Bachelorarbeit

Erkennung von Sperrflächen aus Lasermessdaten

Verfasser:
Sebastian Reichl
07. September 2015

Prüfer:
Prof. Dr. Tomas Sauer
Innstraße 43
94032 Passau

Inhaltsverzeichnis

Zusammenfassung

Im Rahmen dieser Arbeit wird versucht, aus einer Punktwolke Sperrflächen zu extrahieren. Um eine automatische Erkennung der Flächen zu realisieren, werden zunächst Schritte zur Aufarbeitung der Daten ausgeführt, bevor die anschließende automatische Erkennung erfolgt. Es werden verschiedene Methoden dieser Aufarbeitung sowie verschiedene Algorithmen zur Erkennung von Sperrflächen vorgestellt und miteinander verglichen.

Kapitel A

Einführung

Da in der Automobilbranche eine stetige Weiterentwicklung herrscht, ist die Entwicklung neuer und die Verbesserung bestehender Systeme essentiell. Beispielsweise gewinnt die Spurerkennung in Autos im Rahmen des selbständigen Fahrens immer mehr an Wichtigkeit, weswegen natürlich auch gefordert wird, speziellere Formen im Straßenverkehr zu erkennen. Eine solche Form ist das Schrägstrichgatter, im Folgenden Sperrfläche genannt, welche für den Verkehr gesperrte Flächen markiert und unter anderem Unfälle verhindern soll.
Zu Beginn werden einige im späteren Verlauf verwendete Begriffe erklärt. Danach beschäftigt sich diese Arbeit mit der Überführung von Daten aus einer Lasermessung in ein verwertbares Bild und der Aufarbeitung in für den verwendeten Erkennungsalgorithmus verwertbare Daten. Die Lasermessung erfolgte durch einen auf einem fahrenden Auto angebrachten Laser, der aus ausgestrahlten und wieder aufgefangenen Laserstrahlen die Raumkoordinaten und den Reflektanzwert eines Punktes bestimmt. Im Anschluss an diese Aufarbeitung werden verschiedene Ansätze und Möglichkeiten der Verbesserung der erhaltenen Daten miteinander verglichen.
Im nächsten Kapitel wird die Implementierung und Funktionsweise des Erkennungsalgorithmus detailliert erklärt und zwei Möglichkeiten zur Erkennung der in den Daten enthaltenen Sperrflächen gegenübergestellt. Zuletzt wird das Programm mit Hilfe der vom FORWISS Passau bereitgestellten Daten getestet und ein Ausblick auf mögliche Erweiterungen wie eine Ausweitung auf eine größere Zahl von Sperrflächen gegeben.

I Problemstellung

Jede Zeile einer Datei, die dem Programm übergeben werden soll, ist durch eine Lasermessung entstanden und enthält 4 Werte, nämlich x-, y-, z- und Reflektanzwert und stellt einen Messpunkt dar. Daraus ergibt sich in der jeweils vorliegenden Datei eine Vielzahl an Daten, die erfasst und verarbei-

tet werden müssen. Bei einer Fahrt von ungefähr 100 Meter fallen hierbei schon mehr als 100.000 Messpunkte an. Außerdem müssen eventuell auftretende Fehler in den Daten ausgeglichen werden, sodass trotz fehlender Messpunkte ein zufriedenstellendes Ergebnis erzielt werden kann. Aus einer solchen Datei eine möglicherweise auftretende Sperrfläche zu extrahieren, ist die Hauptaufgabe dieser Arbeit.

II Konzept

Die Verarbeitung der Daten und das Erkennen der Sperrfläche soll durch die Erzeugung eines Graphen und Bewertung durch einen Klassifizierer erfolgen und als Ausgabe den Anfang, das Ende und die Anzahl der Schräglinien umfassen. Zuerst müssen die Daten jedoch in ein verwertbares Format überführt werden, was durch teils von OpenCV bereitgestellte und teils durch selbst programmierte beziehungsweise übernommene Methoden erfolgt.

III Details zur Implementierung

Diese Arbeit befasst sich außerdem ausschließlich mit Sperrflächen auf Landstraßen mit mindestens 3 Schrägstrichen. Ausgeschlossen werden zudem zu sehr durch Umwelteinflüsse wie haltende LKW, Automobile oder zu sehr ausgewaschene Markierungen kompromittierte Messdaten. Die Implementierung erfolgte unter Java 7 Update 45 mit der Entwicklungsumgebung Eclipse Kepler Service Release 1. Für Teile der Bildverarbeitung wurde OpenCV 3.0.0 verwendet.

IV Begriffserklärung

1 OpenCV

Abbildung 1: Logo von OpenCV[1]

Die Open Source Computer Vision Library, oder kurz OpenCV, ist eine im September 2006 in der Version 1.0 erschienene Open Source Bibliothek für Maschinensehen und Maschinenlernen mit BSD-Lizenz des Entwicklerteams von itseez[2]. Sie umfasst mehr als 2500 optimierte Algorithmen, die für verschiedenste Anforderungen der Bildverarbeitung, wie Erkennung und Verfolgung von Objekten, genutzt werden können. Die Bibliothek ist unter C, C++, Python, Java und MATLAB verfügbar und funktioniert auf den Betriebssystemen Windows,

[1] Quelle: https://opencv.org/about.html, aufgerufen am 22.07.2015, 14:00 Uhr
[2] http://itseez.com/

Linux, Android und Mac Os. Aktuell ist Version 3.0.0, die am 4.6.2015 erschienen ist. Diese wird auch in dieser Arbeit verwendet.

