
Masterarbeit

Fahrspurerkennung in Luftaufnahmen mittels Fahrzeugtrajektorien

im Master-Studiengang Informatik
der Hochschule Furtwangen

Steffen Schmid

Zeitraum: Wintersemester 2018
Prüfer: Prof. Dr. Christoph Reich
Zweitprüfer: M.Sc. Stefan Kaufmann

Firma: IT-Designers GmbH
Betreuer: M.Sc. Stefan Kaufmann

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 28. Dezember 2018 _____
Unterschrift

Kurzfassung

Schlagworte:

Inhaltsverzeichnis

Kurzfassung	iii
1 Einleitung	1
1.1 Rahmen der Arbeit	2
1.1.1 Das Projekt MEC-View	2
1.1.2 Das MEC-View Teilprojekt Luftbeobachtung	3
1.2 Motivation und Ziele	3
1.3 Aufbau dieser Arbeit	4
2 Rekonstruktion von Fahrzeugtrajektorien aus Luftaufnahmen	5
2.1 Erkennung und Verfolgung von Objekten in Videoaufnahmen	5
2.1.1 Supervised-Tracking	5
2.1.2 Unsupervised-Tracking	7
2.2 Positionsbestimmung in Videoaufnahmen	9
2.3 Herausforderungen bei der Rekonstruktion von Fahrzeugtrajektorien	11
3 Clusteranalyse	12
3.1 Cluster-Sets und Cluster	14
3.1.1 Eigenschaften von Cluster-Sets	14
3.1.2 Eigenschaften von Clustern	14
3.2 Cluster-Algorithmen	16
3.2.1 Vernetzungs-Modelle	17
3.2.2 Prototyp-Modelle	18
3.2.3 Distributions-Modelle	20
3.2.4 Dichte-Modelle	21
3.3 Distanzmaße zum Vergleich von Fahrzeugtrajektorien	23
3.3.1 HU-Distanz	24
3.3.2 Hausdorff-Distanz	24
3.3.3 Longest-Common-Subsequence	25
4 Verwandte Arbeiten	27
4.1 Clusteranalyse von Trajektorien	27
4.2 Erkennung und Definition von Fahrspuren	33
4.3 Defizite vorhandener Lösungen und benötigte Neuerungen	38

5 Konzeption des Spurerkennungs-Moduls	40
5.1 Überblick über das Gesamtsystem	40
5.2 Anforderungen an das Modul	41
5.2.1 Funktionale Anforderungen	41
5.2.2 Nicht funktionale Anforderungen	41
5.3 Entwurf des Moduls	42
6 Realisierung der Fahrspurerkennung	43
6.1 Repräsentation und Vorverarbeitung der Trajektoriedaten	43
6.1.1 Trajektorie-Repräsentation	43
6.1.2 Vorverarbeitung der Trajektorien	46
6.2 Clusteranalyse der Trajektorien	49
6.2.1 Ansatz Spectral-Clustering und modifizierte Hausdorff-Distanz	49
6.2.2 Ansatz DBSCAN und LCSS	53
6.2.3 Erkennung von Abbiegespuren	56
6.3 Spur-Geometrie Bestimmung	59
6.3.1 Ausfilterung von Spurwechselvorgängen	59
6.3.2 Bestimmung der Spurmittellinien	60
6.3.3 Angleichung benachbarter Spur-Enden	62
6.3.4 Bestimmung der Spurhüllen	63
6.3.5 Partitionierung von Fahrspuren	69
6.3.6 Optimierung der Spur-Geometrien	72
6.3.7 Anlegen der Fahrspuren in der TrackerApplication	74
7 Auswertung und Ergebnisse	76
7.1 Evaluierung der Datenvorverarbeitung	76
7.2 Evaluierung der Clusteranalyse	78
7.3 Evaluierung der Spur-Geometrie-Bestimmung	80
7.4 Ergebnisse der Spurerkennung	82
8 Fazit und Ausblick	84
Literaturverzeichnis	85

