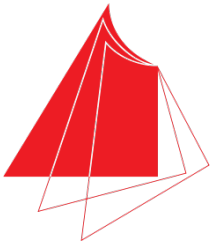


Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Hochschule Karlsruhe - Technik und Wirtschaft

Fahrbahnerkennung mit Matlab

im Fach
Mustererkennung und Bildverarbeitung

Sommersemester 2018

Simon Steinebrunner
MatNr. 62837
stsi1020@hs-karlsruhe.de

Maximilian Zipfl
MatNr. 63048
zima1023@hs-karlsruhe.de

Prof. Dr.-Ing. Christian Langen

3. Juni 2018

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Arbeit

Wir versichern, die von uns vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, haben wir als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die wir für die Arbeit benutzt haben, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum	Simon Steinebrunner
Ort, Datum	Maximilian Zipfl

Kurzfassung

Betreuer:
Prof. Dr.-Ing. Christian Langen

Fahrbahnerkennung mit Matlab

Die vorliegende Studienarbeit gibt einen Überblick über die Theorie und Umsetzung einer Fahrbahnlinienerkennung mit der Software Matlab. Hierfür wird ein Video einer Fahrt mit einem Modellauto aufgenommen. Später sollen die Linien sowohl einzelner Momentaufnahmen, als auch des laufenden Videos erkannt werden. Für die Kantendetektion der Fahrbahnmarkierungen gibt es verschiedene Operatoren, die verwendet werden können. Hierbei stellt sich heraus, dass diese sich im vorliegenden Fall in der Qualität der Kanten kaum unterscheiden. Im nächsten Schritt werden die Kanten mit Hilfe der Hough-Transformation als Gerade erkannt. Da ausschließlich Seiten- und Mittelmarkierung erkannt werden sollen, kann man sich auf Geraden mit einem Winkel von $\pm 60°$ beschränken, was zu einer Minimierung der Rechenzeit führt. Mit dieser Methode ist die Linienerkennung nicht optimal, da die Linien beim Video nicht mittig eingezeichnet werden und es deshalb zu einem Linienzittern führt. Weiterhin werden mehrere Linien gleichzeitig in den Kurvenbereich gelegt. Als Alternative dazu wird zusätzlich auf ein Verfahren eingegangen, bei dem Daten aus einem Plot einer Bildzeile entnommen werden. Deren Maxima werden bestimmt und anschließend einzelne Markierungen in die Mitte der Fahrbahnlinie gelegt. Diese Punkte führen zu einer besseren Liniendetektion ohne zittern und ohne doppelte Erkennung im Kurvenbereich. Spiegelungen auf der Fahrbahn führen zu Fehlerkennungen. Um diese zu eliminieren wird ein Verfahren vorgestellt, welches die Spiegelungen mit der diskreten Ableitung herausfiltert.

Das Ergebnis der Studienarbeit veranschaulicht, dass die Liniendetektion mit den standardisierten Matlab Funktonen Schwächen aufzeigt. Ein besserer Ansatz ist hierbei die Verwendung des Histogrammansatzes, bei dem die Linien klar erkennbar und korrekt detektiert werden.

Abkürzungsverzeichnis

A	Originalbild
a	Steigung
b	y-Achsenabschnitt
D_x	partielle Ableitung in x-Richtung
D_y	partielle Ableitung in y-Richtung
G	Gesamtergebnisbild
g_n	Geraden im Koordinatensystem
G_x	gefaltetes Ergebnisbild nach Tranformation in x-Richtung
G_y	gefaltetes Ergebnisbild nach Tranformation in y-Richtung
P_n	Punkte im Koordinatensystem
Φ	Winkel des Gradienten
$Q(t)$	Quotient aus Varianzen
ρ	Abstand einer Geraden zum Koordinatenursprung
S_1	Schwellenwert 1
S_2	Schwellenwert 2
σ_{in}	Varianz innerhalb eines Segmentes
σ_{zw}	Varianz zwischen den Segmenten
Θ	Winkel der Orthogonalen zur x-Achse

Inhaltsverzeichnis

0.1 Einleitung

Abbildung 1: Modellauto des Studierendenteams C.A.F.Ka

In der Studienarbeit Carolo-Cup wird ein Studententeam von einem fiktiven Fahrzeughersteller beauftragt, anhand eines Modellfahrzeugs im Maßstab 1:10 ein möglichst kostengünstiges und energieeffizientes Gesamtkonzept eines autonomen Fahrzeuges zu entwickeln, herzustellen und zu demonstrieren. Der Hauptsensor des Autos ist eine Kamera. Die erste zu bewältigende Disziplin ist einer Fahrbahn einem Rundkurs zu folgen. Um sich überhaupt auf der Strecke zurechtzufinden müssen die Fahrbahnlinien erkannt und ausgewertet werden.

Die vorliegende Studienarbeit gibt einen Überblick über die Theorie und Umsetzung verschiedener Fahrbahnlinienerkennungen mit der Software Matlab. Hierfür wird ein Video einer Fahrt mit einem Modellauto aufgenommen. Später sollen die Fahrbahnlinien sowohl einzelner Momentaufnahmen, als auch des laufenden Videos erkannt und markiert werden.

Kapitel 1

Liniendetektion mittels Hough-Transformation

1.1 Grundlagen

1.1.1 Kantenerkennung

In der digitalen Bildverarbeitung beruht die Kantenerkennung entlang einer Linie auf dem Unterschied im Farb- oder Grauwert, der Helligkeit oder Textur. Verschiedene Operatoren sollen diese Unterschiede in einem Bild erkennen und damit die Kanten detektieren. Dabei kann es u.a. durch Reflexionen oder schlechter Beleuchtung zu Fehlinterpretationen der Kanten kommen. Hierbei wird der Bereich um jeden Punkt mit einer Faltungsmatrix untersucht. Am leichtesten lassen sich Grauwertänderungen in einem Bild erkennen. Hierfür werden die Ableitungen der Werte in möglichst alle Richtungen genommen. Meistens jedoch genügt die x- und die y-Richtung. Die Ableitung hebt dann die größten Grauwertänderungen, wie sie an Kanten üblich sind, hervor und neutralisiert die Flächen mit konstanten Grauwerten.
Für die Kantenerkennung gibt es verschiedene Operatoren, die sich vor allem in der Faltungsmatrix unterscheiden.

Sobel-Operator

Beim Sobel-Operator wird als Algorithmus die Faltung eingesetzt. Er führt eine Differenzoperation benachbarter Bildpunkte und eine Mittelungsoperation gleichzeitig aus. Weiterhin wird die erste Ableitung des Helligkeitswertes des Bildpunktes berechnet. Das Maximum der Ableitung ist dann der genaue Ort der Kante. Die Matrix des Sobel-Operators ist eine 3x3-Matrix nach der Form:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \cdot A \tag{1.1}$$

und

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \cdot A \tag{1.2}$$

Aus diesen beiden Ergebnissen wird anschließend ein richtungsunabhängiges Gesamtergebnisbild erstellt.

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{1.3}$$

Die Vorteile dieses Filters liegen in der Rauschunterdrückung durch die gewichtete Mittelung. Des weiteren werden die Kanten selbst bei flachem Grauwertübergang hervorgehoben. Ein Nachteil dieser Methode ist die Verbreiterung der Kanten und das nur Kanten detektiert werden können, die senkrecht zur Laufrichtung stehen.

Prewitt-Operator

Dieser Kantenerkennungsoperator ist dem Sobel-Operator ähnlich. Bei diesem Filter werden die Grauwerte jedoch nicht zusätzlich zu der aktuellen Gradientenrichtung gewichtet. Wie auch beim Sobel-Operator erhält man ein Gradientenbild in x- und y-Richtung in der Form:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \cdot A \tag{1.4}$$

und

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \cdot A \tag{1.5}$$

Die Kantenstärke wird ebenfalls gleich berechnet:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{1.6}$$

Auch hier liegt ein Vorteil in der Rauschunterdrückung aufgrund der Mittelung. [wik17c]

Canny-Algorithmus

Der Canny-Algorithmus besteht aus mehreren Phasen und arbeitet mit einem Schwarz-Weiß-Bild. Zuerst wird das Bild mit einem Gauß-Filter geglättet, um Rauschen zu unterdrücken. Der nächste Schritt ist die Kantendetektion. Hierfür wird der Sobel-Operator, wie vorher gezeigt, in x- und y-Richtung berechnet und anschließend das Kantenbild mit

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{1.7}$$

bestimmt.
Nun wird für jedes Pixel die Gradientenrichtung mit der Formel

$$\Phi(x, y) = arctan\left(\frac{D_y(x, y)}{D_x(x, y)}\right) \tag{1.8}$$

berechnet und die Kantenrichtungen werden waagerecht, senkrecht oder diagonal gerundet (0°, 45°, 90°, 135°).

