCV-Implementation des Canny-Kantendetektors heran. Durch die Vektorisierbarkeit der Teilschritte lässt sich über Cannys Kantendetektor sagen, dass die Implementation in Python 3 schnellere Laufzeiten mit sich bringt. Nichtlineare Operationen müssen in Python 3, wie in den meisten anderen Programmiersprachen, über Schleifen geregelt werden. Diese sind verglichen langsam zu beispielsweise der Programmiersprache C. Das liegt begründet im dynamischen Charakters von Python [7].

Für lineare Probleme ist die Vektorisierung ein Weg, die Dynamik Pythons auszunutzen, um Ergebnisse schnell zu berechnen. Der nichtlineare Charakter des SUSAN-Detektors lässt allerdings nur einfache Parallelisierungen zu, keine Vektorisierung.

Allerdings bringt der SUSAN-Kantendetektor einen besonders großen Vorteil mit sich. Cannys Kantendetektor findet nämlich im Gegensatz zum SUSAN-Kantendetektor keine

Ecken, wie auch bereits in [1] beschrieben wird.

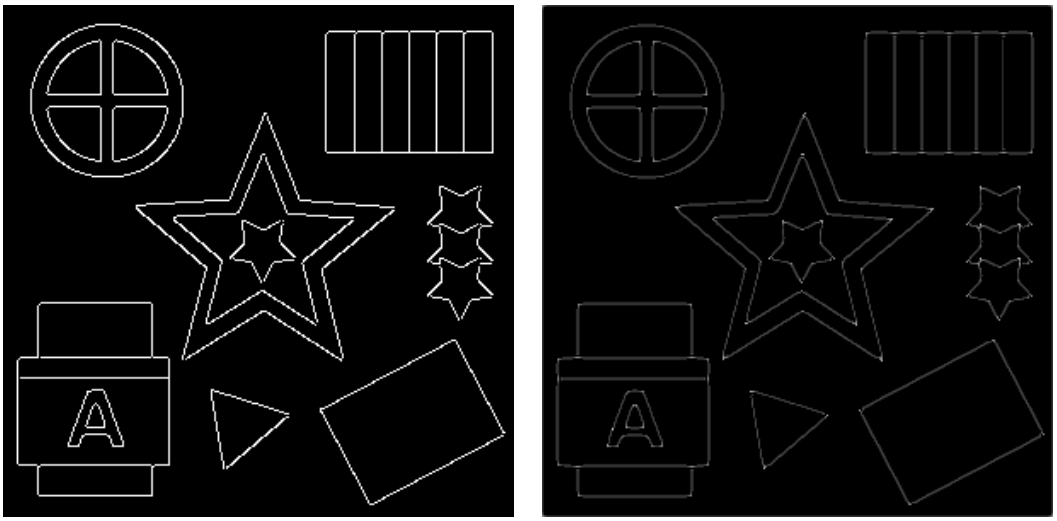

Abbildung 3.13: Links: Canny, Rechts: SUSAN.

3.3.3 Der Eckendetektor

3.4 Heuristik

3.4.1 Erklärung des SUSAN-Prinzips

Durch die folgende Heuristik aus [1] soll dargestellt werden, wie das Prinzip des SUSAN-Kantendetektors funktioniert. Die Heuristik ist nicht als Beweis zu verstehen, da für diese Heuristik Annahmen getroffen werden, die nicht auf Digitalbilder zutreffen.

Angenommen wir haben ein eindimensionales Bild. Unser Ziel ist es, genau die Stellen zu markieren, die den größten Grauwertunterschied zu ihrer Nachbarschaft haben. Das klassische Werkzeug der Analysis, um genau diese Stellen zu identifizieren, ist die Ableitung. Leiten wir eine zweimal differenzierbare, stetige Funktion I zweimal ab, so erhalten wir eine Funktion I'', dessen Nullstellen die Stellen des größten Unterschieds in I repräsentieren.

So eine Ableitung existiert allerdings nicht für Digitalbilder. Darum nehmen wir zunächst an, dass unser Bild eine solche Funktion $I : [0, B-1]_{\mathbb{R}} \to [0, 255]_{\mathbb{R}}$ ist. Implizit wird im Folgenden die Annahme getroffen, dass I eine bijektive Funktion ist. Das ist im Allgemeinen zwar nicht der Fall, allerdings liegt uns bei der digitalen Verarbeitung von Bildern im Gegensatz zur eindimensionalen Analysis immer beides vor: Die Position und die Intensität eines Pixels lassen sich hier durch geschicktes Speichern aufeinander zurückführen.