Abbildungsverzeichnis

1.1	Grundkonzept des MEC-View Projektes	2
2.1	Übersicht Tracking mit Klassifikator	6
2.2	Haar-ähnliche Merkmale a), Beispiele für erkannte Regionen in einem Gesicht b)	7
2.3	Aufbau eines CNNs	8
2.4	Lochkameramodell	9
3.1	Rohdaten (links) und erwünschtes Clustering-Ergebnis (rechts)	13
3.2	Ablauf einer Clusteranalyse	13
3.3	Visualisierung verschiedener Clusterarten	16
3.4	Agglomeratives Clustering dargestellt als Dendrogramm (links) und geschachteltes Cluster-Diagramm (rechts)	17
3.5	Funktionsweise von k-Means	19
3.6	Darstellung des EM-Cluster-Algorithmus über mehrere Iterationen	21
3.7	Schritte des DBSCAN Algorithmus	22
3.8	Erreichbarkeit in DBSCAN	22
3.9	Trajektorien im 2-dimensionalen Raum	23
4.1	Zweistufiger Clustering-Vorgang von Hu et al.	28
4.2	Funktionsweise der modifizierten Hausdorff-Distanz	29
4.3	Zerlegung einer Trajektorie in Sub-Trajektorien	30
4.4	Verschiebung einer Trajektorie im Raum	31
4.5	Ergebnisse der Spurmittellinien-Erkennung (Ren et al.)	33
4.6	Spurhüllen in Hu et al., 2006	34
4.7	Routen-Definition und Ergebnisse Routen-Erkennung (Makris et al.)	35
4.8	Referenz-Trajektorien und Pfade (Morris et al.)	36
4.9	Ergebnisse Histogramm Erstellung und Spurextraktion (Hsieh et al.)	38
5.1	Kontext des Moduls Spurerkennung	40
5.2	Grundstruktur des Spurerkennungs-Algorithmus	42
6.1	Erkannte Fahrzeuge und deren Front-Position	44
6.2	Aufbau Trajektorie-Klasse	44
6.3	Stuttgarter Neckartor a) und Trajektorien b)	45
6.4	Punktwolken vor Lichtsignalanlagen a), Unterbrechungen aufgrund von Überdeckung b)	46

6.5	Ergebnisse der zwei ersten Vorverarbeitungsschritte	48
6.6	Identifikation von unterbrochenen Trajektorien	49
6.7	Ergebnisse Clusteranalyse B10-Entennest (Ansatz Atev et al.)	52
6.8	Ergebnisse Clusteranalyse DBSCAN und LCSS-Distanz	55
6.9	Abbiegespur a), zugehörige Trajektorien b), Spurcluster mit Abbiegespuren c)	56
6.10	Trajektorie-Cluster mit Spurwechselvorgängen	59
6.11	Trajektorie-Cluster ohne Spurwechselvorgänge	60
6.12	Spurmittellinie in einem Trajektorie-Cluster a), Spurmittellinien Neckartor- Kreuzung b)	62
6.13	Konzept der Spur-Enden-Angleichung	63
6.14	Mittellinien ohne Angleichung a) und mit Angleichung b)	63
6.15	Bestimmung Spürhüllen	64
6.16	Aufbau LaneGeometry Klasse	64
6.17	Bestimmung Spürhüllen	66
6.18	Plot Spurmittellinien und Hüllen a), Ergebnis Spur-Geometrien in Tracke- rApplication b)	67
6.19	Beispiele “unechter” Spur-Geometrien	68
6.20	Primäre (blau) und sekundäre (grün) Spuren a), sich überlagernde und kreuzende Spur-Paare b)	69
6.21	Überlagerung zweier Spur-Abschnitte	70
6.22	Plot partitionierte Spur-Geometrien a), Ergebnis in der TrackerApplication b)	72
6.23	Beispiel zu schmale Spur-Geometrien	72
6.24	Beispiel zu schmale Spur-Geometrien	73
6.25	Datenmodell der Spurdefinition	74
6.26	Angelegte Fahrspuren in der TrackerApplication	75
7.1	Ergebnis Vorverarbeitung Heilbronner-Straße	76
7.2	Straßenausschnitt Steinheim a), gekürzte Trajektorien b)	77
7.3	Trajektorie-Cluster Heilbronner-Straße und Neckator-Kreuzung	78
7.4	Trajektorien Düsseldorf Datensatz a), Spurcluster b)	79
7.5	Ergebnisse Spur-Geometrie-Bestimmung	80
7.6	Auftretende Probleme in den Spur-Geometrien	81
7.7	Erkannte Fahrspuren auf geraden Straßenabschnitten	82
7.8	Erkannte Fahrspuren auf Kreuzungen	83
7.9	Erkannte Fahrspuren Kreisverkehr	83