Danach folgt die Unterdrückung von Schein-Maxima. Hierbei werden die beiden Nachbarpixel in Gradientenrichtung betrachtet. Besitzt eines der Nachbarpixel einen größeren Pixelwert, wird der Grauwert auf Null gesetzt. Ist dies nicht der Fall, wird der Pixelwert beibehalten. Die Kanten werden also ausgedünnt, um sie genau zu lokalisieren. [wik17b]

Als letztes wird definiert, ab welchem Wert der Pixel zu einer Kante gehört oder nicht. Das sogenannte Hystereseverfahren definiert zwei Schwellwerte $S_1 < S_2$, wobei man das Bild scannt und alle Pixel mit Werten $> S_2$ als Kantenpixel deklariert und die angrenzenden Kanten verfolgt, solange die Pixelwerte $\geq S_1$ sind. Diese zusammenhängenden Pixel werden dann als Kanten ausgegeben.

Vergleich der Filterarten

Um einen direkten Vergleich der drei Algorithmen zu erhalten, werden diese auf dasselbe Bild angewandt. Hierfür wird das gewählte Ausgangsbild (Abbildung 1.4) zunächst in ein Binärbild gewandelt. Um die störenden Reflexionen zu eliminieren beträgt der Schwellenwert bei der Umwandlung, wie zuvor empirisch ermittelt, 220 (vergleiche Abbildung 1.6b). Auf dieses Schwarz-Weiß-Bild werden nun die Algorithmen Prewitt (Abbildung 1.2), Sobel (Abbildung 1.1) und Canny (Abbildung 1.3) angewandt. Die beiden erstgenannten Algorithmen, Prewitt und Sobel, unterscheiden sich kaum bis gar nicht im Ergebnisbild. Die Kanten werden gut erkannt. An einzelnen Stellen sind die Kanten allerdings leicht unterbrochen. Dies ist beim Canny-Algorithmus nicht der

Fall. Hier sind die Linien der Kanten ohne Unterbrechung durchgezogen, was einen flüssigeren Verlauf darstellt. Für die Kantenerkennung sind jedoch alle drei Verfahren geeignet, da der Unterschied nur minimal ist. Da der Canny-Algorithmus einen höheren Rechenaufwand mit sich führt, sollte entweder der Prewitt- oder der Sobel-Algorithmus bevorzugt werden.

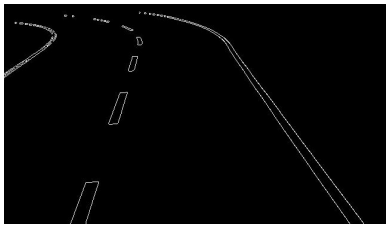


Abbildung 1.1: Sobel

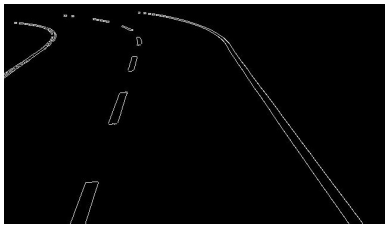


Abbildung 1.2: Prewitt

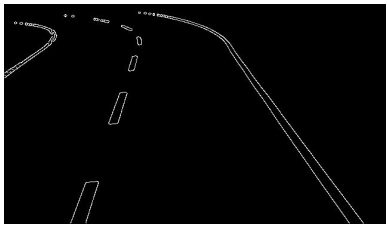


Abbildung 1.3: Canny

[wik17a]

1.1.2 Otsu-Verfahren

Um die Kanten detektieren zu können, muss das Grauwert-Bild in ein Binärbild (Schwarz-Weiß-Bild) umgewandelt werden. Somit sollen Kanten deutlicher hervorgehoben werden. Um den korrekten Grauwert zu erhalten, ab dem sich die Pixel in ihrer Helligkeit unterscheiden, gibt es Schwellenwertverfahren. Diese teilen die verschiedenen Grauwerte in einzelne Segmente. Im vorliegenden Fall sind es genau zwei Segmente. Zum einen die gesuchten Liniensegmente, zum anderen die Umgebungssegmente (Fahrbahn und Hintergrund). Der Vorteil an dem auszuwertenden Bild ist, dass die zu detektierenden Fahrbahnmarkierungen sich unter guten Vorraussetzungen optisch deutlich von der eigentlichen Fahrbahn hervorheben. Daher sollen später nur die weißen Fahrbahnlinien auch weiß im Ergebnisbild dargestellt werden.

Um diesen Schwellenwert zu bestimmen gibt, es das OTSU-Verfahren. Dieses ist ein globales Verfahren und ermittelt den Schwellenwert mittels Varianzen. Es werden die kleinsten Varianzen σ_{in}^2 innerhalb der jeweiligen zwei Segmenten bestimmt. Anschließend berechnet dieses Verfahren die größte Varianz σ_{zw}^2 zwischen den beiden Segmenten und bildet folgenden Quotienten:

$$Q(t) = \frac{\sigma_{zw}^2(t)}{\sigma_{in}^2(t)} \tag{1.9}$$

Der Schwellenwert wird nun so gewählt, dass der Quotient $Q(t)$ maximal wird.

Im vorliegenden Fall ergibt sich das Problem, dass über das gesamte Bild ein großer Helligkeitsverlauf vorliegt (siehe Abbildung 1.4).

Abbildung 1.4: Kamerabild der Fahrbahn

Das Histogramm ist nun, wie in Abbildung 1.5 zu sehen, nicht mehr bimodal. Anstatt zwei eindeutigen Maxima enthält es viele kleine lokale Maxima. Das Otsu-Verfahren legt hier den Schwellenwert bei 112 fest.

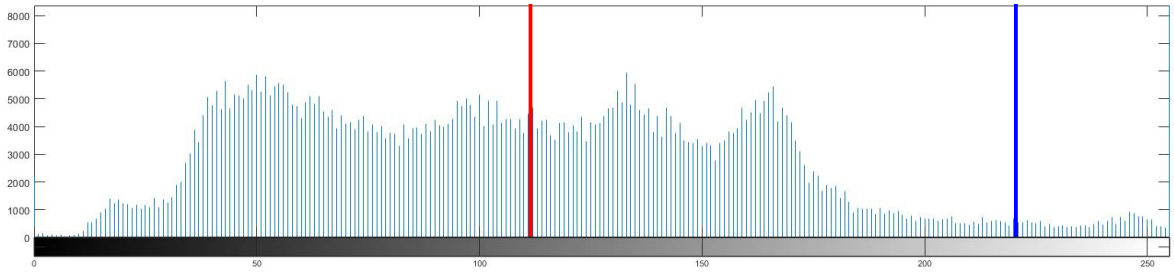


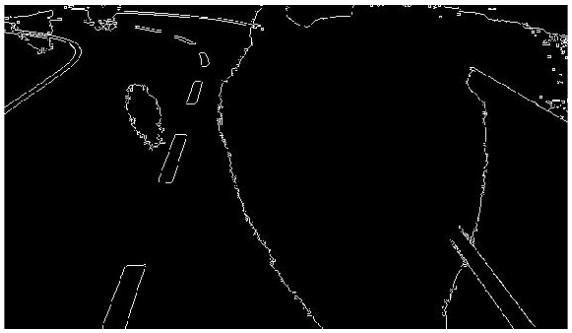
Abbildung 1.5: Histogramm der Helligkeitswerte der in Abbildung 1.4 zusehenden Fahrbahn. Die rote Linie kennzeichnet den, durch das Otsu-Verfahren ermittelte Schwellenwert; die blaue Linie, den durch Erfahrungswerte ermittelten Wert.