2 Adaptive Thresholding

Das Thresholding[3] stellt eine Methode zur Überführung eines Graustufenbildes in ein Schwarz-Weiß-Bild dar, wobei ein bestimmter Wert festgelegt wird, mit dem die Entscheidung getroffen wird, ob ein Pixel auf schwarz oder auf weiß gesetzt wird. Überschreitet der Farbwert eines Pixels diesen Wert, wird der Pixel als weiß identifiziert und sein Farbwert auf den Wert 255 geändert, sonst wird er als schwarz angesehen und auf 0 gesetzt.
Das adaptive Thresholding[2] stellt eine spezielle Form des Thresholding dar, wobei hier nicht ein globaler Wert für das ganze Bild benutzt, sondern für jedes Pixel ein eigener Schwellwert berechnet wird. Das hilft beispielsweise dabei, ausgewaschene Linien besser zu verfolgen oder den Schwellwert auf sich ändernde Beleuchtung[5] auf dem Bild anzupassen. Dieser Wert wird berechnet, indem eine Umgebung einer vom Benutzer festgelegten Größe betrachtet wird und in dieser alle Pixelwerte betrachtet werden. Hierzu kann zum Beispiel der Median aller Werte benutzt und als Threshold-Wert verwendet werden, oder eine gewichtete Gauß-Funktion, um aus der Summe aller dieser Werte den Schwellwert zu bestimmen.

3 Medianfilter

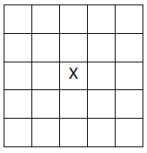

Abbildung 2: Eine 5x5 Maske.

Der Median, auch Zentralwert genannt, ist der mittlere Wert einer Auflistung an Zahlenwerten. Beim gleichnamigen Filter wird die direkte Nachbarschaft des aktuell zu betrachtenden Pixels einbezogen, um so einen neuen Wert für diesen zu bestimmen. Das Prinzip des Filters ist hierbei sehr simpel: Der Wert des aktuellen Pixels wird durch den Median aller umgebenden Pixelwerte ersetzt. Möglich sind hierbei immer ungerade Größen von quadratischen Masken, wie 3x3 oder 5x5. Der Vorteil dieses Filters zeigt sich bei sogenanntem „Salt & Pepper - Noise“, also Verunreinigungen auf dem Bild, die aus rein weißen und rein schwarzen Pixeln bestehen, da diese Ausreißer durch den Median sehr effektiv ersetzt werden können[1].

4 Gauß-Filter

Im Gegensatz zum Medianfilter wird hier nicht einfach ein Wert der Umgebung als neuer Pixelwert benutzt, sondern der neue Pixelwert wird durch eine gewichtete Gauß-Funktion bestimmt, wobei sie bei der Anwendung in einer Matrix angenähert wird. Diese fällt je nach Größe des Kernel verschieden aus, Abb. 3 zeigt eine solche Matrix bei einem 3x3 großen Kernel. Der

Wert des gerade betrachteten Pixels erhält hierbei das Gewicht 4, die Nachbarn jeweils entweder 2 oder 1. Es wird also der Pixelwert aller Pixel im Kernel mit dem jeweiligen Gewicht multipliziert, alle addiert und anschließend durch die Summe aller Gewichte – hier 16 – geteilt.[1].

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

Abbildung 3: Matrix eines Gaussfilters.[3]

5 Normalisierter Box-Filter

Ähnlich zum Gauß-Filter bedient sich der normalisierte Box-Filter einer Matrix, nach der die Pixelwerte errechnet werden. Allerdings gehen hier alle Pixel mit dem gleichen Gewicht ein und die errechnete Summe wird durch die jeweilige Zahl der betrachteten Pixel geteilt, weswegen er zu den linearen Filtern gehört. Außerdem kann durch ihn der Gauß-Filter angenähert werden; so approximiert dieser nach dreimaliger Ausführung den Kernel eines Gauß-Filters auf 3% genau[1].

6 Erosion

Bei der morphologischen Erosion wird mit Hilfe eines Strukturelements ein Teil eines Objektes „weggeschliffen“, um so feinere Linien zu erhalten oder unerwünschte Artefakte zu entfernen. A bezeichnet im folgenden ein solches Objekt, an dem eine Erosion mittels eines Strukturelementes B durchgeführt werden soll. B_z ist das erwähnte Strukturelement, das in das Objekt A gelegt wurde und den Mittelpunkt z besitzt. Dieser Punkt z ist ein beliebiger Punkt im Objekt A. Die Erosion lässt sich ausdrücken durch:

$$A \ominus B = \{z \in E \mid B_z \subseteq A\},$$

Ein Punkt von A bleibt also nach der Erosion erhalten, wenn ein Objekt B_z vollständig in A enthalten ist. Betrachtet man Abb. 4, heißt das: Sollte ein Punkt von B_z über die schwarze Fläche von A herausragen, wird der Punkt z bei der Erosion gelöscht[1].

[3]Quelle: https://en.wikipedia.org/wiki/Kernel_(image_processing), aufgerufen am 21.07.2015, 20:00 Uhr

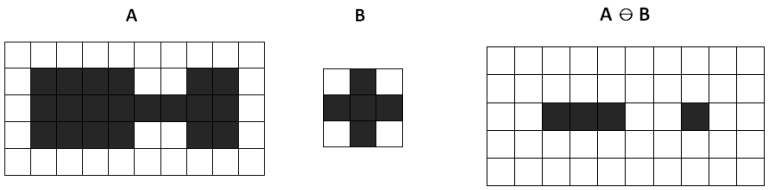

Abbildung 4: Beispiel einer Erosion mit A als Ausgangsbild und B als Strukturelement

7 Skelettierung

Ein Punkt in dem Skelett eines Objektes ist der Mittelpunkt des größten Kreises, der zwischen den Konturen dieses Objektes enthalten ist. Dabei berührt dieser Kreis zwei oder mehr Punkte der Grenzen des Objektes. Die Implementierung dieses Ansatzes erfolgt mithilfe des Algorithmus von Gonzales[6].
Hierbei wird eine Maske auf die Nachbarschaft eines Pixels angewandt. Abb. 5 zeigt die Nachbarschaft des zu betrachtenden Pixels p_1. Für diese Matrix werden Regeln aufgestellt, anhand derer ein Pixel zur Löschung markiert wird, sollte der betrachtete Pixel alle diese Regeln erfüllen. Der Algorithmus folgt zwei Stufen. In der ersten Stufe wird ein Set an Regeln verwendet, um Pixel an der Objektgrenze zur Löschung zu markieren, sollten sie alle Regeln erfüllen. Diese sind:

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

Abbildung 5: Nachbarschaftsregelung des „Thinning“ Algorithmus

(a) $2 \leq N(p_1) \leq 6$

(b) $T(p_1) = 1$

(c) $p_2 \cdot p_4 \cdot p_6 = 0$

(d) $p_4 \cdot p_6 \cdot p_8 = 0$

Hierbei ist $N(p_1)$ die Anzahl von Nachbarn ungleich 0 und $T(p_1)$ die Anzahl der Schwarz-Weiß-Übergänge in der Sequenz p_2, p_3,..., p_9.