Listings

6.1	Pseudocode Trajektorie Resampling	47
6.2	Pseudocode LCSS Bestimmung	54
6.3	Pseudocode Split-Punkt Bestimmung	57
6.4	Pseudocode Cluster Post-Processing	59
6.5	Pseudocode Cluster Post-Processing	61
6.6	Pseudocode Überprüfung der Parallelität zweier Mittellinien	65
6.7	Pseudocode Identifikation Spurwechsel-Spur	68
6.8	Pseudocode Auswahl zu partitionierende Fahrspur	71

1 Einleitung

Staus und zäh fließender Verkehr sind sowohl auf Schnell- und Autobahnen, als auch in Städten ein großes Problem und Ärgerniss für Autofahrer. Sie kosten diese nicht nur wertvolle Zeit, sondern auch viel Geld. Laut einer Studie von [Cookson et al.] kosten Staus jeden deutschen Autofahren pro Jahr durchschnittlich 1770 €. In Summe ergeben sich hieraus beinahe 80 Milliarden Euro an Kosten. Stau ist allerdings nicht nur finanziell für Privatpersonen oder auch Unternehmen ein großer Faktor, sondern er erhöht auch das Unfallrisiko und trägt maßgeblich zur schlechten Luftqualität in Innenstädten bei. Aufgrund längerer Fahrzeiten und der häufigen Be- und Entschleunigung, steigt der Kraftstoffverbrauch der Fahrzeuge und dadurch auch die Schadstoffbelastung in der Luft [Hemmerle, 2016].

Die wichtigste Voraussetzung um Staus präventiv entgegenwirken zu können, ist, den Verkehr so gut wie möglich zu verstehen. Nötig ist ein Verständnis des Straßenverkehrs als Ganzes, sowie der Auswirkungen, welche einzelne Verkehrsteilnehmer und deren Verhalten, auf diesen haben. Hierzu ist das Erstellen von Simulationen sowie die Auswertung realer Verkehrsaufkommen unerlässlich. Die auf diese Weise gesammelten Erkenntnisse bilden die Grundlage, um Straßenabschnitte, insbesondere auch in Innenstädten, intelligent zu gestalten. Des Weiteren können sie eingesetzt werden, um beispielsweise Ampelschaltungen in Städten zu optimieren, wovon auch bestehende Infrastrukturen profitieren können.

Dank der Tatsache, dass unbemannte Luftfahrzeuge (UAV) wie Drohnen immer leichter und günstiger verfügbar sind, und die von ihnen erstellten Aufnahmen teils eine sehr gute Qualität besitzen, werden diese immer häufiger zur Analyse des Straßenverkehrs eingesetzt. Über Methoden aus dem Umfeld des maschinellen Sehens und maschinellen Lernens können aus Luftaufnahmen eine Vielzahl an interessanter Informationen extrahiert werden.

Diese Arbeit beschäftigt sich mit der Realisierung einer automatischen Fahrspurerkennung in Luftaufnahmen. Hierzu werden die Trajektoriedaten von Fahrzeugen ausgewertet. Die Analyse von Verkehrssituation wird, nicht zuletzt auch aufgrund der zunehmenden Relevanz des autonomen Fahrens, immer wichtiger. Eine automatisierte Spurerkennung ist hierbei ein wichtiger Teil des Prozesses, da mit Hilfe der erkannten Spuren unter anderem Spurwechsel- und Überholvorgänge sowie das Verhalten der Fahrzeuge auf einer Spur untereinander untersucht werden können.

