Fahrspurerkennung für autonome Fahrzeuge basierend auf OpenCV

Malte Janssen

Januar 2019

Contents

1 Einleitung/ Zielsetzung **3**

2 Schwierigkeiten **4**

3 Algorithmus **6**

4 Parameterstudien **11**

4.1 Canny Edge Detection 11

4.2 Hough Linien Transformation 12

5 Future Work **12**

1 Einleitung/ Zielsetzung

Der Markt für autonome Autos wächst derzeit sehr schnell. Viele Unternehmen arbeiten an dieser komplexen Thematik, die viele Teilaufgaben, wie z. B. Robotik, Navigation, Computer-Vision und Mechanik beinhaltet. Eines der zentralen Bereiche des autonomen Fahrens ist die Computer Vision, denn um sicher in unbekannten Umgebungen autonom fahren zu können, muss ein Fahrzeug seine Umgebung wahrnehmen können. Diese Projektarbeit beschränkt sich auf diesen Bereich mit dem Fokus einen Algorithmus zur Fahrspurerkennung für autonome Fahrzeuge zu entwerfen und entwickeln.
Das Ziel dieser Projektarbeit war, eine Alternative zu dem in der Projektarbeit "Faust" entwickelten Spurenerkennungsalgorithmus für autonome Fahrzeuge, zu entwickeln. Zum Start dieser Projektarbeit hatte das Team des Projekts bereits einen Algorithmus entwickelt, der mit Konturen und einem sliding window Ansatz arbeitet.
Der in dieser Projektarbeit erläuterte Algorithmus arbeitet stattdessen mithilfe von Linensegmenten, die mittels der Hough-Transformation erkannt werden. Das Projekt wurde mithilfe von der Programmiersprache C++ und der OpenCV Bibliothek realisiert.

Figure 1: Beispielstraße in bereits vorhandenen Lösungen - Quelle: http://jokla.me/robotics/lane-detection/

2 Schwierigkeiten

Wenn die Begriffe Opencv, Hough Transformation und Lane Detection im Internet gesucht werden, werden viele Anleitungen und Implementierungen gefunden. Diese Implementierungen sind jedoch deutlich simpler, was diese Projektarbeit rechtfertigt. Im Folgenden werden einige der Probleme, die im Laufe dieser Projektarbeit aufgetreten sind und nicht in simpleren, bereits existierenden Lösungen behandelt werden, aufgezählt.

Kurven - In den im Internet zur Verfügung stehenden Lösungen fahren die Fahrzeuge immer auf nahezu geraden Autobahnen. Wenn überhaupt eine Kurve vorhanden ist, ist diese so sanft, dass sie in Nähe des Fahrzeuges quasi eine Gerade ist. Dies ist für die Hough Lines Transformation, die gerade Linien erkennt, ideal. Ein Beispiel einer solchen Straße kann in Grafik 1 gesehen werden.

Kurven / Region of Interest - Ein weiteres Problem, dass steile Kurven mit sich bringen ist, dass sich nicht starr auf die Strecke geradeaus des Fahrzeugs konzentriert werden kann. In allen Beispielen wird nur die momentane Spur des Fahrzeugs beachtet und eine Region of Interest angewandt, um alles außerhalb zu ignorieren. In Grafiken 2, 3 wird deutlich, wie eine mehrspurige gerade Straße mittels einer Region auf Interest auf die aktuelle

Figure 2: Bild einer geraden Straße - Quelle: http://jokla.me/robotics/lane-detection/

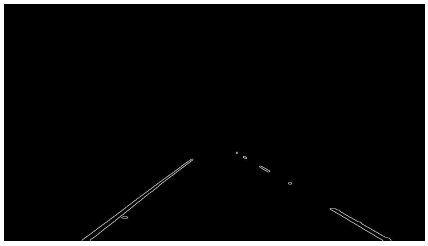

Figure 3: ROI einer einzigen Spur - Quelle: http://jokla.me/robotics/lane-detection/

Spur reduziert wird. Mittels der Berechnung der Steigung der Linien kann nun ohne Probleme zwischen der linken und rechten Fahrspurbegrenzung differenziert werden. Dies ist auf der Teststrecke mit steilen kurven nicht möglich. Hier müssen alle Liniensegmente erkannt und zur richtigen Linie gruppiert werden.