Nach der Löschung der markierten Pixel folgt die zweite Stufe, bei der ein zweiter Satz Regeln auf die verbleibenden Pixel angewendet wird, wobei (a) und (b) gleich bleiben:

(c') $p_2 \cdot p_4 \cdot p_8 = 0$

(d') $p_2 \cdot p_6 \cdot p_8 = 0$

Dies wird solange wiederholt, bis in den Daten keine Änderung mehr erfolgt. Nachdem der Algorithmus terminiert, erhält man eine einen Pixel breite Linie, deren Punkte immer den gleichen Abstand von beiden Objektgrenzen haben[7].

8 Pruning

Pruning stellt eine wichtige Ergänzung zur Skelettierung dar, da bei dieser unerwünschte Artefakte auftreten können, wie kleine Linien an den eigentlich erwünschten Linien. Hierbei werden also Äste, die eine bestimmte Länge unterschreiten, entfernt. Auch hier wird ein Strukturelement verwendet, mit dessen Hilfe die Endpunkte des entsprechenden Astes immer wieder gelöscht werden und so parasitäre Äste erkannt werden können. Erreicht wird dies durch das Kronecker-Produkt des Ausgangsbildes mit einer Sequenz von Strukturelementen B mit dem Ziel, nur Endpunkte zu finden:

$$X_1 = A \otimes \{B\}$$

Das Strukturelement besteht aus zwei 3x3 Matrizen, die jeweils um 0°, 90°, 180°und 270°gedreht werden, sodass insgesamt acht Strukturelemente zur Anwendung kommen. Diese Gleichung lässt sich je nach Einstellung durch den Benutzer beliebig oft wiederholen und hat als Ergebnis ein Bild ohne parasitäre Äste, jedoch sind auch manche erwünschte Daten verschwunden. Das macht eine Rekonstruktion notwendig, die dadurch erfolgt, dass zunächst mittels

$$X_2 = \bigcup_{k=1}^{8} (X_1 * B^k)$$

ein Datenset erzeugt wird, das nur die Endpunkte enthält. B^k sind die verschiedenen Strukturelemente, die gedreht wurden. Als nächstes werden diese Endpunkte drei mal erweitert, wobei A als Begrenzung benutzt wird:

$$X_3 = (X_2 \oplus H) \cap A$$

H ist hier ein Strukturelement, dass nur Einsen enthält. Der Schnitt mit A verhindert, dass Punkte außerhalb der gewünschten Fläche entstehen. Schließlich wird durch die Vereinigung von X_3 und X_1 das erwünschte Ergebnis erzielt.

$$X_4 = X_1 \cup X_3$$

So entsteht ein Bild ohne parasitäre Äste[6].

Kapitel B

Überführung der Rohdaten in verwertbare Daten

I Einlesen der Daten

Ausgangspunkt für diese Arbeit sind durch Speicherung von Messpunkten einer Lasermessung entstandene .csv–Dateien, die eingelesen und verarbeitet werden. Diese Dateien enthalten, je nach Aufnahmedauer, eine Vielzahl von Messpunkten (siehe Abb. 6), die betrachtet und zunächst zu einem Graustufenbild verarbeitet werden.

```
0.78795,8.18559,1.79331,0.272
0.704498,8.26937,1.75926,0.101
0.610954,8.18941,1.72111,0.236
0.537743,8.56478,1.69122,0.209
0.442224,8.46063,1.65227,0.362
0.345248,8.15829,1.61273,0.257
0.255771,7.96969,1.57625,0.176
0.169957,7.94798,1.54125,0.233
```

Abbildung 6: Ausschnitt aus einer .csv Datei

Da der Laser auf einem Auto montiert ist, sind die Daten in Linien quer zur Fahrtrichtung aufgenommen worden. Um diese Linien, in dieser Arbeit als *ScanLines* bezeichnet, auch in einem Bild zu erhalten und auch einfacher auf ihnen arbeiten zu können, wurde der euklidische Abstand zweier Punkte betrachtet, die zuvor mittels eines Filestreams aus einem String eingelesen wurden. Überschreitet dieser einen Schwellwert, werden alle nachfolgenden Datenpunkte in eine neue Linie eingetragen. Dies wird dann durch Skalierung der jeweiligen Koordinaten in ein zweidimensionales Array überführt (siehe Abb. 7).

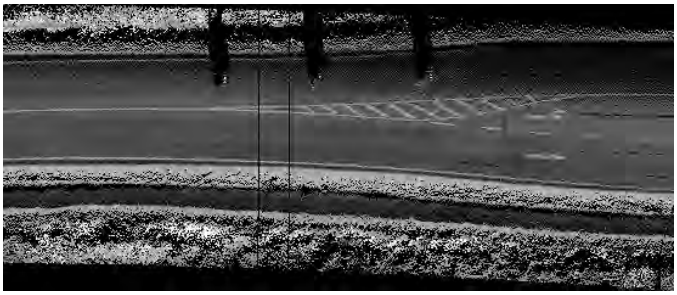

Abbildung 7: In ScanLines eingeteiltes Bild

II Verarbeiten der Daten

1 Schließung von Lücken in den Daten

Dadurch, dass die Erfassung der Daten durch einen auf einem fahrenden Auto montierten Laser erfolgt, können eventuell auftretende Hindernisse die Reflektanz eines Punktes verhindern, wodurch Lücken entstehen. Um diese Lücken zu schließen, wurden die fehlenden Daten mittels Interpolation aus vorhandenen Daten ermittelt. Hierbei wurden zwei Möglichkeiten der Interpolation verglichen, nämlich die horizontale Interpolation und die Berechnung durch Einbeziehung der zweidimensionalen Nachbarschaft. Bei der horizontalen Interpolation werden die Werte links und rechts neben dem zu füllenden Wert betrachtet und je nach Entfernung ein neuer Wert bestimmt. Das Ergebnis zeigt Abb.8. Bei der Interpolation unter Berücksichtigung der zweidimensionalen Nachbarschaft wird die nähere Umgebung des Pixels untersucht und mit einer Maske Gewichte für die jeweiligen Werte Nachbarschaft vergeben und so ein neuer Wert für den zuvor schwarzen Pixel bestimmt. Wie in Abb.9 zu sehen ist, unterscheidet sich das Ergebnis deutlich von dem der horizontalen Interpolation und wurde in dieser Arbeit favorisiert.