1.1 Rahmen der Arbeit

In den nachfolgenden Abschnitten wird das Forschungsprojekt MEC-View und dessen Teilprojekt Luftbeobachtung vorgestellt.

1.1.1 Das Projekt MEC-View

Das Forschungsprojekt MEC-View, welches vom Bundesministerium für Wirtschaft und Energie (BMWi) gefordert wird, hat das Ziel, autonomes Fahren im urbanen Raum zu ermöglichen und für alle Verkehrsteilnehmer sicher zu gestalten. Gerade in Innenstädten sind die Möglichkeiten des autonomen Fahrens aufgrund von unübersichtlichen Kreuzungen, Fußgängern und Fahrradfahrern, verdeckten Sichten und anderen Faktoren begrenzt. Abbildung 1.1 gibt einen Überblick über das Forschungsprojekt und veranschaulicht dessen Ziel.

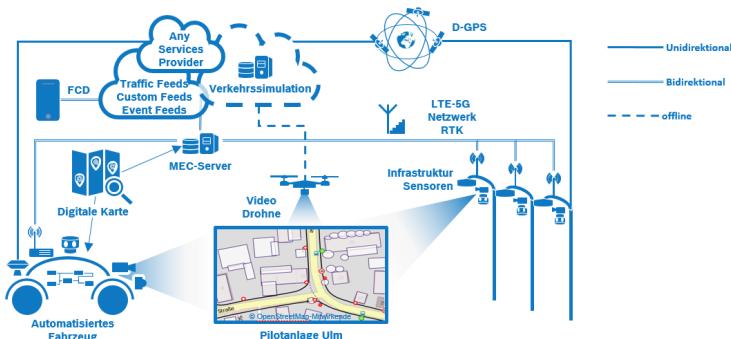


Abbildung 1.1: Grundkonzept des MEC-View Projektes [MEC-View, 2018]

In urbanen Umgebungen sollen neben Daten von fahrzeuginternen Sensoren auch Informationen externer Infrastruktur-Sensoren verwendet werden, damit ein Fahrzeug eine fundierte Verhaltensentscheidung auf Basis eines detaillierten Umfeldmodells treffen kann. Im Rahmen des Projektes MEC-View wird eine Pilot-Anlage zur Umfelderfassung an einer vorfahrtsberechtigten Straßenkreuzung in Ulm aufgebaut und getestet. In dieser Anlage werden die Verkehrsteilnehmer über Kameras und LIDAR-Sensoren erfasst und die ermittelten Daten über ein schnelles LTE/5G-Mobilfunknetz an einen *Mobile Edge Computing* (MEC) Server übertragen. Hier werden die Daten in Echtzeit zu einem Umfeldmodell fusioniert, welches anschließend den autonomen Fahrzeugen zur besseren Navigation zur Verfügung gestellt wird. Beteiligt an diesem Forschungsprojekt sind neben der IT-Designers GmbH unter anderem auch die Daimler AG, die Robert Bosch GmbH, Osram, Nokia und die Universität Ulm. Jeder Projektpartner ist verantwortlich für unterschiedliche Teilspekte des Projektes. Die IT-Designers GmbH, bei welcher diese Arbeit angefertigt wird, entwickelt den MEC-Server und ist verantwortlich für das Teilprojekt Luftbeobachtung. [MEC-View, 2018]

1.1.2 Das MEC-View Teilprojekt Luftbeobachtung

Im MEC-View Teilprojekt Luftbeobachtung werden Verkehrssimulationen erstellt, welche dabei helfen das Verhalten des Verkehrs besser zu verstehen und es somit ermöglichen, diesen zu optimieren. Mithilfe der Simulationen kann beispielsweise untersucht werden, wie durch die Anpassung von Verkehrssteuerungsanlagen oder die Änderung des Fahrverhaltens einzelner Fahrzeuge eine Verbesserung der Verkehrssituation erreicht werden kann. Die Erkenntnisse können insbesondere auch in die Verhaltenssteuerung von autonomen Fahrzeugen mit einfließen.