3 Algorithmus

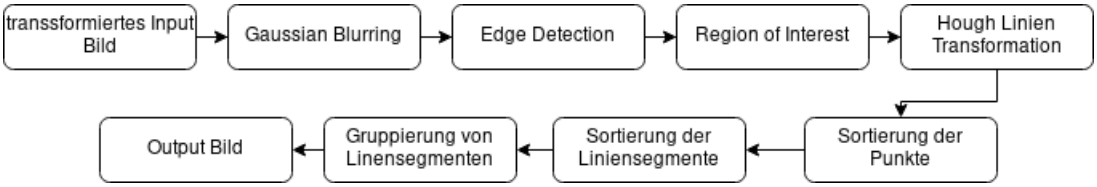

Figure 4: Zusammenfassung des Algorithmus

Der vorgeschlagene Algorithmus besteht aus verschiedenen Schritten, die im folgenden erläutert und in dem Flowchart Diagramm 4 zusammengefasst werden.

Input - Als Input für den Algorithmus dienen transformierte Bilder aus der "Vogelperspektive". Ein solches Beispiel ist in Grafik 5 zu sehen.

Gaussian Blurring - Als Erstes wird Gaussian Blurring auf das Bild angewendet. Gaussian Blurring verringert die Schärfe des Bilds. So wird das Detaillevel des Bilds verringert. Dies hört sich erst mal intuitiv kontraproduktiv an, hilft jedoch insbesondere die verzerrten Linien in weiter Entfernung vom Fahrzeug, die durch die Transformation des Bildes entstanden, zu glätten. Dieser Effekt kann in den Grafiken 5 und 6 nachvollzogen werden.

Canny Edge Detection - Der Canny Edge Algorithmus wird zur Kantenerkennung eingesetzt. Kantenerkennung beschreibt die Technik, mithilfe von Unstetigkeiten in der Helligkeit, Grenzen von Objekten in Bildern zu finden. Durch die Canny Edge Detection wird das Bild binarisiert. Die Linien der Spuren werden durch die von der Canny Edge Detection gefundenen Kanten repräsentiert und in Grafik 7 gezeigt.

Region of Interest - Eine Region of Interest wird ausgewählt, um ungewollte Teile des Bilds herauszufiltern. Zu diesen ungewollten Teilen des Bilds zählen die Segmente, die von der Kamera nicht aufgefangen werden (rechts und links neben dem Fahrzeug). Dies geschieht, da hier Kanten erkannt werden, die dann im nächsten Schritt als Linien erkannt werden würden, da hier