Dieser Schwellenwert genügt aber nicht den Anforderungen für spätere Detektion der Fahrbahnmarkierung. Wie in Abbildung 1.6a zu erkennen ist, werden einzelne Spiegelungen ebenfalls als Kante erkannt und führt zu einer Fehlinterpretation der Linienerkennung. Die Kantendetektion arbeitet bei diesem Beispiel mit dem Prewitt-Operator. Um ausschließlich Linien der Fahrbahnmarkierungen zu detektieren, wird der Schwellenwert stückweise höher gesetzt, bis ein zufriedenstellendes Ergebnis erzielt wird. Nach mehreren Testläufen ergibt die Graustufe 220 das beste Ergebnis. Beim Verwenden dieses Wertes entsteht eine verbesserte Kantendetektion ohne Einfluss der Spiegelungen (Abbildung 2).
[Klo06]

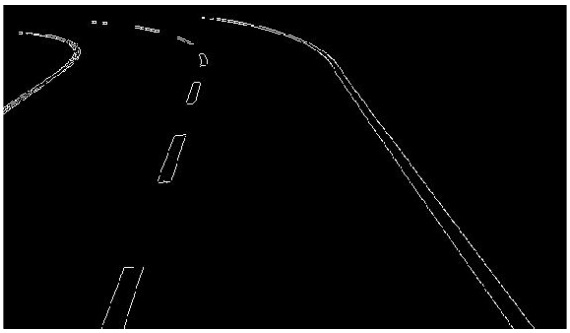
1.1.3 Hough-Transformation

Die Hough-Transformtaion kann in einem Schwarz-Weiß-Bild Geraden und andere geometrische Formen erkennen.
Eine Gerade kann in einem zweidimensionalen, kartesischem Koordinatensysten folgendermaßen beschrieben werden:

$$y = ax + b \tag{1.10}$$



(a) Ergebnisbild mit der Schwellenwertfunktion des Otsu-Verfahren (112)



(b) Ergebnisbild nach Einstellen des empiririschen Schwellenwertes (220)

Abbildung 1.6: Vergleich der binären Ergebnisbilder der Umrechnung in Schwarz-Weiß-Bilder mit verschiedenen Schwellenwerten

Mit dieser Gleichung (1.10) können fast alle Geraden in dem Koordinatensystem dargestellt werden. Die Ausnahmen dabei sind alle Geraden, welche paralell zu der y-Achse verlaufen. Diese hätten die Steigung a von ∞ (unendlich) und können dehalb nicht dargestellt werden. Um auch diese Ausnahmen darstellen zu können, behilft man sich mit einem Trick; das Koordinatensystem wird um 90° rotiert und es werden alle fehlenden Geraden mit dieser neuen Basis betrachtet. Um den erhöhten Rechenaufwand zu minimieren, kann man Geraden auch in der *Hesseschen Normalform* darstellen:

$$\rho = x \cdot cos(\Theta) + y \cdot sin(\Theta) \tag{1.11}$$

Hierbei können alle Geraden in einer Ebene durch den Winkel Θ der Orthogonalen (zur x-Achse) und dem Abstand ρ zur Geraden zum Koordinatenursprung dargestellt werden.

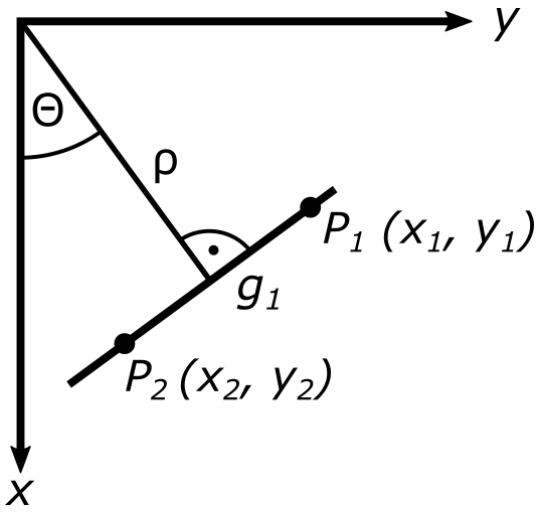


Abbildung 1.7: Karthesisches Koordinatensystem mit der Beschreibung der Geraden g_1 durch die Hessesche Normalform

Durch die in der Abbildung 1.7 veranschaulichte Darstellung der Hesseschen Normalform sind alle Punkte (in diesem Beispiel P_1 und P_2) auf der Geraden g_1 und deren

dazugehörigen Koordinaten ($x_{1,2},\ y_{1,2}$) festgelegt.

Durch jeden Pixel in dem zu analysierendem Bild verlaufen unendlich viele Geraden mit denen dazugehörigen ρ und Θ. In Abbildung 1.8 sind beispielhaft die Geraden g_1, g_2 und g_3, welche zu dem Punkt (Pixel) P_1 gehören, in das Koordinatensystem eingezeichnet. Für jeden Pixel in dem Bild werden alle dazugehörigen Geraden durch ρ und Θ beschrieben und in einem Array abgespeichert. Jeder Punkt auf dem Bild „wählt“ für die Geraden, die durch ihn verlaufen.

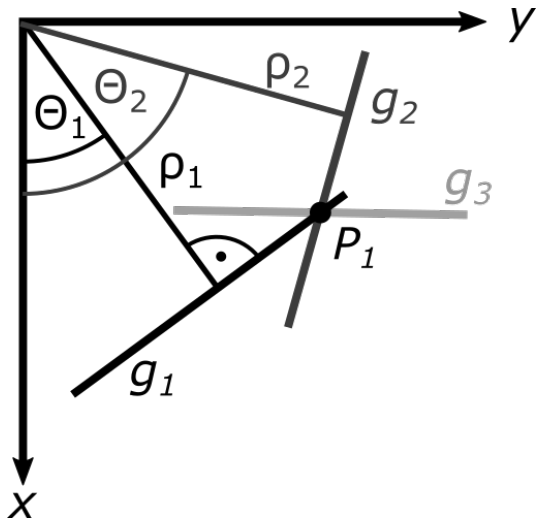


Abbildung 1.8: $g_{1,2,3}$ sind Beispiele für Geraden durch den Punkt P_1.

Betrachten wir nun den Punkt P_2 in Abbildung 1.7. Durch ihn verlaufen ebenso wie bei Punkt P_1 unendlich viele Geraden, allerding auch die Gerade g_1. Dadurch erhält diese Gerade in dem sogenannen „Voting-Array“ schon zwei Stimmen. Natürlich „voten“ alle Pixel, die sich in dem Bild auf der Gerade befinden, auch für diese Gerade. Deshalb erhofft man sich, dass die Gerade, die man in dem Bild detektieren möchte, auch die meisten Wahlstimmen erhält.
Theoretische sollen alle Geraden betrachtet werden, die durch einen Punkt verlaufen. Da aber dieses Problem mithilfe eines Computers gelöst werden soll, betrachetet man eine diskretisierte Anzahl an Geraden. Man könnte zum Beispiel die Winkel der Geraden in 1°-Schritten steigend iterieren.
Das ursprüngliche Koordinatensystem, dass sich aus der x- und der y-Achse zusammensetzt, wird in einen anderen diskreten Modellraum transformiert. Die Koordinaten dieses Raums (Akkumulator) sind ρ, Θ und die Anzahl der dazugehörigen Wahlstimmen.
Wie man aus der Formel der Hesseschen Normalform (vgl. Gleichung 1.11) vermuten lässt, verlaufen die Funktionen der Geraden eines Pixel in einer Sinusfunktion. In Abbildung 1.9 ist modellhaft eine Gerade und zwei daraufliegende Punkte gezeigt. Beide

Funktionen haben einen gemeinsamen Schnittpunkt. Dieser Punkt (ρ', Θ') beschreibt die Gerade g_1, auf der die beiden Punkte $P_{1,2}$ liegen.