Die Simulationen werden im MEC-View Projekt anhand von Luftbeobachtungen erstellt, welche von Drohnen getätigten werden. In den Videoaufnahmen werden mithilfe von neuronalen Netzen die Positionen der Fahrzeuge ermittelt. Mittels dieser kann anschließend beispielsweise die Geschwindigkeit und Beschleunigung der einzelnen Verkehrsteilnehmer bestimmt werden. Zur Erstellung der Simulationen ist es zudem wichtig, eine Kenntnis der Topologie der untersuchten Straßen, das heißt des Verlaufs der Fahrbahnen und Spuren, zu besitzen. In Kombination mit den Fahrzeugpositionen können so interessante Kenngrößen wie der Verkehrsfluss oder die Verkehrsdichte bestimmt werden.

1.2 Motivation und Ziele

Im Rahmen dieser Arbeit wird ein Verfahren zur automatischen Erkennung von Fahrspuren in Luftaufnahmen auf Basis von Trajektoriedaten entwickelt. Die Spurerkennung wird in die Anwendung *Vehicle-Tracker* integriert, welche im Rahmen des MEC-View Luftbeobachtungs Projekt erstellt wird. Sie dient der Analyse von Luftbeobachtungen des Straßenverkehrs. Um eine solche Auswertung durchführen zu können, ist es wichtig den Verlauf und die Positionen der Fahrspuren, welche sich auf einem bestimmten Straßenabschnitt befinden, zu kennen. In der Applikation *Vehicle-Tracker* mussten bislang die Fahrspurverläufe in einer Aufnahme händisch definiert werden. Dieser Prozess ist insbesondere dann aufwendig, wenn die zu untersuchenden Straßenabschnitte beispielsweise mehrspurige Kreuzungen oder Kreisverkehre beinhalten. Das in dieser Arbeit entwickelte Spurerkennungs-Modul soll die manuelle Spur-Definition ersetzen und es so ermöglichen in Zukunft mehr Luftaufnahmen mit weniger Aufwand auszuwerten.

Der Verlauf und die Geometrie der Fahrspuren wird in dieser Thesis anhand der Bewegungsbahnen von Fahrzeugen, den sogenannten Trajektorien, ermittelt. Im Gegensatz zu einer visuellen Detektierung hat das Verfahren den Vorteil, dass Fahrspuren auch in Aufnahmen mit schlechten Lichtverhältnissen oder Verdeckungen der Fahrbahnen erkannt werden können.

Zum Thema Spurerkennung existieren zwar bereits einige Veröffentlichungen, allerdings können die hierin vorgestellten Methoden meist nur in sehr speziellen Szenarien eingesetzt werden oder die erkannten Spuren entsprechen den realen Fahrspurverläufen nur schlecht.

Ziel dieser Arbeit ist es, ein Verfahren zu entwickeln, welches Fahrspuren in möglichst vielen unterschiedlichen Straßentopologien erkennen kann. Die aus den Luftaufnahmen abgeleiteten Spuren sollen außerdem den realen Fahrbahnverläufen so gut wie möglich entsprechen.

1.3 Aufbau dieser Arbeit

Die vorliegende Arbeit ist wie folgt strukturiert:

- Die zur Verständnis der Arbeit und des entwickelten Spurerkennungs-Algorithmus benötigten Grundlagen sind in **Kapitel 2 und 3** beschrieben. Kapitel 2 erläutert, wie aus Luftaufnahmen die Positionen von Fahrzeugen rekonstruiert und diese anschließend in ein Weltkoordinatensystem überführt werden können. Kapitel 3 stellt die grundlegenden Konzepte der Clusteranalyse vor, welche bei der Umsetzung der Spurerkennung zum Einsatz kommt.
- In **Kapitel 4** werden verwandte Arbeiten, welche sich bereits mit der Thematik der Spurerkennung und der Clusteranalyse von Trajektorien befassen, vorgestellt und untersucht. Zudem werden defizite der vorhandenen Lösungen und benötigte Neuerungen aufgezeigt.
- In **Kapitel 5** wird das Konzept für die Umsetzung der Spurerkennung vorgestellt. In diesem Abschnitt werden zudem Anforderungen festgehalten und das Spurerkennungs-Modul wird in den Gesamtkontext der Applikation *Vehicle-Tracker* eingeordnet.
- Nach der Konzeption wird in **Kapitel 6** vorgestellt, wie die Spurerkennung konkret umgesetzt wurde. Es werden die verschiedenen Schritte des entwickelten Algorithmus erläutert.
- In **Kapitel 7** wird die entwickelte Methode zur Spurerkennung in Luftaufnahmen evaluiert. Es wird auf die Qualität und Probleme der wichtigsten Verarbeitungsschritte des Algorithmus eingegangen. Außerdem werden die konkreten Ergebnisse in Form von Screenshots vorgestellt.
- **Kapitel 8** bildet den Schluss dieser Masterarbeit. Hier wird in aller Kürze ein Fazit gezogen und ein Ausblick gegeben, welche Verbesserungen in Zukunft an dem entwickelten Algorithmus noch vorgenommen werden können.

2 Rekonstruktion von Fahrzeugtrajektorien aus Luftaufnahmen

In diesem Kapitel wird erläutert, wie auf Luftaufnahmen Trajektorien von Fahrzeugen rekonstruiert werden können. Eine solche Rekonstruktion wird zwar im Rahmen dieser Arbeit nicht entwickelt, allerdings ist es wichtig ein Grundverständnis der benötigten Methoden und Schritte zu besitzen, um die Trajektoriedaten und die darin enthaltenen Defekte interpretieren zu können. Es wird nachfolgend beschrieben, wie bewegte Objekte in Videoaufnahmen erkannt und verfolgt werden können um anschließend deren Positionen in einem Weltkoordinatensystem zu bestimmen. Am Ende des Kapitels werden einige Schwierigkeiten, welche bei der Rekonstruktion der Trajektorien auftreten können aufgezeigt.

2.1 Erkennung und Verfolgung von Objekten in Videoaufnahmen

Es existieren diverse Ansätze, welche zur Erkennung und Verfolgung von Objekten in Videoaufnahmen eingesetzt werden können. Nachfolgend werden zwei unterschiedliche Vorgehens-Arten vorgestellt.

2.1.1 Supervised-Tracking

Bei einem Verfahren aus der Kategorie *Supervised-Tracking* wird ein initial manuell ausgewählter Bildbereich automatisch mit Hilfe eines erlernten Klassifikators verfolgt. Der Klassifikator muss hierbei zwischen Fahrzeugen und der Umgebung unterscheiden können. Der hier beispielhaft vorgestellte Supervised-Tracking Ansatz beruht auf der Arbeit von [Grabner et al., 2006]. Sein grundlegendes Vorgehen ist in Abbildung 2.1 dargestellt und kann wie folgt beschrieben werden:

- a) Ein zu verfolgendes Objekt befindet sich zum Zeitpunkt t an bekannter Position p_1
- b) Zum Zeitpunkt $t + 1$: Anwendung des Klassifikators auf Positionen um p_1
- c) Erstellen einer *Confidence Map*, welche die Wahrscheinlichkeit darstellt, das verfolgte Objekt gefunden zu haben
- d) Updaten des Trackers auf Position des Maxima der *Confidence Map*