Abbildung 8: Bild nach horizontaler Interpolation

Abbildung 9: Interpolation unter Berücksichtigung der Nachbarschaft

2 Überführung in ein Schwarz-Weiß-Bild

Die Überführung in ein diskretes Bild erfolgt in mehreren Schritten und auch hier wurden verschiedene Methoden getestet und verglichen.

a Sliding Window 1

Das Sliding Window enthält immer die letzten zehn Werte, die nicht als weiß identifiziert wurden. Anschließend werden die Punkte noch mit ihren – nicht weißen – linken Nachbarn so verglichen, dass der Durchschnitt ihrer Reflektanzen mit der des zu prüfenden Wertes verglichen wird. Liegt dieser Wert 19% über dem der Nachbarn, wird er als weiß identifiziert. Diese Zahl wurde experimentell als die identifiziert, bei der genug Störeinflüsse entfallen, jedoch nicht viele wichtige Punkte verloren gehen. Nachteil dieses Ansatzes, bei dem in jeder Zeile beim ersten Wert einer ScanLine begonnen wird, ist jedoch, dass die ersten Werte einer ScanLine so nicht eingeteilt werden können. Sie besitzen keine letzten zehn Werte anhand derer sie eingeteilt werden können. So wird hier der erste Punkt mit Hilfe eines fest eingestellten Wertes überprüft und dann zum Sliding Window hinzugefügt. So geschieht es mit allen Werten, bis zehn im Sliding Window vorhanden sind; danach greift die Regel, dass nur Werte, die als schwarz identifiziert wurden, mit ihrem ursprünglichen Reflektanzwert zum Sliding Window hinzugefügt werden. Das Ergebnis zeigt Abb.10.

b Sliding Window 2

Dieser verbesserte Ansatz arbeitet so, dass die Prüfung in der Mitte einer ScanLine startet und in zwei Teilen erfolgt. So werden erst von der Mitte aus nach unten zehn Werte zu der Liste der letzten Werte hinzugefügt. Danach werden die Punkte nach und nach verglichen und immer, wenn ein Punkt 19% über dem Durchschnittswert liegt, auf weiß gesetzt. Diese Zahl wurde aus dem vorherigen Ansatz übernommen, da sie sich auch hier als guter Werte herausstellte. Als schwarze identifizierte Punkte werden zu dieser Liste hinzugefügt, während der älteste verdrängt wird. Anschließend werden wieder die drei linken Nachbarn betrachtet. Überschreitet der zu betrachtende Punkt auch deren Durchschnittsreflektanz, wird er endgültig als weiß identifiziert. Das Ergebnis zeigt Abb. 11.

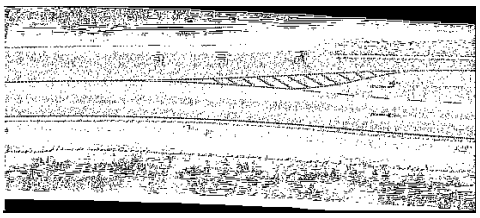

Abbildung 10: Sliding Window 1

Abbildung 11: Sliding Window 2

c Adaptive Threshold

Vergleichsweise wurde die von OpenCV zur Verfügung gestellt Methode des Adaptive Thresholding benutzt[4]. Auf den ersten Blick ist dort allerdings mehr Rauschen vorhanden, was aber mit den nachfolgend beschriebenen Algorithmen minimiert werden konnte. Der Vorteil bei dieser Art der Überführung ist, dass von OpenCV zur Verfügung gestellte Methoden ausgiebig getestet wurden sowie von Version zu Version optimiert und weitläufig benutzt werden. An Abb. 12 erkennt man bei genauem Betrachten einen Kanal aus schwarzen Pixeln um die Sperrfläche. Das erleichtert die Trennung zwischen Sperrfläche und Rauschen bzw. anderen Fehlern der Aufnahme.

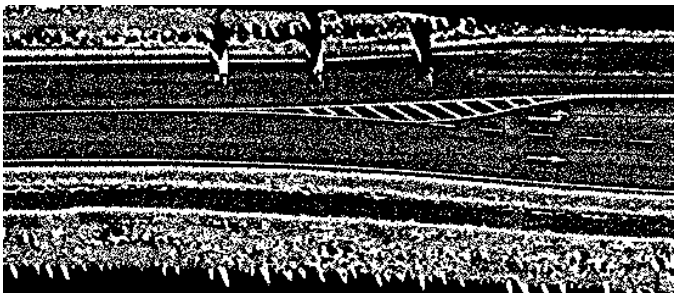

Abbildung 12: Bild nach adaptive Threshold

3 Entfernen von unerwünschten Artefakten

a Entfernen durch Anzahl von Nachbarn

Bei diesem Ansatz wurde die Nachbarschaft jedes Punktes untersucht und ein Zähler geführt, der darüber Auskunft gibt, wie viele Nachbarn der zu betrachtende Punkt besitzt. Wurde eine Anzahl von drei Nachbarn nicht erreicht oder eine Anzahl von sieben Nachbarn überschritten, wurde der Farbwert des Punktes auf 0 (schwarz) gesetzt. Diese Werte wurden gewählt, da ein Pixel außerhalb einer Sperrfläche meist weniger als drei Nachbarpixel der selben Farbe und ein Pixel in einem großen Rauschartefakt nur weiße Nachbarn besitzt. Abbildung 13 zeigt diesen Ansatz in Verbindung mit Sliding Window 2. Nachteil hierbei ist, dass auch Punkte der Sperrfläche

gelöscht wurden und andere Artefakte erhalten blieben, die nicht erwünscht sind.