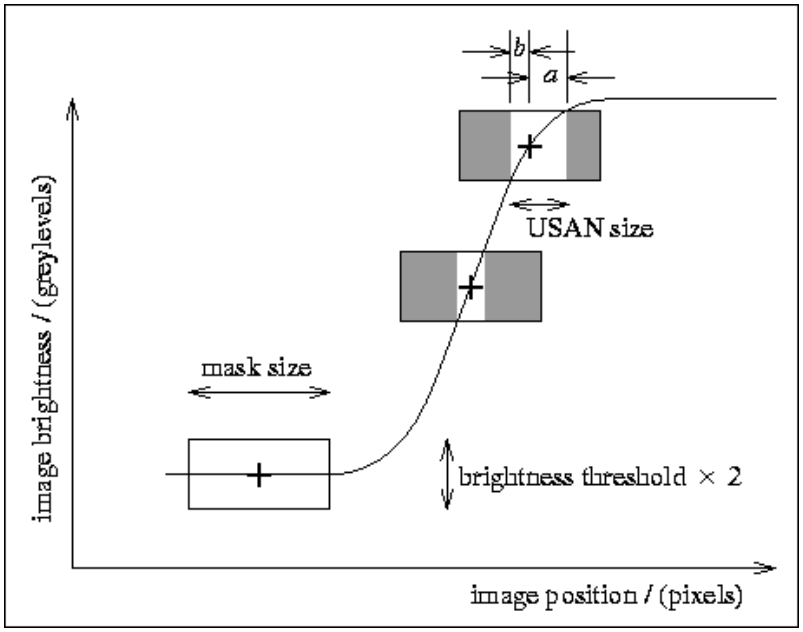

Abbildung 3.14: Das Bild I (Bild entnommen aus [1]). Das Kreuz steht für den Nucleus der USAN und die Kästen für die Maske.

Der SUSAN-Kantendetektor beruht auf folgendem Prinzip: Für einen Schwellwert t gibt es, abhängig von der spezifischen Stelle im Bild x_0 eine untere Grenze $x_0 + a(x_0)$ und eine obere Grenze $x_0 - b(x_0)$ der USAN. Dann gilt an den Grenzen:

$$I(x_0 + a(x_0)) = I(x_0) + t$$
$$I(x_0 - b(x_0)) = I(x_0) - t.$$

Das Prinzip des SUSAN-Kantendetektors lässt sich folgendermaßen formulieren: An genau den Stellen, an denen $n(x_0) = a(x_0) + b(x_0)$ ein Minimum hat, ist die Antwort $A(x_0) > 0$. Wir erhalten unter der Annahme, dass n an x_0 ein lokales Minimum erreicht:

$$n'(x_0) = a'(x_0) + b'(x_0) = 0$$

Die obigen Gleichungen können wir nach a und b umstellen. So erhalten wir

$$a(x_0) = x_0 - x(I(x_0) + t)$$
$$b(x_0) = x(I(x_0) - t) - x_0,$$

wobei x die Umkehrfunktion von I ist. Damit ist $n(x_0) = x(I(x_0) + t) - x(I(x_0) - t)$ und daraus folgt

$$n'(x_0) = x'(I(x_0) + t) \cdot I'(x_0) - x'(I(x_0) - t) \cdot I'(x_0) = 0,$$

anders formuliert,

$$x'(I(x_0)+t)\cdot I'(x_0) = x'(I(x_0)-t)\cdot I'(x_0)$$

und damit

$$x'(I(x_0)+t) = x'(I(x_0)-t),$$

falls die Bildfunktion nicht konstant ist. Es folgt

$$(I(x_0)+t)' = (I(x_0)-t)'.$$

Lassen wir t gegen Null laufen, so finden wir

$$\lim_{t\to 0} (I(x_0)+t)' - (I(x_0)-t)' = I''(x_0) = 0.$$

Demnach treffen sich die maximale Antwort des SUSAN-Kantendetektors und die zweite Ableitung der Bildfunktion. Tatsächlich verwendet der SUSAN-Kantendetektor allerdings keine Ableitungen. Der SUSAN-Algorithmus bestimmt lediglich die Anzahl der ähnlichen Pixel in einer Umgebung.