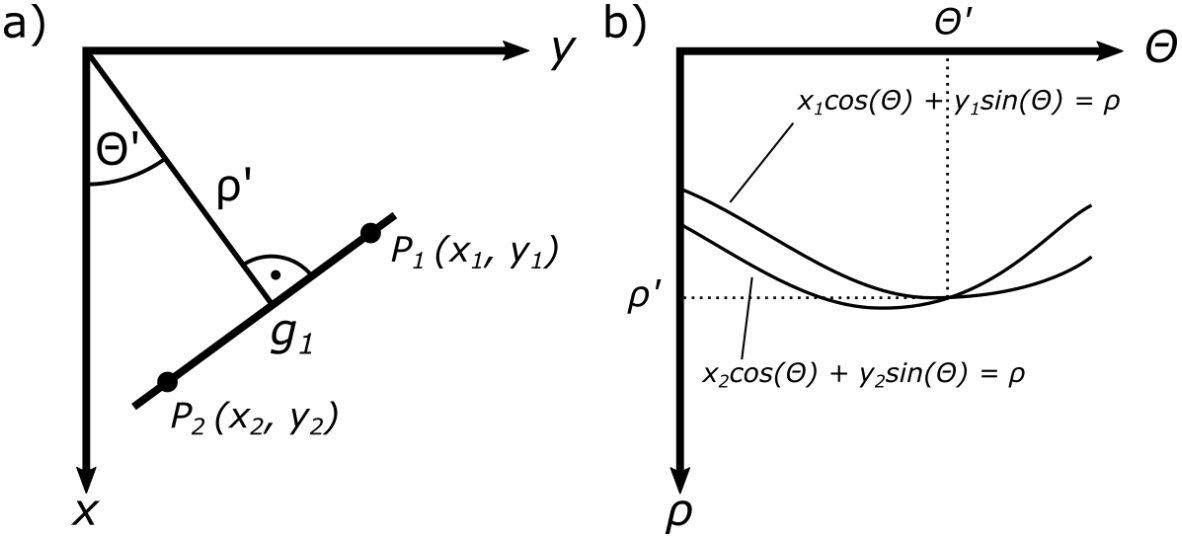


Abbildung 1.9: a) Darstellung der Geraden g_1 und zwei daraufliegender Punkte $P_{1,2}$ in dem herkömmlichen Koordinatensystem. b) Abbildung aller Geraden der Punkte P_1 und P_2 im als Hesseschen Normalform in ein Array (ρ, Θ) eingetragen.

Der nächste Schritt ist dieser, dass man nun das gesamte Bild, also jeden zu betrachtenden Pixel und alle dazugehörigen Geraden in den Modellraum transformiert. Anschließend sucht man das Maximum, also den Punkt, an dem sich die meisten Sinusfunktionen schneiden. Dieser beschreibt dann die Strecke in dem Bild, auf der die meisten Pixel liegen. Natürlich lassen sich auch das zweite bis n-te Maximum bestimmen, um mehrere Geraden zu identifizieren.

Wendet man nun die beschriebenen Algorithmen auf ein reales Bild an, erhält man das in Abbildung 1.10 zu sehende Ergebnis. Die Maxima in Abbildung 1.10.2, sind

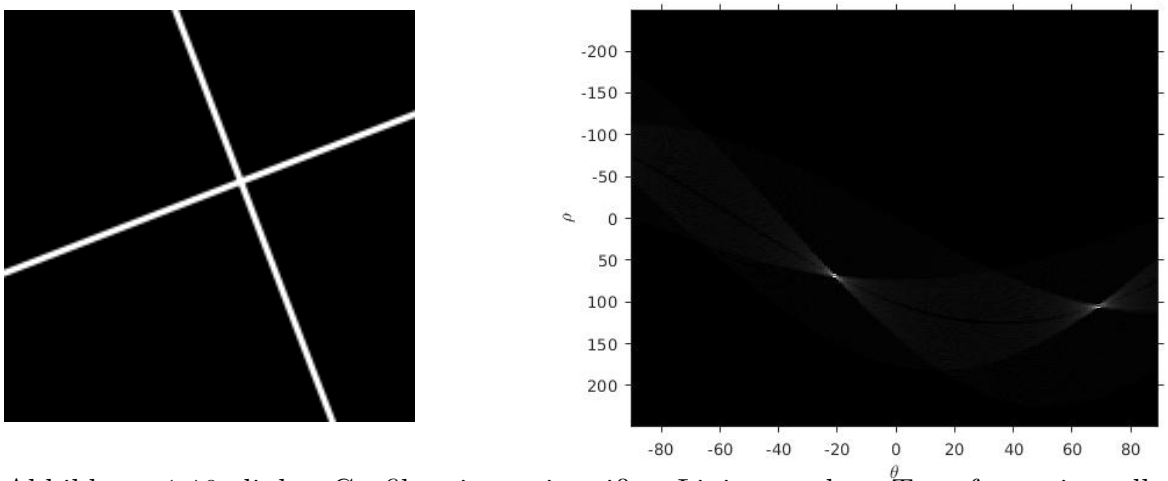


Abbildung 1.10: links: Grafik mit zwei weißen Linien. rechts: Transformation aller weißen Pixel des linken Bilds in die Voting-Matix.

durch eine intensivere Färbung der Pixel verbildlicht. Die Maxima bei (ρ 70, Θ -20)

und (ρ 100, Θ 70) stellen die beiden Geraden des Bildes in Abbildung 1.10.1 dar. Bei diesem Beispiel ist allerding zu beachten, dass der Koordinatenursprung nicht, wie bei Abbildung 1.7 in der oberen linken Ecke, sondern, dass die Koordinaten unten links entspringen. Dies spielt eigentlich keine Rolle, sondern muss nur für die Hin- und Rücktransformation gleich betrachtet werden. [wik18] [Ann09]

1.2 Implementierung

Das Programm ist in verschiedene Teilschritte unterteilt. Zu beginn wird die Videodatei in Matlab geladen. Anschließend werden Variablen deklariert. Dafür wird die Frameanzahl des Videos bestimmt und damit eine vierdimensionale Matrix für alle ausgelesenen Frames bereitgestellt.

```
1 obj = VideoReader('video.mp4'); %video.mp4 -> Kameramitschnitt
2 thresh = 220;
3 nframes = get(obj, 'NumberOfFrames');
4 bildarray = zeros(368,640,3,round(nframes),'uint32');
```

Folgende Filter und andere Algorithmen beziehen sich immer auf ein einzelnes Frame. Am Schluss werden alle bearbeiteten Frames wieder in eine Videodatei zusammengefügt. Aus Performancegründen ist es möglich nicht jedes Frame zu betrachten, sondern es werden eine gewisse Anzahl an Bildern übersprungen. Dies ist so lange möglich, solange die zeitlich aufgelöste Änderung der Fahrbahnmarkierungen nur sehr gering sind. Bei hohen Geschwindigkeiten und gleichbleibender Fahrbahnkurvigkeit müssen gegebenenfalls in einem Zeitintervall mehr Frames betrachetet werden.

```
5 skip = 4; % in diesem Fall wird nur jeder vierte Frame betrachtet
6 for k = 1 : nframes/skip
7         singleFrame = read(obj, k*skip)
8         ...
20 end
```

Das Videobild besitzt noch Farbinformationen[1], diese sind aber für die folgenden Betrachtungen uninteressant. Deshalb wird die Farbinformation (h) und die Sättigung (s) des Bildes mit Hilfe von `rgb2gray()` verworfen. Nur die Helligkeitsinformation (v) bleibt erhalten. Anschließend wird mit dem Befehl `im2bw()` aus dem Graustufenbild ein Schwarz-Weiß-Bild generiert. Die Funktion verwendet das Schwellenwertverfahren. Alle Helligkeitswerte, die einen vorgegebenen Wert unterschreiten, werden auf 0 (schwarz) gesetzt. Liegt ein Helligkeitswert eines Pixels darüber, wird er auf 255 (weiß) gesetzt (vergleiche Abbildung 1.6b).

```
8         bild_sw = rgb2gray(singleFrame);
9         bild_thresh = im2bw(bild_sw,thresh/255);
```