Abbildung 13: Sliding Window 2 nach Entfernung einzelner Punkte

b Entfernen durch Zusammenhangskomponente

Eine Verbesserung bringt hier eine Zusammenhangskomponente. Eine Zusammenhangskomponente ist eine zusammenhängende Fläche, die aus mehreren Pixeln besteht. Ob Pixel zusammenhängen oder nicht, kann durch die Bewertung ihrer Nachbarschaft entschieden werden. Hierzu gibt es zwei Möglichkeiten:

- Betrachtung der Vierernachbarschaft und
- Betrachtung der Achternachbarschaft

Bei der Vierernachbarschaft werden die Pixel ober- und unterhalb des aktuell zu betrachtenden Pixels sowie diese links und rechts davon betrachtet. Sollte ein dort ein noch nicht betrachteter Pixel vorgefunden werden, wird mit diesem nach demselben Schema verfahren. Die Achternachbarschaft betrachtet zusätzlich zu den Nachbarn, die durch die Vierernachbarschaft betrachtet wurden, noch die, die an den Ecken eines Vierecks liegen, dessen Mittelpunkt der aktuelle Pixel ist. Mit gefundenen Pixeln wird gleich verfahren. Diese zwei Möglichkeiten bieten Vor- und Nachteile: Die Vierernachbarschaft birgt den Vorteil, dass mehr unerwünschte Artefakte entfernt werden, allerdings können durch sie auch gewollte Elemente entfallen, da zum Beispiel die Aufnahme nicht optimal ist und Lücken zwischen Linien vorhanden sind. Die Achternachbarschaft hingegen hat diesen Nachteil nicht, jedoch bleiben hier mehr Artefakte erhalten, da hier alle umgebenden Pixel betrachtet werden. Die Entscheidung fiel auf diese, da der Nachteil der Artefakte nicht so schwerwiegend ist wie der Verlust eventuell wichtiger Linien. Den Unterschied zwischen beiden Ansätzen in Verbindung mit Sliding Window 2 zeigen die Abbildungen 14 & 15. Hier tritt der Vorteil des adaptive Thresholding noch besser hervor, wie man anhand von Abb. 16 erkennt. Hier sind weniger Punkte der eigentlichen Sperrfläche gelöscht und das Rauschen zuverlässiger

eliminiert. Es sind zwar außer der Sperrfläche noch andere größere Artefakte vorhanden, allerdings kann man mit diesen besser umgehen als mit kleinen Störartefakten.

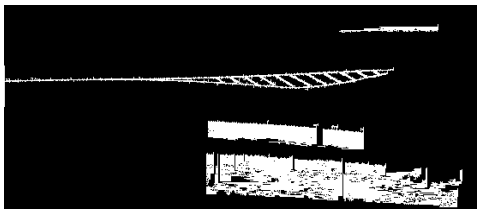

Abbildung 14: Zusammenhangskomponente mit Vierernachbarschaft

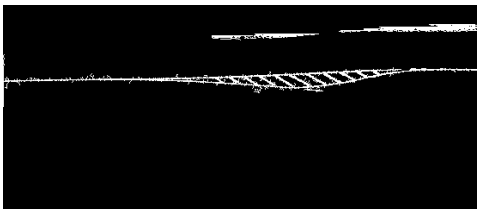

Abbildung 15: Zusammenhangskomponente mit Achternachbarschaft

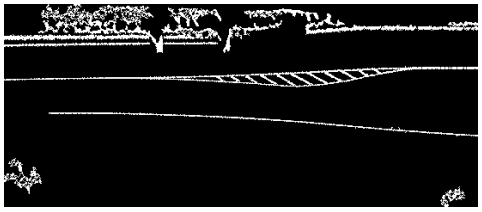

Abbildung 16: Zusammenhangskomponente mit Achternachbarschaft nach adaptive Thresholding

4 Optimierung der Bilder

a Anwendung verschiedener Filter

Nach der Entwicklung der Algorithmen zur Veränderung des Bildes wurde versucht, mit Hilfe der OpenCV Bibliothek das Ergebnis zu verbessern. Hierzu wurde zuerst ein Medianfilter bzw. ein normalisierter Box-Filter auf das Graustufenbild (siehe Abb 17) angewandt (siehe Abb. 17 und 21). Anschließend wurde die Eliminierung von Störeinflüssen durch eine Zusammenhangskomponente nach vorheriger Anwendung von Sliding Window 2 bzw. Adaptive Thresholding (siehe Abb. 18 und 22) betrachtet. Im Gegensatz hierzu ist der Gauß-Filter zu nennen, dessen Einfluss auf das Ergebnis ebenfalls getestet wurde (siehe Abb. 19 und Abb. 20). Der Zweck war, Verbesserungen im Bild zu erreichen und so ein klareres Bild zu erhalten, mit dem das Programm besser umgehen kann. Das beste Ergebnis erzeugte hierbei der Box-Filter vor adaptivem Thresholding (siehe Abb. 22).