Abbildung 3.15: Filtermaske im 1D-Bild. Der Nucleus ist gelb markiert. Die orangenen Pixel liegen innerhalb der Maske. Die weißen Pixel liegen außerhalb der Maske.

In einem eindimensionalen Digitalbild $P : [0, B-1]_{\mathbb{Z}} \to [0, 255]_{\mathbb{Z}}$ identifizeren wir durch eine Maske wie zum Beispiel in Abbildung 3.15 die USAN der einzelnen Pixel, wie auch schon im Algorithmus in 3.1. Dabei stellen wir für jede USAN fest, wie viele Pixel r ähnlich zu dem Nucleus r_0 in der Mitte der Maske sind. Wir erinnern uns an die Definitionen aus dem Algorithmus

$$A(r_0) = \max\{0, g - n(r_0)\},\ n(r_0) = \sum_r c_t(r, r_0),\ c_t(r, r_0) \approx \begin{cases} 1 & \text{falls } |I(r) - I(r_0)| \le t \\ 0 & \text{sonst} \end{cases}.$$

Anhand eines Beispielbildes wird nun vorgeführt, wie der SUSAN-Kantendetektor eine Kante identifiziert.

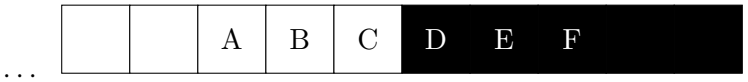

Abbildung 3.16: Die weißen Pixel haben den Grauwert 255, die schwarzen Pixel den Grauwert 0

Wir stellen fest, dass die mit A-F markierten Pixel verschiedene Anzahlen an ähnlichen Nachbarn in ihrer Maske haben.

$$n(A) = 4 \quad n(B) = 3 \quad n(C) = 2$$
$$n(D) = 2 \quad n(E) = 3 \quad n(F) = 4$$

Entsprechend finden wir eine Antwort mit $g = 4$ wie in Abbildung 3.17.

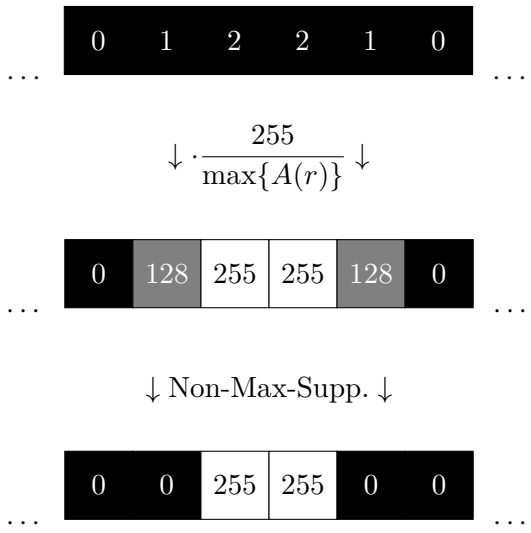

Abbildung 3.17: Antwort

Durch das SUSAN-Prinzip und die Non-Maximum-Suppression konnten wir die Kante ohne Ableitung bestimmen.

Literaturverzeichnis

[1] Stephen M. Smith, J. Michael Brady,
SUSAN - A New Approach to Low Level Image Processing,
International Journal of Computer Vision 23(1), 45-78,
1997

[2] Stephen M. Smith,
Edge Thinning Used in the SUSAN Edge Detector,
Technical Report TR95SMS5,
1995

[3] Robert Fisher, Simon Perkins, Ashley Walker, Erik Wolfart,
Gaussian Smoothing,
The Hypermedia Image Processing Reference 2,
2004

[4] Pillow: the friendly PIL fork,
`https://python-pillow.org/`

[5] NumPy,
`https://numpy.org/`

[6] multiprocessing - Process-based parallelism,
`https://docs.python.org/3/library/multiprocessing.html`

[7] Jake VanderPlas,
Why Python is Slow: Looking Under the Hood,
`https://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/`,
2014

[8] Markus Konrad,
Vectorization and parallelization in Python with NumPy and Pandas,
`https://datascience.blog.wzb.eu/2018/02/02/vectorization-and-parallelization-in-python-with-numpy-and-pandas/`,
2018