[1] Wird mit dem HSV-Farbraum beschrieben

Das binäre Bild kann nun mit dem Kantenoperator verrechnet werden. Hierbei soll der „Canny-Operator“ angewendet werden.
Die in Abschnitt 1.1.3 beschriebene Hough-Transformation wird im folgenden Schritt auf das auf Kanten beschränkte Bild angewandt. Die Funktion `hough()` liefert eine zweidimensionale Matrix und zwei Arrays zurück. Der wichtigste Rückgabeparameter ist die „Voting-Matrix“ H. Die beiden anderen Rückgabewerte sind für die Dimensionierung der Matrix: Θ T und ρ R. Um die Rechenzeit zu minimieren, werden einzelne Parameter der Hough-Transformation an die Anwendung angepasst. Da mit diesem Programm ausschließlich die Seitenmarkierung und die Mittelmarkung detektiert werden soll, kann man sich auf Geraden mit einem Winkel von $\pm 60°$ beschränken. Die Diskretisierung des Winkels wird auf 1° angesetzt. Eine genauere Diskussion hierzu wurde nicht durchgeführt[2]. Der Resultierende Hough-Raum wird in Abbildung 1.11 dargestellt. Hierbei sind deutlich die drei Fahrbahnmarkierungen als Maximaschar in dem Schaubild erkennbar. Außerdem werden nicht nur drei große Maxima erkannt, sondern mehrere Maxima die sich nur leicht in den Parametern unterscheiden. Das führt dazu, dass eine Fahrbahnmarkierung öffters mit zwei Linien bezeichnet wird.

```
10          BW = edge(bild_thresh,'Canny',0.1);
11          [H,T,R] = hough(BW,'Theta',-60:1:60);
```

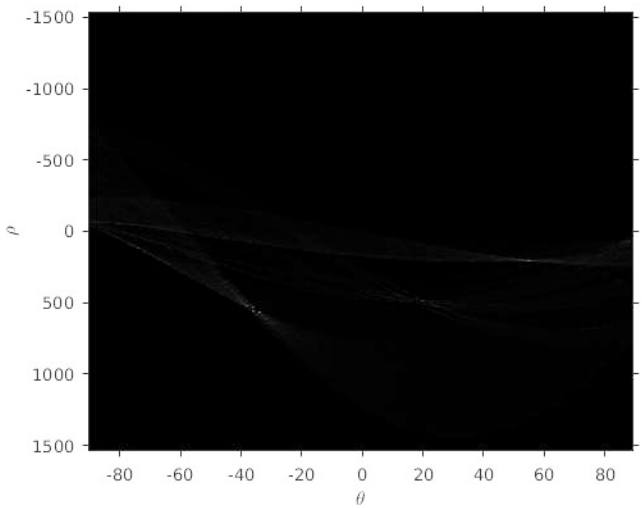


Abbildung 1.11: Houghraum des Bilds des Fahrbahn

Sobald die Voting-Matrix aufgestellt ist, werden die fünf größten Werte mit den dazugehörigen Koordinaten (Θ, ρ) bestimmt. Die Maximumfunktion verhindert außerdem noch, dass zu kleine Werte berücksichtigt werden, falls keine Linien im Originalbild detektiert werden. Generell wäre es sinnvoll nur drei Linien aus dem Hough-Raum zu extrahieren,

[2]Bei Versuchen mit Winkel zwischen 0,5° und 5° konnten Verbesserungen der Rechenzeiten von 10% erzielt werden.

da nur höchsten zwei Außen- und eine Mittelmarkierung in dem Bild liegen sollten. Breite Linien werden teilweise als Linienpaar erkannt. Außerdem werden Kurven als Aneinanderreihung vieler Teilsegmente erkannt. Durch eine höhere Anzahl der Maxima stellt man sicher, dass die gewünschten Linien bei der Maxima-Schar dabei sind. Sind die Extremwerte der Matrix gefunden, können aus den dazugehörigen Koordinaten die Geraden gezeichnet werden. Da durch die Hough-Transformation nur Geraden, also Linien ohne Anfang und Enden generiert werden, würden diese das resultierende Bild sehr unübersichtlich machen. Die Funktion `houghlines()` betrachtet deshalb auch noch das Binärbild. Hierbei werden die Geraden über das Bild gelegt. Im Folgeschritt werden alle Pixel, die bereits den Wert 255 besitzen, eingefärbt. Somit enstehen viele kurze Strecken. Die Funktion erlaubt außerdem noch, dass kleine Lücken zwischen weißen Pixeln aufgefüllt werden können.

```
12          P  = houghpeaks(H,5,'threshold',ceil(0.2*max(H(:))));
13          lines = houghlines(BW,T,R,P,'FillGap',40,'MinLength',60);
```

Der nachfolgende Code zeichnet die durch die oben genannten Algorithmen berechneten Geraden in das Ausgangsbild ein. Ein Großteil der Rechenkapazität wird für die grafische Darstellung beansprucht (siehe Abbildung 1.12).

```
14          imshow(singleFrame),hold on;
15          for k = 1:length(lines)
16                  xy = [lines(k).point1; lines(k).point2];
17                  plot(xy(:,1),xy(:,2),'LineWidth',4,'Color','blue');
18          end
19          F = getframe;
```

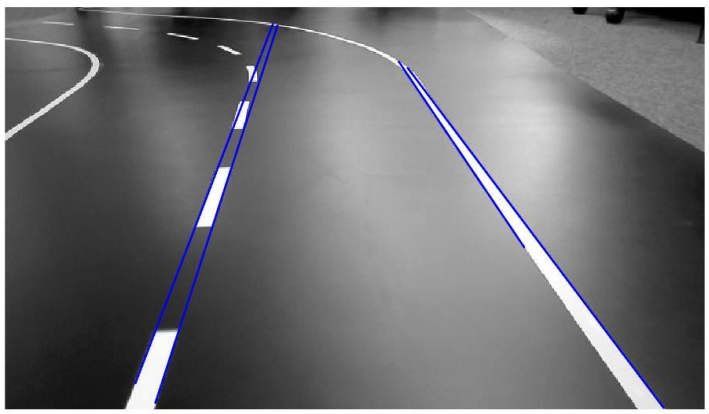


Abbildung 1.12: Ausgangsbild mit eingezeichneten Linien

Wie oben bereits beschrieben, werden die Fahrbahnmarkierungen mit zwei Linien gekennzeichnet. Zudem wird die linke Fahrbahnbegrenzung überhaupt nicht erkannt.

1.3 Fazit der Hough-Transformation

Mit Hilfe der Hough-Transformation werden alle Linien in einem Bild erkannt. Da für die Fahrbahnauswertung aber nur bestimmte Linien (Fahrbahnmarkierungen) erkannt werden sollen, muss der Algorithmus erweitert werden. Hierbei soll er sich nur noch auf eine gewisse Anzahl an Maxima (ausgeprägteste Linien) beschränken. Wie in Abbildung 1.12 werden teilweise Linien übersehen und besonders deutliche[3] Fahrbahnmarkierungen doppelt (linke und rechte Kante) nachgezeichnet. Für die Fahrbahnerkennung genügt eine einfache Beschreibung aller drei Fahrbahnmarkierungen durch eine Gerade.
Besonders störanfällig arbeitet der Algorithmus bei Kurven mit kleinem Kurvenradius (siehe Abbildung 1.13. Aufgrund der begrenzten Anzahl der berücksichtigten Maxima wird in einigen Fällen nur eine kurvige Fahrbahnlinie durch mehrere kurze Strecken erkannt. Ein möglicher Lösungsansatz wäre es, den Bereich zu beschränken auf den dieser Algorithmus angewendet werden soll. Dabei soll nur der untere Bereich des Bilds betrachtet werden, also die Fahrbahn in nächster Nähe zum Fahrzeug. Aufgrund der unhandlichen Implementierung wird dieser Ansatz nicht weiter beleuchtet.