Abbildung 17: Medianfilter auf Graustufenbild

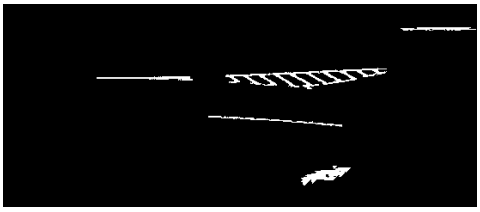

Abbildung 18: Nach Überführung in ein Schwarz-Weiß-Bild

Abbildung 19: Gauß-Filter auf Graustufenbild

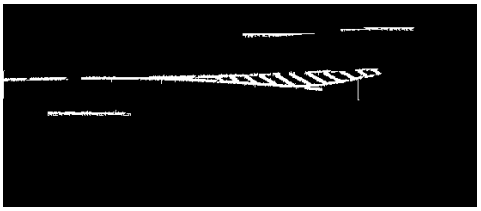

Abbildung 20: Nach Überführung in ein Schwarz-Weiß-Bild

Abbildung 21: Graustufenbild nach Anwendung des Box-Filters

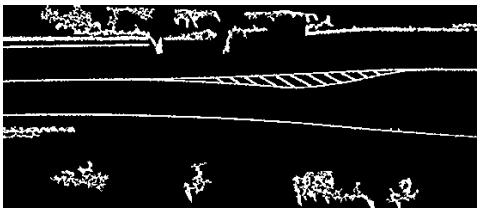

Abbildung 22: Nach Überführung in ein Schwarz-Weiß-Bild

b Erosion

Anschließend wurde die Auswirkung einer Erosion betrachtet, da diese Linien ausdünnt und so wichtige Merkmale leichter erkennbar und auch die Unterscheidung zwischen Linien einfacher macht. Zuerst wurde die Erosion ohne vorherige Filterung und im Vergleich dazu nach der Anwendung von Median- bzw. Gauß-Filter betrachtet. Das Ergebnis ist in Abb. 23, Abb. 24 und Abb. 25 zu sehen. Nach diesen Ergebnissen wurde nur noch mit dem Ansatz des adaptiven Thresholding gearbeitet.

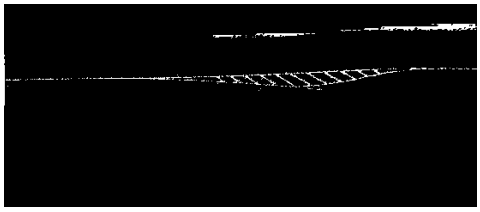

Abbildung 23: Erosion ohne vorherige Filterung

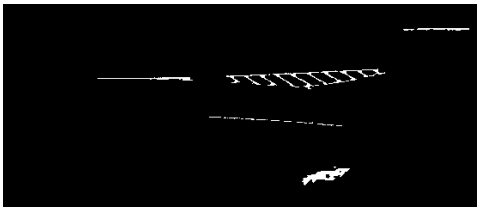

Abbildung 24: Erosion nach Medianfilterung

Abbildung 25: Erosion nach Gauß-Filterung

c Skelettierung

Da der Ansatz der Anwendung einer morphologischen Erosion nicht zielführend war, wurde die morphologische Skelettierung herangezogen. Diese erzeugt aus mehreren Pixel dicken Linien solche, die nur noch ein Pixel breit sind. Dies wird deswegen favorisiert, da auf diesen ausgedünnten Linien besser gearbeitet werden kann und die Erkennung verbessert sowie die Anzahl der Fehlerquellen minimiert wird. Das Ergebnis zeigt Abb. 26, bei der zusätzlich zum Adaptive Thresholding und dem Entfernen der Artefakte durch eine Zusammenhangskomponente zusätzlich noch ein sogenanntes Pruning durchgeführt wurde, um Artefakte zu entfernen. Abb. 27 zeigt das Ergebnis, nachdem zuvor noch ein Medianfilter zur Anwendung kam. Abb. 28 zeigt dasselbe nach Anwendung eines Gauß-Filters und Abb. 29 nach Anwendung eines normalisierten Box-Filters, der hier die besten Ergebnisse liefert.

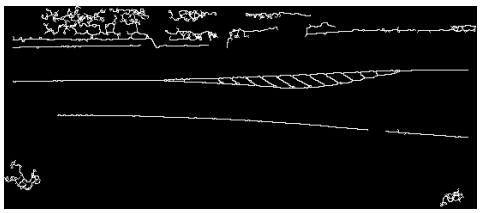

Abbildung 26: Skelettierung ohne vorherige Filterung

Abbildung 27: Skelettierung nach Anwendung eines Medianfilters

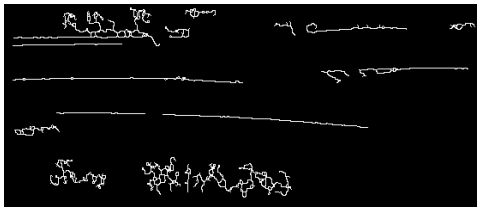

Abbildung 28: Skelettierung
nach Anwendung eines Gauß-Filters

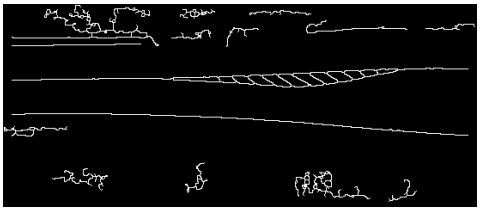

Abbildung 29: Skelettierung
nach Anwendung eines Box-Filters

Kapitel C

Erkennung der Sperrflächen

Der Ablauf des Algorithmus erfolgt in mehreren Schritten.

I Aufbau des Graphen

Um den Graphen aufbauen zu können, müssen zuerst Knoten erkannt werden. Dies wird durch die Verfolgung des Linienverlaufs bewerkstelligt, wobei immer eine scanLine mit der vorherigen verglichen wird. Hier können 5 Situationen auftreten, die zu verschiedenen Methodenaufrufen führen (Pixel wird hier als Synonym einer beliebig langen ununterbrochenen Folge von einem bis mehreren Pixeln verwendet):

(1) beginLine	In der aktuellen scanLine tritt ein Pixel ohne Nachbarlinie auf; dieser Pixel wird als Knoten in den Graphen eingetragen und als Anfangsknoten der Linie gesetzt.
(2) continue	Der aktuell betrachtete Pixel hat genau eine Nachbarlinie. Als Anfangsknoten wird der Anfangsknoten des Nachbarn übernommen, aber kein neuer Knoten eingetragen.
(3) divideLine	Zwei Pixel der aktuell betrachteten scanLine besitzen den selben Nachbarn. Dieser Nachbar wird als neuer Knoten in den Graphen eingetragen, eine neue Kante mit dem neuen Knoten als Ende und dem gemerkten Anfangsknoten erzeugt und ebenfalls in den Graphen eingetragen. Anschließend wird der neue Knoten bei den zwei neu entstandenen Linien als Anfangsknoten gesetzt.

(4) mergeLine Ein Pixel der aktuell betrachteten scanLine besitzt zwei Nachbarlinien. Dieser Pixel wird als neuer Knoten in den Graphen eingetragen, zwei neue Kanten mit jeweils den beiden Anfangsknoten der zusammentreffenden Linien und dem neuen Knoten als Ende erzeugt und ebenfalls in den Graphen eingetragen. Anschließend wird der neue Knoten als Anfangsknoten der neuen Linie gesetzt.