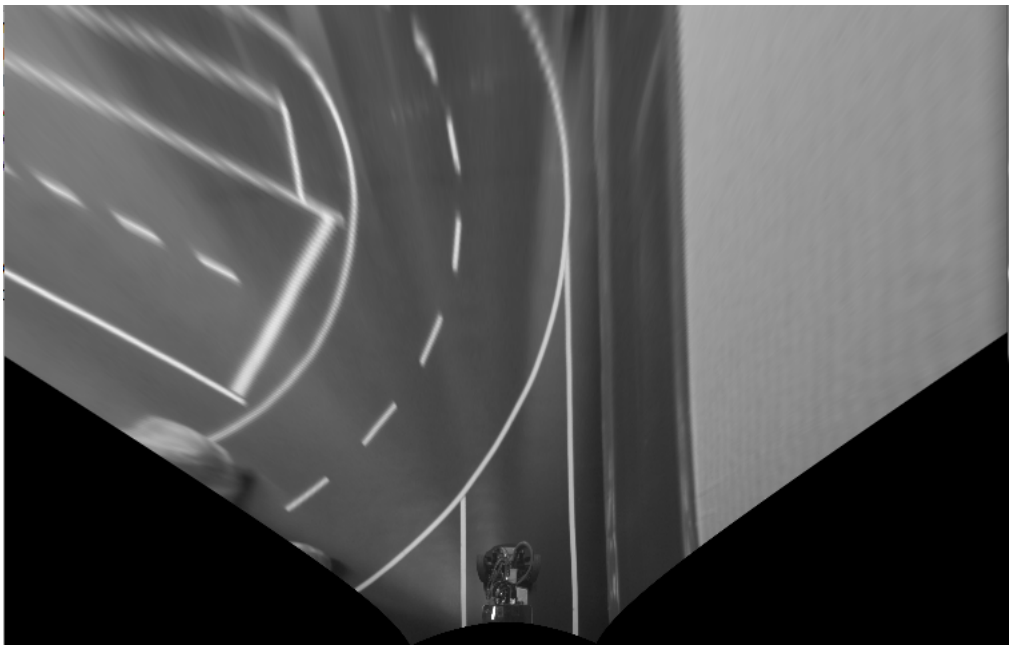

Figure 5: Transformierte Ansicht der Straße

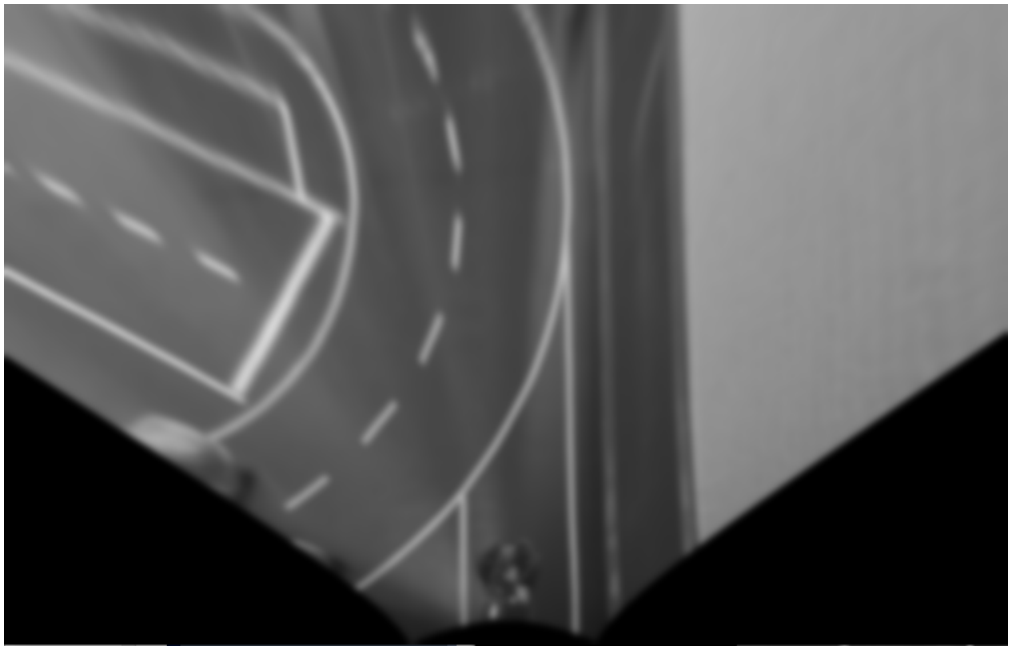

Figure 6: Bild nach Anwendung von Gaussian Blur Filter

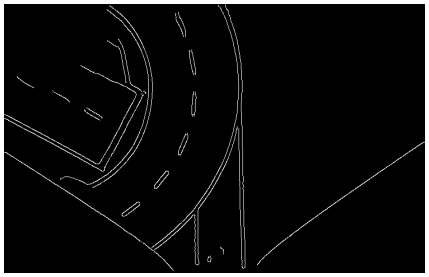

Figure 7: Bild nach Canny Edge Detection vor Ausschnitt von ungewollten Zonen

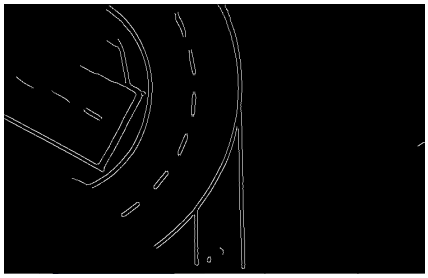

Figure 8: Bild nach Canny Edge Detection nach Ausschnitt von ungewollten Zonen

aber in jedem Fall keine Linie ist, können diese Bereiche herausgeschnitten werden. Grafik 8 zeigt das Bild nach diesem Vorgang.

Hough Linien Transformation - Durch die Hough Linien Transformation können gerade Linien im Bild erkannt werden. Dies bedeutet, dass Kurven nicht direkt erkannt werden können. Es ist jedoch möglich mehrere gerade Segmente in einer Kurve zu erkennen. Auch bei geraden Linien werden oftmals nur Segmente erkannt. Deshalb ist der Hough Algorithmus so parametrisiert, dass er möglichst viele Liniensegmente erkennt. Siehe dazu auch Sektion 4.2. Die Straße nach Anwendung der Hough Transformation ist in Grafik 9 zu erkennen.

Sortierung der Punkte von Liniensegmenten - Erkannte Liniensegmente werden durch zwei Punkte repräsentiert. Den Startpunkt und den Endpunkt. Es ist nicht garantiert das der erste Punkt eines Liniensegments der Startpunkt und der zweite der Endpunkt ist. Für eine vereinfachte Weiterverarbeitung werden die Punkte der Fragmente so angeordnet, dass davon ausgegangen werden kann, dass der erste Punkt den Startpunkt repräsentiert (Relativ zum Auto auf der Y Achse). Dieses Verfahren wird in Grafiken 9 und 10 ersichtlich. Hier sind Start- und Endpunkte in unterschiedlichen Farben markiert. In Grafik 9 ist durch einen gelben Kreis hervorgehoben, wie Punkte von Liniensegmenten unterschiedlich sortiert sind. In Grafik 10 sind diese dann richtig sortiert.

Figure 9: unsortierte Punkte

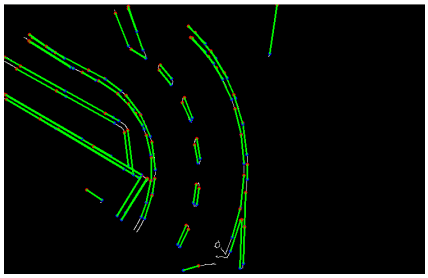

Figure 10: sortierte Punkte

Sortierung der Liniensegmente - Die Segmente werden nach der Y Koordinate der Startpunkte so sortiert, dass Segmente mit hoher Y Koordinate vorne liegen. Dies geschieht, um den nächsten Schritt, die Gruppierung von Segmenten, zu ermöglichen.

Gruppierung von Linensegmenten Die durch die Hough Linien Transformation erkannten Linienfragmente müssen zu zusammengehörigen Linien gruppiert werden. Dazu reicht es, aufgrund von Kurven und anderen Markierungen neben der Spur des Fahrzeugs, nicht einfach rechts und links von der Position des Fahrzeugs zu gucken.
Um die Linien zu gruppieren, werden die im vorherigen Schritt sortierten Fragmente iteriert. So werden alle Linienfragmente im Bild von unten nach oben durchgegangen. Für jedes Liniensegment determiniert eine Funktion, auf die im folgenden Paragrafen noch einmal eingegangen wird, die Zugehörigkeit zu anderen Segmenten, um alle zueinander gehörigen Segmente in einen Container zu gruppieren. Das Ergebnis dieser Gruppierung wird in Grafik 11 dargestellt. Segmente werden je nach Zuordnung zu einer Linie farblich unterschieden.