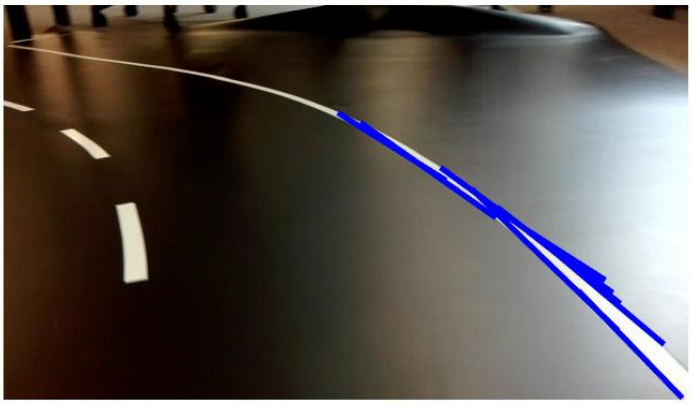


Abbildung 1.13: Fehlerhafte Liniendetektion bei einer Kurve

Um die beschriebenen Probleme zu umgehen, wird zusätzlich ein neuer Ansatz für die Fahrbahndetektierung diskutiert. [mat]

[3] mit einem hohen Kontrast

Kapitel 2

Fahrbahndetektion mit Histogrammen

2.1 Grundlagen

Folgender Lösungansatz soll die falsch-positiv Detektierungen der Spiegelungen und die Ansammlung vieler Linien bei Kurven beheben. Das Hauptproblem an dem oben diskutierten Lösungsversuch ist, dass die Fahrbahnmarkierungen nur durch eine sehr begrenzte Anzahl Geraden beschrieben werden müssen. In kurvigen Fahrbahnabschnitten stößt dieser Algorithmus deshalb an seine Grenzen.
Der nachfolgende Algorithmus beschreibt die Fahrbahnmarkierungen durch einzelne Punkte (Markerkreuze). Durch diese Punkte wird gewissermaßen der Kurvenverlauf der Fahrbahnbegrenzungen diskretisiert.
Diese Punkte werden mit Hilfe eines Plots der Pixelhelligkeiten und darauf angewendete Funktionen bestimmt.
Die verwendete Auflistung der Pixel ist kein Histogramm im eigentlichen Sinne. Hierbei werden keine Klassen und deren Häufigkeiten aufgelistet, sondern die Helligkeitswerte der Pixel entlang einer horizontalen Pixelreihe betrachtet.

Abbildung 2.1: Fahrbahnbild für die Maximafindung der einzelnen Zeilen. Die beiden roten Linien markieren die untersuchten Pixelzeilen (170 & 340)

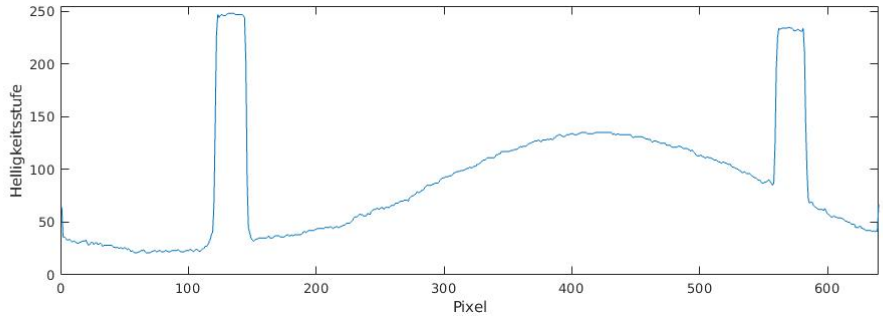


Abbildung 2.2: Plot der Helligkeitsstufen über die Pixel der 340. Pixelreihe des Bildes (untere rote Linie) aus Abbildung 2.1

In dem Graphen aus Abbildung 2.2 erkennt man drei lokale Maxima. Zwei Kurvenberge mit einer Pulsbreite von etwa 30 Pixel und ein sehr breit und langsam ansteigender Berg. Vergleicht man nun diesen Plot mit dem Ausgangsbild, kann man daraus folgern, dass ein schmaler Puls zu einer Fahrbahnmarkierung gehören muss. Das andere Maximum wird folglich durch die Spiegelung auf der Farbahnmitte erzeugt.
Nun soll zwischen den verschiedenen Maximatypen unterschieden werden. Alle Einflüsse, die durch Spiegelungen und durch leichte Farbunterschiede der Fahrbahn verursacht werden, sollen vernachlässigt werden. Solange die Helligkeitswerte der Spiegelungen unter denen der Fahrbahnmarkierung liegen, kann, wie in Kapitel 1.1.2, ein Schwellenwert auf das ganze Bild angewendet werden. In diesem Beispiel würde ein Schwellenwert von 150 bereits ausreichen um nur die Fahrbahnmarkierungen darzustellen.
Betrachtet man nun aber den Plot der 170. Zeile ist eine eindeutige Unterscheidung mittels Schwellenwert zwischen Fahrbahnmarkierung und Spiegelung nicht möglich (siehe Abbildung 2.3).

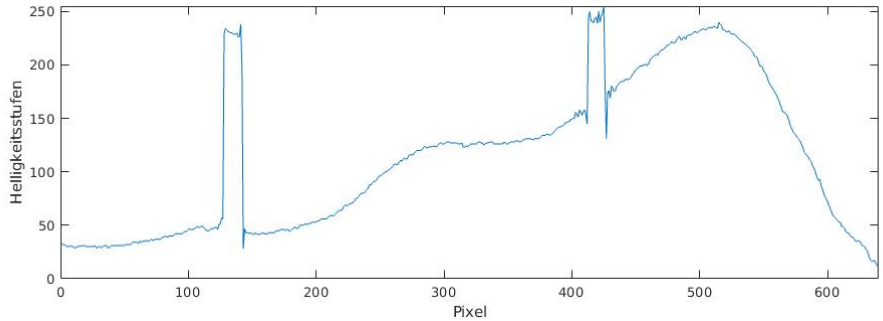


Abbildung 2.3: Plot der Pixelzeile mit deutlichem Einfluss der Spiegelung

Ziel des Programms soll es sein, dass die Maxima mit den zugehörigen Koordinaten bestimmt werden uund anschließend mithilfe von Markern in das Ausgansbild eingetragen werden.

2.2 Implementierung

Am Anfang muss das zu analysierende Video geladen werden. Anschließend können einzelne Frames extrahiert werden, auf die der Algorithmus angewendet werden kann. Ähnlich wie in 1.2 werden immer 4 Frames übersprungen um die Performance des Programmes zu steigern. Außerdem werden anfangs noch Variablen definiert um bei späteren Variationen diese einfach ändern zu können.

```
1  obj = VideoReader('video.mp4');
2  thresh = 220;
3  skip = 4;
4  nframes = get(obj,'NumberOfFrames')
```

Folgender Code ist für die Umformung jedes geladenen Frames in das passende Format: Hierbei muss aus einem farbigen Bild ein Graustufenbild erzeugt werden. Auf dieses kann man anschließend ein Schwellenwertverfahren anwenden damit es in ein Schwarz-Weiß-Bild umgewandelt wird. Der Schwellenwert wird ähnlich, wie bei dem oben beschriebenen Verfahren, empirisch ermittelt und beruht auf keinem mathematischen Verfahren. Das Variablen-Array „marker_pos“ ist ein Platzhalter für die später eingetragenen Markerkoordinaten.

```
5  for q = 1 : nframes/skip
6      singleFrame = read(obj, q*skip);
7      bild_sw = rgb2gray(singleFrame);
8      bild_thresh = im2bw(bild_sw,thresh/255);
9      marker_pos = [0 0]
10         ...
24 end
```

Die wenigen, folgenden Codezeilen spiegeln im Prinzip den ganzen Liniendetektion-Algorithmus wider.
Aufgrund der Übersichtlichkeit und aufgrund von einer Performanceverbesserung wird der Algorithmus nur auf jede zehnte Bildzeile angewandt. Die Werte (0/1) einer solchen Bildzeile werden in ein Array abgespeichert, um damit besser arbeiten zu können. Anschließend werden die Maxima bestimmt. Es handelt sich hierbei um einen Impuls mehrere Impulse (vergleiche Abbildung 2.6. Daher wird in der Funktion `islocalmax` das Maximum als der Punkt definiert, der sich genau in der Mitte zwischen den beiden Flanken eines Impulses befindet. Das Maximum wird als 1 übertragen und alle anderen Werte zu 0 gemacht. Im Anschluss muss die Pixelkoordinate zu diesem Peak in das Markerarray eingetragen werden.

```
25         for j = 1:10:361
26                 bildzeile = bild_thresh(j,:);
27                 TF = islocalmax(bildzeile,'MinSeparation',30);
28                 [col] = find(ismember(TF, 1));
29                 col_1 = col'
30                 col_1(:,2) = j;
```

```
31                  marker_pos = [marker_pos;col_1];
32          end
```

Am Ende werden alle gespeicherten Koordinaten mit der in Matlab enhaltenen Funktion (`insertMarker`) in das Ausgangsbild eingetragen (Abbildung 2.4).

```
33          RGB = insertMarker(bild_sw,marker_pos,'color','blue','Size',5);
34          imshow(RGB);
35          F = getframe;
```