(5) endLine Ein Pixel der vorher betrachteten scanLine besitzt keinen Nachbarn in der aktuellen scanLine. Es werden ein neuer Knoten und eine neue Kante erzeugt, wobei der neue Knoten der Punkt der vorherigen Linie ist und als Endknoten der neuen Kante dient.

II Elimination unnötiger Knoten

Da eine Sperrfläche aus einer durchgehenden Markierung entsteht und auch wieder in einer solchen endet, müssen diese Anfangs- und Endpunkte mit ihren Kanten entfernt werden, damit nur noch die Sperrfläche an sich erhalten bleibt. Außerdem werden dadurch viele Punkte des Rauschens entfernt und so kann die Sperrfläche besser extrahiert werden, da weniger Störeinflüsse erkannt werden. Dies wird dadurch erreicht, dass Knoten inklusive ihrer anliegenden Kanten entfernt werden, die weniger als zwei solche Kanten besitzen, wie Abb. 31 zeigt. Dies ist außerdem für die Extraktion von Sperrflächen erforderlich.

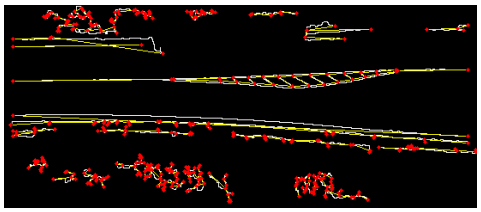

Abbildung 30: Der nach den vorgestellten Regeln aufgebaute Graph.

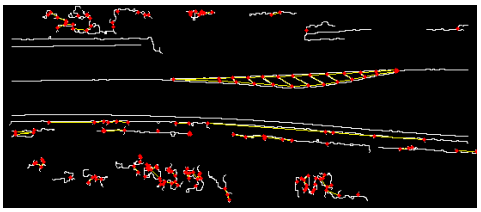

Abbildung 31: Ein Graph ohne unnötige Knoten.

III Unterteilung des Graphen in zusammenhängende Teilgraphen

Nach der Elimination unnötiger Knoten ist ein Graph vorhanden, der noch alle Knoten und Kanten eines Bildes enthält, die vor der Elimination mehr als zwei Nachbarn hatten. Da man hier allerdings Sperrflächen nicht ein-

fach erkennen kann, wurde es notwendig, den großen Graphen in zusammenhängende Teilgraphen zu unterteilen, die dann an die Erkennungsalgorithmen übergeben werden können. Hierzu wurden alle Knoten nacheinander betrachtet und bei Betrachtung eines noch nicht besuchten Knotens wurde ein neuer Teilgraph erzeugt. Alle noch nicht besuchten Nachbarn dieses Teilgraphen wurden dann zu einer Liste hinzugefügt. Die Nachbarn der so neu hinzugefügten Knoten wurden wiederum zur Liste hinzugefügt, bis alle zusammenhängenden Knoten eines Teilgraphen besucht wurden und kein neuer Knoten hinzugefügt werden konnte. So entsteht eine Liste von Teilgraphen aus dem ursprünglichen Graphen, mit der nun weitergearbeitet werden kann.

IV Extraktion der Sperrfläche

1 Erkennung perfekter Sperrflächen

Durch verschiedene Störeinflüsse können auch noch Graphen im Bild enthalten sein, in denen auch Knoten (rot markiert) und Kanten (gelb markiert) gefunden werden können, wie in Abb. 31 zu sehen ist, aber keine Sperrfläche enthalten. Um aus diesen Graphen denjenigen oder diejenigen zu extrahieren, die eine Sperrfläche enthalten, wird eine bestimmte Eigenschaft dieser Graphen ausgenutzt. Die Aufteilung der Knoten in einer Sperrfläche ist so, dass man am unteren Rand der Sperrfläche bis zu ihrem Ende und am oberen zurück zum Anfang laufen kann, ohne einen Knoten zweimal zu besuchen. Auf diese Weise kann eine Sperrfläche von Rauschen unterschieden werden, da Rauschen nicht so regelmäßig ist; außerdem können so mehrere Sperrflächen in ein und demselben Bild erkannt werden. Nach dem Ausführen dieses Schrittes stellt sich das Ergebnis wie folgt dar:

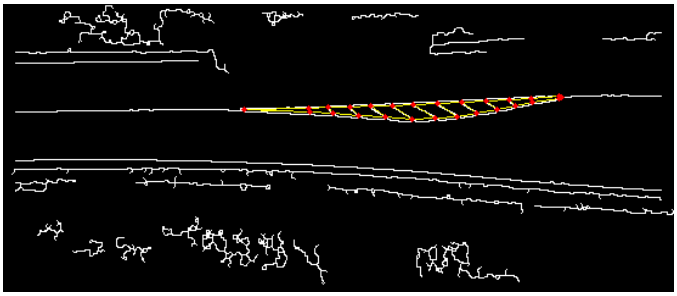

Abbildung 32: Als Sperrfläche erkannter Graph

Man sieht, dass nur noch die Sperrfläche markiert ist und so das Ziel erreicht wurde. Anschließend kann die Position der Sperrfläche in Form ihres ersten und letzten Punktes ausgegeben werden.

Die Anzahl der Schräglinien wird über die Anzahl der Knoten bestimmt, nämlich:

$$\frac{AnzahlKnoten-2}{2}$$

Da jede Schräglinie zwei Knoten besitzt und Anfangs- und Endknoten keine Schräglinienknoten darstellen, müssen diese zwei Knoten aus der Gesamtzahl der Knoten herausgerechnet werden. So kann mithilfe des Graphen die Problemstellung gelöst werden.