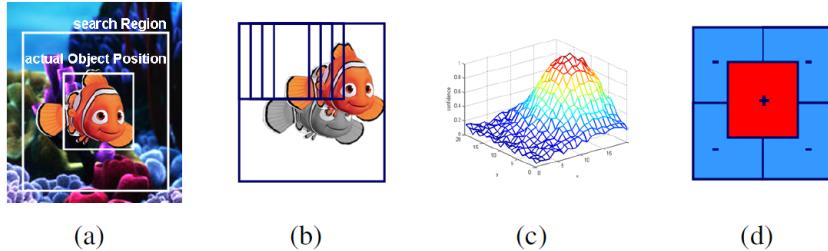


Abbildung 2.1: Übersicht Tracking mit Klassifikator [Grabner et al., 2006]

Zum Erlernen eines stabilen Klassifikators, wird in [Grabner et al., 2006] ein On-line AdaBoost Algorithmus eingesetzt, welcher mehrere *schwache* Klassifikatoren zu einem *starken* Klassifikator kombiniert. Schwache Klassifikatoren müssen nur eine Erkennungsrate von mehr als 50% besitzen und somit wenig besser als zufallsbedingtes Auswählen sein. Starke Klassifikatoren entstehen durch die Kombination von mehreren schwachen Klassifikatoren. Die Auswahl von schwachen Klassifikatoren erfolgt über sogenannte Selektoren, welche aus einer Menge immer jenen wählen, welcher die geringste Fehlerrate bei der Erkennung der Trainings-Objekte besitzen. Der Klassifikator mit der schlechtesten Erkennungsrate wird in jeder Trainingsiteration ersetzt, um das Training zu verbessern. Großer Vorteil der On-line AdaBoost Methode ist, dass sie es ermöglicht, starke Klassifikatoren während des eigentlichen Trackingvorganges zu erlernen. Nach jedem Trackingschritt wird das erfolgreich erkannte Objekt in Trainingssätze zerlegt, auf welche die Klassifikatoren angewandt werden um ihre Performance zu evaluieren. So wird in jedem Schritt die Menge der schwachen Klassifikatoren und der Selektoren aktualisiert. Die Wahl von effizient berechenbaren schwachen Klassifikatoren macht dies möglich.

Die in [Grabner et al., 2006] verwendeten Klassifikatoren sind binär, das heißt, sie teilen Objekte in die zwei Klassen *erkannt* und *nicht erkannt* auf. Konkret werden Haar-ähnliche Bildmerkmale nach [Viola et al., 2001] als schwache Klassifikatoren verwendet. Diese sind ein Mittel zur Identifikation von Kontrastunterschieden in Bildern, welche sich sehr gut zur Erkennung von Kanten und Linien eignen. Ein Beispiel der Haar-ähnlichen Merkmale und ihres Einsatzes bei der Gesichtserkennung ist in Abbildung 2.2 dargestellt.

Diese Merkmale werden als schwache Klassifikatoren mit zufälliger Skalierung, Größe und Position auf dem Bild platziert. Sie suchen in dieser Region anschließend nach den von dem Muster definierten Konturunterschieden. Eine Bereich gilt als erkannt, wenn der Betrag der Differenz der Pixelsumme des weißen und schwarzen Bereiches des Musters unter einem festgelegten Grenzwert liegt.

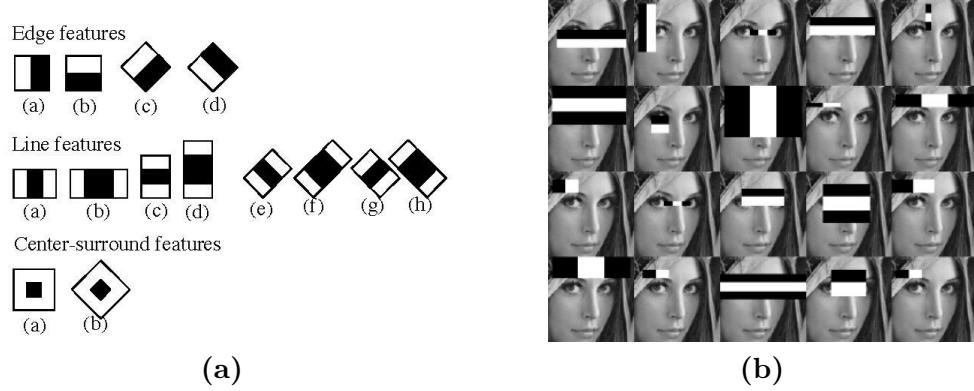


Abbildung 2.2: Haar-ähnliche Merkmale a), Beispiele für erkannte Regionen in einem Gesicht b) [Divyansh Dwivedi, 2018]