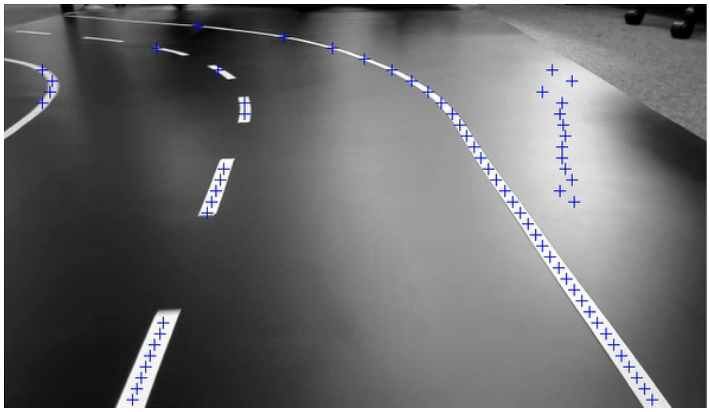


Abbildung 2.4: Fahrbahnmarkierungen mit Markern (Kreuzen) mit Hilfe eines Algorithmus eingetragen

2.3 Verbesserungen des Algorithmus

2.3.1 Minimierung der Falscherkennung durch Spiegelungen

Als Versuchsbild wird die Pixelzeile 170 verwendet. Wie der Plot aus Abbildung 2.3 anschaulich zeigt, führt die Spiegelung am rechten Fahrbahnrand zu einer Fehlinterpretation und wird fälschlicherweise zum Teil als Linie erkannt. Dies lässt sich auch in der zugehörigen Abbildung 2.5 erkennen. Die Software interpretiert sowohl die Mittellinie, als auch die rechte Fahrbahnmarkierung als Linie. Zusätzlich ergibt die Spiegelung eine weitere „Fahrbahnlinienerkennung“, welche falsch ist. Es zeigt sich deutlich (Abbildung 2.6), dass die Pulsbreite der Spiegelung deutlich höher ist als die der Fahrbahnlinien. Um dem entgegenzuwirken wird eine Methode verwendet, bei der die Breite der Fahrbahnmarkierungen betrachtet wird. Der Treshhold wird hierbei auf den Wert von 220 gesetzt.
Nun wird die Anzahl der Aneinanderreihung der Pixelwerte die auf 1 gesetzt sind, also die weißen Pixel welche die Fahrbahnmarkierungen darstellen sollen, gezählt. Die Pixelanzahl der Linienbreite beträgt 15 – 16 Pixel. Die Pixelanzahl der Spiegelung beträgt 54 und ist somit signifikant höher. Um dies auszunutzen, wird bei einem Pixelwert von 1 ein Zähler gestartet, der die Pixelanzahl solange hochzählt, bis ein darauffolgender Pixelwert 0 ist.

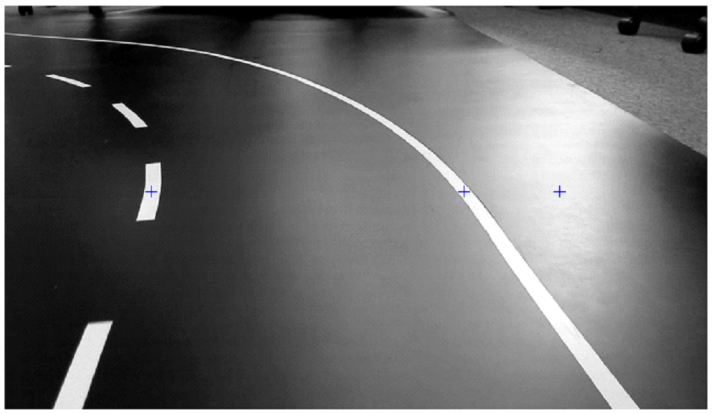


Abbildung 2.5: Einzelnes Frame mit der Codeanawendung auf die Pixelzeile 170 und der Fehlinterpretation aufgrund der Spiegelung

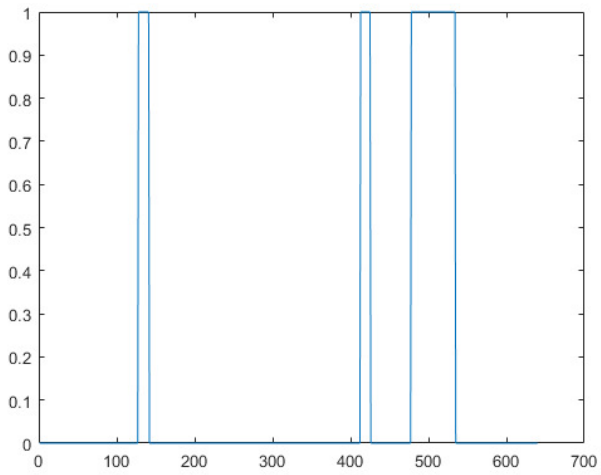


Abbildung 2.6: Zu Abbildung 2.5 zugehöriger Plot der Pixelzeile 170

```
for anzahlPixel = 2:1:640
        pixelWert = q(anzahlPixel);
        if ((q(anzahlPixel) < (q(anzahlPixel-1))))
                anzahlMaxima = count;
                if anzahlMaxima > threshhold_maxima
                        for i = 1:anzahlMaxima
                                q(anzahlPixel-i) = 0;
                        end
                end
        end

        if (pixelWert == 0)
                count = 0;
        else
                count=count+1;
        end
end
```

Um eine sichere Kante zu erkennen, wird der Zähler bei einem Wechsel von einem Pixelwert 1 auf 0 gestoppt und die Pixelanzahl der Werte 1 mit einem Schwellenwert verglichen. Da, wie vorher schon erwähnt, die Pixelanzahl der Linien zwischen 15 – 16 Pixel liegt, ist der Schwellenwert hierbei auf 20 festgelegt. Sollte, wie in der vorliegenden Zeile (Abbildung 2.6 rechter Peak) zu sehen, der Zähler einen größeren Wert als 20 aufweisen, kann es sich folglich nicht um eine Fahrbahnmarkierung handeln. In diesem Fall ist es eine Spiegelung, die zur Fehlinterpretation führt. Das gezählte Array wird dann in einer Schleife auf 0 gesetzt und die Spiegelung somit eliminiert. Das Ergebnis lässt sich anschaulich anhand des darauffolgenden Plots zeigen, bei dem die Spiegelung nicht mehr als Fahrbahnmarkierung erkannt wird.

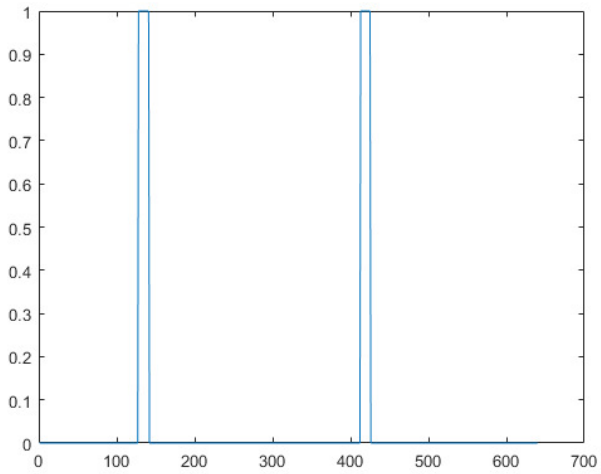


Abbildung 2.7: Plot des verbesserten Codes mit unterdrückten Spiegelungen

Wird dieses Code-Snippet auf das komplette Frame angewendet, ergibt sich nur eine Fehlinterpretation am äußersten rechten Rand der Fahrbahn (siehe Abbildung 2.8.

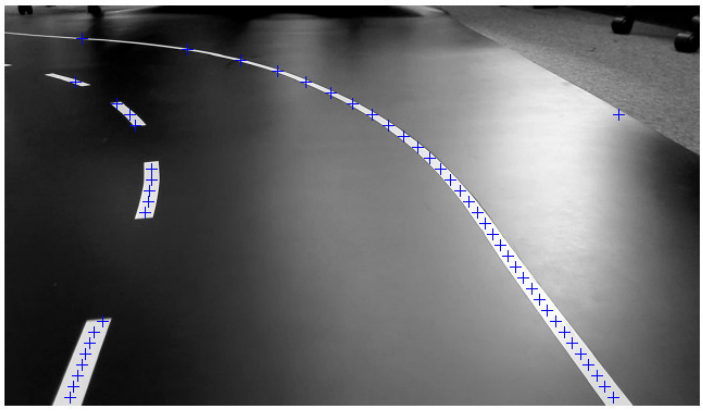


Abbildung 2.8: Bild mit verbesserten Code und unterdrückten Spiegelungen

2.4 Fahrbahndetektion mit diskreter Ableitung

Da die Spiegelung bei der vorherigen Methode teilweise immer noch zu Falscherkennungen führt, wird noch ein weiterer Ansatz verfolgt. Dieser arbeitet mit der diskreten Ableitung. Wird bei einem Grauwertbild eine Zeile aus einem Frame herausgenommen, dann ergibt sich der in Abbildung „Grauwert einer Zeile“ dargestellte Grauwertverlauf.