2 Erkennung durch Bewertungsfunktion

Ein weiteres Problem stellt sich jedoch bei der Betrachtung von Sperrflächen heraus, die nicht ganz perfekt sind, da zum Beispiel durch die vorherige Verarbeitung Informationen verloren gingen. Um jedoch auch diese Sperrflächen erkennen zu können, wurde eine Funktion zur Bewertung von Teilgraphen und Erkennung eventuell enthaltener Sperrflächen erarbeitet. Eine Sperrfläche besteht im Idealfall aus:

- Zwei Knoten mit zwei Nachbarn (b)
- Mindestens sechs Knoten mit drei Nachbarn (t)
- Kein Knoten mit mehr als drei Nachbarn (o)

Nach diesen Eigenschaften wurde die Funktion

$$f : \mathbb{N}^3 \to [0,1], f(t,b,o) = \frac{t}{|b-2| + max(0, 6-t) + 3 * o + t}$$

entwickelt, die diese Bewertung vornehmen soll. Diese Form wurde gewählt, da eine perfekte Sperrfläche eine Bewertung von 1 bekommen sollte und solche, die Fehler aufweisen, eine niedrigere. Das Vorhandensein von mehr Knoten mit drei Nachbarn sollte die Bewertung erhöhen und mit Auftreten von Knoten mit mehr als drei Nachbarn herabstufen. Nach Berechnung des Wertes wurde dieser auf das Überschreiten eines Schwellwertes überprüft, wobei sich 0,8 als ein stabiler Wert herausstellte. Auch hier ist die Erkennung von mehreren Sperrflächen möglich, da auch hier nacheinander jeder Teilgraph des Bildes untersucht wird. Hier wird die Anzahl der Schräglinien durch $\lceil \frac{t}{2} \rceil$ bestimmt, da jede Schräglinie zwei Knoten mit drei Nachbarn besitzt und ein einzelner Knoten mit drei Nachbarn meistens zu einem zweiten solchen Knoten gehört, der allerdings durch Störeinflüsse im Bild nicht enthalten ist. Abb. 33 zeigt eine solche Sperrfläche, die einen Fehler aufweist, aber dennoch erkannt wurde.

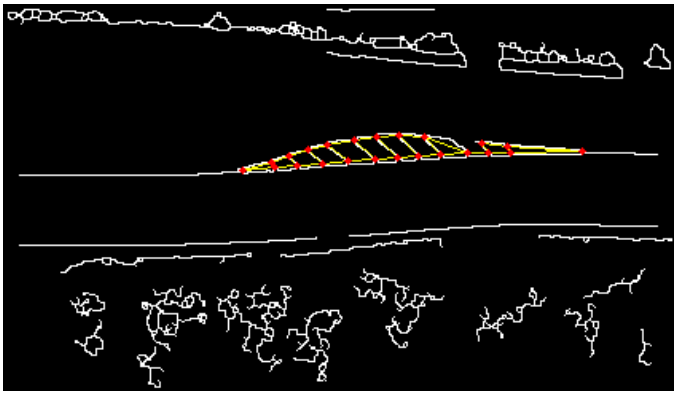

Abbildung 33: Durch Bewertungsfunktion erkannte Sperrfläche, die einen Fehler aufweist

Kapitel D

Auswertung und Ausblick

I Bewertung der Ergebnisse

Untersucht wurden zwölf Datensets mit und ohne Sperrflächen. Die auftretenden Ereignisse werden in vier Kategorien eingeordnet:

- **True positive:** Es ist eine Sperrfläche im Bild enthalten und sie wird als solche erkannt
- **True negative:** Ein Teilgraph wird nicht als Sperrfläche erkannt und ist auch keine
- **False positive:** Ein Teilgraph wird als Sperrfläche erkannt, obwohl er keine solche ist
- **False negative:** Eine im Bild vorhandene Sperrfläche wird nicht als solche erkannt

Die Auswertung der vorliegenden Daten liefert folgende Ergebnisse:

Methode	True positive	True negative	False positive	False negative	Teilgraphen
Suche nach perfekten Sperrflächen	4	307	8	6	325
Erkennung durch Bewertungsfunktion	7	310	5	3	325

Nach Optimierung der Einstellungen des adaptive Thresholding und der Maske der Lückenschließung entstanden folgende Datenpunkte:

Methode	True positive	True negative	False positive	False negative	Teilgraphen
Suche nach perfekten Sperrflächen	4	424	6	6	430
Erkennung durch Bewertungsfunktion	8	420	0	2	430

Das heißt nach Optimierung der Parameter konnte bei der Erkennung durch die Bewertungsfunktion eine Erkennung von Störgraphen als solche von 100% erzielt werden und 80% der Sperrflächen identifiziert werden, während die Methode nach perfekten Sperrflächen zu suchen noch immer Störgraphen als Sperrflächen identifiziert.

II Mögliche Erweiterungen des Programms

Mögliche Erweiterungen des Programms stellen z.B. von vorne herein ausgeschlossene Sperrflächentypen wie Sperrflächen auf Autobahnen oder innerorts, die sich in Form bzw. der Abmessung deutlich von denen der Landstraßen unterscheiden. Allerdings wäre der vorher vorgestellte Ansatz auch für innerstädtische Sperrflächen nutzbar, da er sich nicht auf Abmessungen stützt, was jedoch aufgrund fehlender Daten nicht bewiesen werden kann. Außerdem wäre eine weitere Möglichkeit das System auf weiche Echtzeitfähigkeit zu erweitern, sodass nach der Messung eine ScanLine an den Algorithmus übergeben werden könnte.

Literaturverzeichnis

[1] W. Burger und M. J. Burge. Digitale Bildverarbeitung. Springer-Verlag, first edition, 2005.

[2] E. R. Davies. Computer and Machine Vision. Academic Press, fourth edition, 2012.

[3] A. K. Jain. Fundamentals of digital image processing. Prentice-Hall International, fourth edition, 1989.

[4] OpenCV Dev Team. Java opencv documentation. http://docs.opencv.org/java/, 2014. Önline; accessed 2-September-2015".

[5] J. Parker. Gray level thresholding in badly illuminated images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(8):813–819, Aug 1991.

[6] Rafael C. Gonzales und Richard E. Woods. Digital Image Processing. Pearson Education, Inc., UpperSaddle River, New Jersey, USA, third edition, 2008.

[7] J. Reislhuber. Graph recognition. Master's thesis, University of Passau, Innstraße 33, Passau, Germany, 2011.

Erklärung selbständiger Arbeit

Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich diese Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, 07. September 2015

Sebastian Reichl