Zugehörigkeitsfunktion Die Zugehörigkeitsfunktion ("belongTogether()") determiniert, ob zwei Liniensegmente zueinander gehören. Hier wurde viel experimentiert, um eine möglichst optimale Gruppierung von Segmenten zu ermöglichen. Am Ende wurde sich für eine relativ simple Implementierung entschieden, da die Ergebnisse dieser im Vergleich zu komplexeren, deutlich erfolgversprechender waren.
Diese Implementierung prüft zwei Fälle. Zum einen prüft sie, ob sich zwei endlose Fragmente schneiden. Ist dies der Fall, wird geprüft, ob der End-

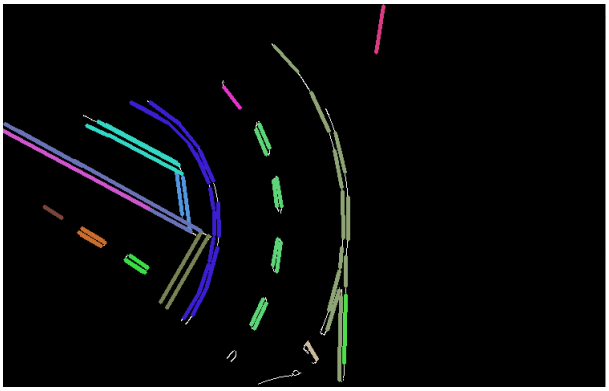

Figure 11: Gruppierte Liniensegmente

punkt des vorherigen Segments dicht genug an dem Startpunkt des nächsten Segments liegt und, ob der Winkel der beiden Fragmente vergleichbar ist. Dann gehören beide Segmente zueinander.
Hier musste acht gegeben werden, dass die geprüfte Distanz nicht zu klein und nicht zu groß ist. Ist diese zu klein, werden Segmente, die eigentlich zueinander gehören nicht als zugehörig erkannt. Ist sie zu groß, kann es passieren, dass Fragmente, die im nächsten Fall gruppiert werden, übersprungen werden und dann falsch zu einer neuen Linie gruppiert werden.
Werden die beiden Segmente im ersten Fall für als nicht zugehörig erkannt, wird ein zweiter Fall geprüft. In diesem Fall sollen Segmente, die parallel zueinander sind und sich somit nicht schneiden, als zueinander gehörend erkannt werden. Da über ein Segment lediglich Start und Endpunkt bekannt sind, wird die minimale Distanz aller Punktekombinationen beider Segmente geprüft. Ist diese klein genug und der Winkel vergleichbar wird noch sichergestellt, dass die Punkte mit der minimalen Distanz auf der x-Achse sehr dicht aneinander liegen. So wird in den meisten Fällen ausgeschlossen, dass ein Segment zu einer parallelen Linie gehört. Dies hätte zwar auch mit einer Verringerung der Distanz, die nötig ist, um zu entscheiden, ob Segmente zueinander gehören, erreicht werden können, dann wurden in den Experimenten jedoch parallele Segmente, die zu der gleichen Linie gehören oft nicht erkannt. Dies ist auch mit der aktuellen Distanz ab und zu noch möglich, wenn die Punkte der Segmente ungünstig liegen. Wenn ein paralleles Segment beispielsweise in der Mitte eines anderen Segments startet und deutlich später endet, ist die minimale Distanz zu dem anderen Segment verhältnismäßig groß, obwohl die Segmente sehr dicht aneinander liegen.