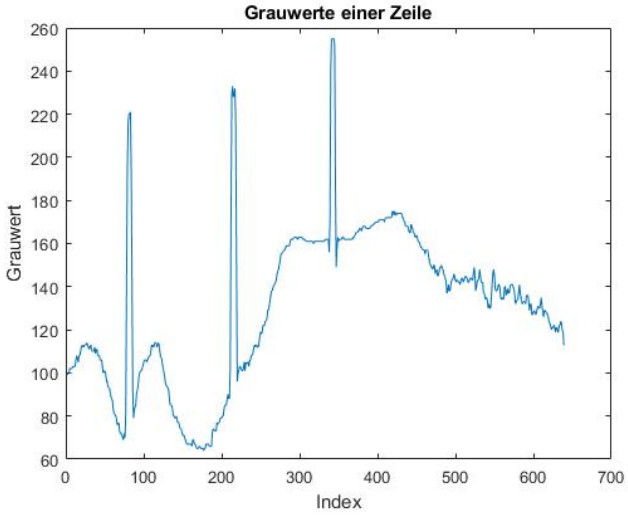


Abbildung 2.9: Grauwertverlauf einer Zeile

Die drei Linien sind deutlich dargestellt. Der Wellenberg bei einer Indexzahl von ca. 420 stellt eine Spiegelung mit leichter Intensität dar. Bei einer anderen Stelle, wie in Abbildung 2.3 zu sehen, ist die Amplitude wesentlich höher und es gibt somit eine falsche Linienerkennung. Um nun die Linie detektieren zu können, wird für jede Bildzeile die diskrete Ableitung gebildet, indem diese um einen Index verschoben wird und anschließend von der ursprünglichen Bildzeile subtrahiert wird.

```
1 bildzeile =bild_sw(j,:);
2 bildzeile =double(bildzeile); %bildzeile in double Wert umgewandelt
3 bildzeile2=bildzeile(2:end)-bildzeile(1:end-1); %die Bildzeile wird um
    eins verschoben und gegeneinander subtrahiert um die diskrete
    Ableitung zu erhalten
```

Das Ergebnis, also die Ableitung, ist in Abbildung 2.10 dargestellt. Die Peaks in positive y-Richtung werden, beim Übergang von links nach rechts kommend, an der Kante von der Umgebung zu einer Fahrbahnmarkierung erzeugt. Stößt man im weiteren Verlauf an das Ende der Fahrbahnmarkierung, ist die Ableitung in negativer y-Richtung. Dies wird dazu genutzt, die Fahrbahnmarkierung mittels der großen Änderung (Ableitung) zu detektieren.
Um weitere falsche Erkennungen zu reduzieren, wird der der negative Peak beim Übergang Fahrbahnlinie zur Umgebung invertiert und verglichen, ob sich die beiden

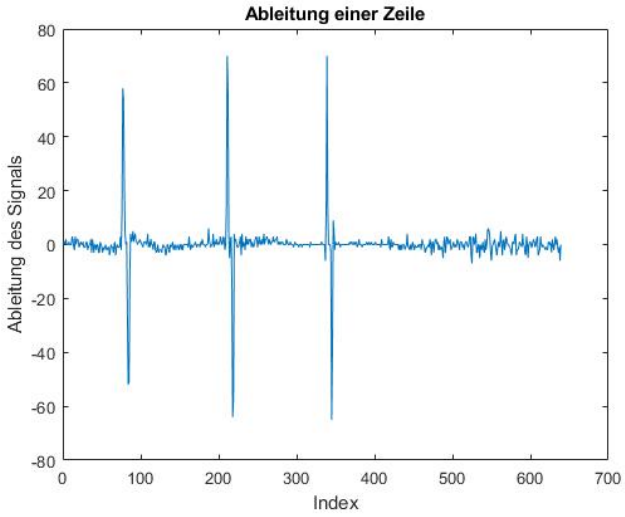


Abbildung 2.10: Ableitung einer Zeile

größten Amplituden (die jeweils die Ränder einer Fahrbahnmarkierung darstellen) innerhalb einer bestimmten Pixelanzahl befinden. Anschließend wird der Treshold niedrig angesetzt um alle Fahrbahnmarkierungen zu detektieren. Aufgrund des niedrigen Tresholds, werden zusätzlich viele Falscherkennungen angezeigt. Diese werden zu einem späteren Zeitpunkt wieder herausgeworfen. Als nächstes wird eine Liste „indexliste“ angelegt, die die Mitte der Fahrbahnmarkierung mit Hilfe der beiden Peaks abspeichert.

```
1  for i=1:length(peak)
2      listpeaks = peak2(find((peak2 > peak(i))));
3      %Peaks innerhalb bestimmtem Abstand
4      listpeaks2 = peak2(find(peak2 < peak(i) + 10));
5      if length(listpeaks) > 0
6          indexliste = round ((peak(i) + listpeaks(1)) / 2); %Mitte der
               Peaks
7          if bildzeile(indexliste) > 212  %filtert Linie > 212 raus
8              ergebnisliste = [ergebnisliste indexliste];  %Linienwerte
                   speichern
9          end
10     end
11 end
```

Um zuletzt die falschen Markierungen aufgrund des zuvor verwendeten niedrigen Tresholds zu beseitigen, wird ein weiterer Treshold eingeführt, der alle gefundenen Punkte löscht, deren Pixelwert kleiner als 212 ist und somit keine Linie darstellen können. Die übrig gebliebenen korrekten Fahrbahnmarkierungen werden in der Matrix „ergebnisliste“ abgespeichert. Diese Liste wird zum Schluss mit den Markern gekoppelt und ausgegeben. Es werden nun nur noch die korrekten Fahrbahnmarkierungen gekennzeichnet. Die Spiegelung und andere Störeffekte sind eliminiert worden.

2.5 Fazit

Das Untersuchen einzelner Pixelzeilen erweist sich, in Hinsicht der fehlerfreien Liniendetektion, als deutlich effektiver. Hierbei werden beinahe alle Fahrbahnmarkierungen erkannt und in das Ausgangsbild mithilfe von Markern eingezeichnen. Betrachetet man nur die Maxima mit diesem Verfahren sind helle Spiegelungen und andere Helligkeitsverläufe auf und neben der Farbahn als Fahrbahnmarkierung erkannt worden. Das Unterscheiden verschiedener „Pulsbreiten “verbessert das Ergebnis deutlich.
Das Auswerten der Steigung der einzelnen Maxima führt noch zu einem sichereren Detektieren der Fahrbahnmarkierungen. Zudem werden bei diesen drei Verfahren keine rechenintensiven Algorithmen verwendet, wodurch die Rechenzeit verkürzt und somit die Performance sichtbar verbessert wird.

Abbildungsverzeichnis

Literaturverzeichnis

[Ann09] Anne, Solberg: Hough transform. (2009), Oct, S. 1–39

[Klo06] Klocke, Heiner: Bildsegmentierung. In: *Algorithmische Anwendungen* (2006), Feb

[mat] *BW*. https://de.mathworks.com/help/images/ref/hough.html

[wik17a] *Canny-Algorithmus*. https://de.wikipedia.org/wiki/Canny-Algorithmus. Version: Dec 2017

[wik17b] *Prewitt-Operator*. https://de.wikipedia.org/wiki/Prewitt-Operator. Version: Dec 2017

[wik17c] *Sobel-Operator*. https://de.wikipedia.org/wiki/Sobel-Operator. Version: Dec 2017

[wik18] *Hough transform*. https://en.wikipedia.org/wiki/Hough_transform. Version: May 2018