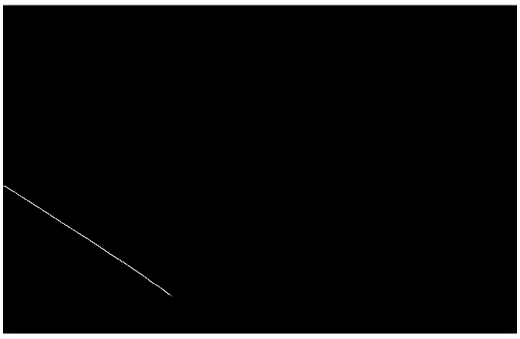

Figure 12: lower: 1, higher: 250

Figure 13: lower: 225, higher: 250

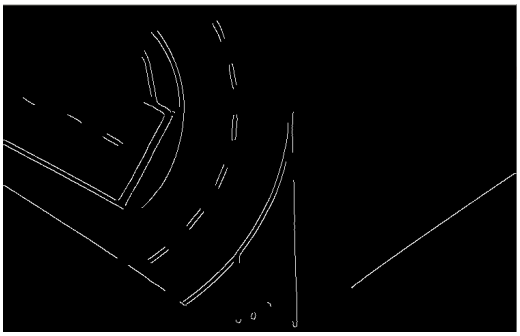

Figure 14: lower: 100, higher: 120

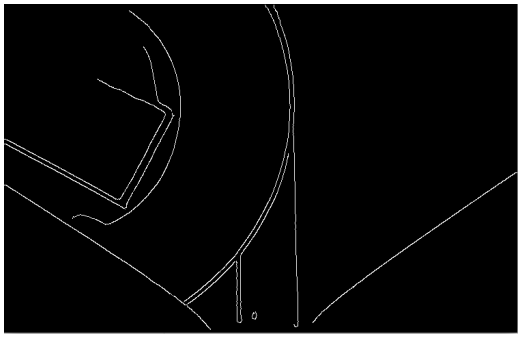

Figure 15: lower: 50, higher: 150

4 Parameterstudien

4.1 Canny Edge Detection

Canny Edge Detection benutzt zwei Schwellenwerte um Kanten zu erkennen. Wenn ein Pixelgradient höher als der obere Schwellenwert ist, wird das Pixel als Kante akzeptiert. Liegt ein Pixelgradientenwert unter dem unteren Schwellenwert, wird er abgelehnt. Um ein gutes Resultat zu erhalten wurde eine umfangreiche Parameterstudie abgeschlossen. Die Grafiken 12-15 zeigen einige dieser Tests. Grafik 16 zeigt das beste Resultat.

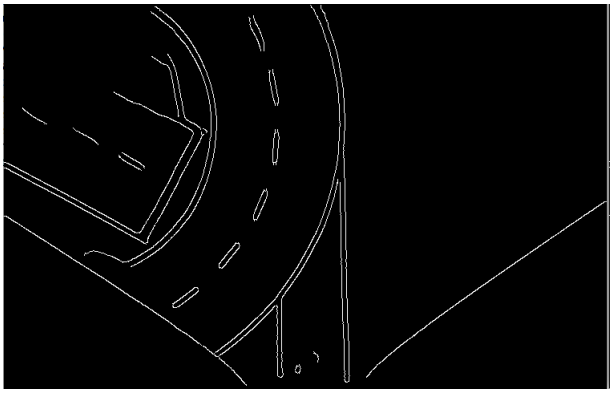

Figure 16: Best: lower: 50, higher: 120

4.2 Hough Linien Transformation

Um eine möglichst genaue Repräsentation einer Linie, insbesondere in einer Kurve, zu erhalten, müssen möglichst viele Segmente erkannt werden. Um dies zu erreichen wurde eine umfangreiche Parameterstudie realisiert. Da jedoch die Unterschiede der Experimente in einem Bild nicht unbedingt ersichtlich werden, müssten ganze Bilderreihen gezeigt werden. Dies würde zu vielen Seiten mit nur Bildern führen, daher wurde sich dagegen entschieden dies hier zu dokumentieren.

5 Future Work

Parallele Segmente, die zu einer Linie gehören, entstehen durch die Edge Detection, da eine relativ breite Linie zu beiden Seiten Unterschiede in der Helligkeit aufweist. Ein probierter Ansatz war Linien nur als eine Kante zu erkennen. Dies wurde mittels morphologischer Methoden erreicht. Zuerst wurden die aus der Canny Edge Detection resultierenden Kanten erweitert, sodass parallele Kanten, die zu einer Linie gehören zusammengefügt werden. Danach wurden die Kanten erodiert, um wieder dünnere Linien, die für die Linienerkennung nötig sind, zu bekommen. Grafiken 17 und 18 zeigen dieses Verfahren. Es wurde sich jedoch gegen dieses Verfahren entschieden, da einige Linien nur als einzelne Kante erkennt werden, welche nach dem erweitern dünnere Linien geben und dann mit dem Erodieren verloren gehen. Der Informationsverlust war in einigen Frames, wie z.B. dem in Grafiken

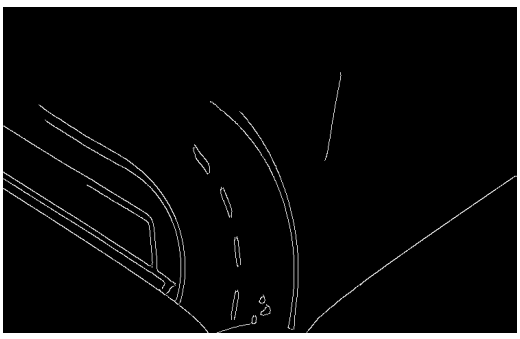

Figure 17: lower: 100, higher: 120

Figure 18: lower: 50, higher: 150

17 und 18 gezeigten Frame, zu groß. An dieser Stelle könnte jedoch weiter angesetzt werden und bessere Parameter gefunden werden.

Eine andere Option wäre es auf die Canny Edge Detection zu verzichten und andere Binarisierungsverfahren einzusetzen.

Eine weitere Stelle, an der der Algorithmus verbessert werden kann ist die Funktion, die prüft, ob Liniensegmente zugehörig sind. Diese ist nicht optimal und signalisiert in einigen Fällen Zugehörigkeit, obwohl keine besteht.

Außerdem kann der gesamte Algorithmus, der Liniensegmente zu Linien gruppiert, verbessert werden. Hier werden oft Segmente übersprungen. Dies wird in Grafik 19 erkenntlich. Es wird ersichtlich, dass es sich um Segmente handelt, die übereinander liegen. Eine Möglichkeit wäre es hier den Schneidepunkt dieser übereinanderliegenden Segmente zu finden. Nun kann geprüft werden, ob dieser Punkt auf beiden Segmenten liegt. Ist dies der Fall, können diese beiden Segmente zu einem Segment zusammengefügt werden.

Als letzter Schritt, um diesen Algorithmus kompatibel mit dem Algorithmus des Faust Teams zu machen, müssen noch gegenüberliegende Punkte zurückgegeben werden. Hier müsste eine Berechnung erfolgen, die zum einen diese Punkte auf einer Spur auswählt und zum anderen den gegenüberliegenden Punkt auf der anderen Spur berechnet.

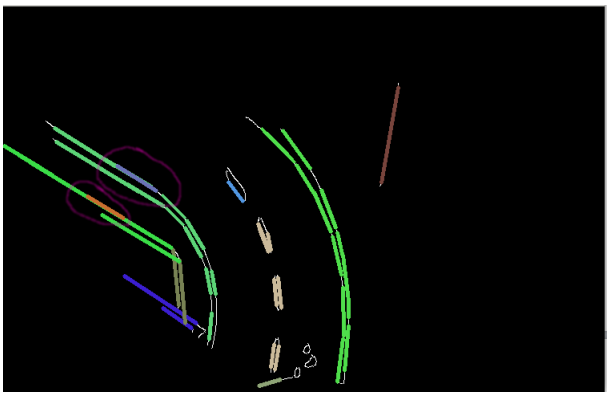

Figure 19: Übersprungene Segmente