Haar-ähnliche Bildmerkmale eignen sich auch gut zur Erkennung von Fahrzeugen in Bildern, da auch diese anhand klarer Kanten identifiziert werden können. Gelingt die Verfolgung eines Objektes in einer Videoaufnahme, so existiert nach dem Tracking idealerweise für jedes Video-Frame, in welchem das Objekt zu sehen ist, eine Position in Pixel-Koordinaten. Der große Nachteil des Supervised-Trackings ist, dass die Positionen verfolgter Objekte initial manuell definiert werden müssen. Wenn in einer Aufnahme viele Objekte verfolgt werden sollen, dann ist diese initiale Positionsbestimmung sehr aufwendig.

In nachfolgendem Abschnitt wird daher eine weitere Klasse von Tracking-Verfahren beschrieben, welche die Positionen von Objekten voll-automatisch in Aufnahmen ermitteln können.

2.1.2 Unsupervised-Tracking

Von *Unsupervised-Tracking* wird gesprochen, wenn zur Identifikation und Verfolgung eines Objektes in einem Video keine initiale *Region of Interest* (ROI) definiert werden muss, welche die Startposition des Objektes festlegt. Die Erkennung und Verfolgung funktioniert daher komplett automatisch. Es existieren verschiedene Verfahren, welche in die Kategorie des Unsupervised-Tracking fallen. Auch in der *Vehicle-Tracker* Anwendung, auf welcher diese Arbeit aufbaut, wird Unsupervised-Tracking eingesetzt, um die Positionen der Fahrzeuge in den Videoaufnahmen zu bestimmen. Das hier eingesetzte Verfahren wird nachfolgend beispielhaft in groben Zügen erläutert.

Während im Fall des oben beschriebenen Supervised-Tracking-Ansatzes Objekte grundsätzlich einmal händisch detektiert und anschließend verfolgt werden, werden im Fall des Ansatzes, welcher in der *Vehicle-Tracker* Applikation zur Anwendung kommt, die Positionen der Fahrzeuge in jedem Video-Frame neu bestimmt. Die Positionsinformationen eines

Fahrzeugs werden anschließend zu zusammenhängenden Trajektorien zusammengefasst. Erkannt werden die Fahrzeuge mithilfe der *Object-Detection-API* von Tensorflow¹.

Das quelloffene Machine-Learning Framework Tensorflow bietet in seiner *Object-Detection-API* verschiedene Netzwerk-Architekturen und vortrainierte Modelle an, welche zur Erkennung und Klassifizierung von Objekten eingesetzt werden können [Huang et al., 2018]. Die hierzu angebotenen neuronale Netze sind Weiterentwicklungen von *Convolutional Neural Networks* (CNN), welche sich gut zur maschinellen Verarbeitung von Bilddaten eignen.

In Abbildung 2.3 ist der Aufbau eines CNNs inklusive eines *Fully-Connected Layers* dargestellt. Ein genaues Verständnis der Funktionsweise eines solchen Netzwerks ist im Fall dieser Arbeit nicht notwendig. Grundlegend besteht es allerdings aus zwei Komponenten: Der erste Teil des Netzwerkes ist für das *Feature Learning* verantwortlich, während der zweite Teil der Klassifizierung dient. In der Phase des *Feature Learnings* extrahiert das Netzwerk aus einem Bild Merkmale, mittels welcher die zu erkennenden Objekte beschrieben werden können. Anhand der extrahierten Merkmale können ROIs bestimmt werden. Um welches Objekt es sich in einer speziellen ROI handelt, wird anschließend anhand des *Fully-Connected-Layers* bestimmt.

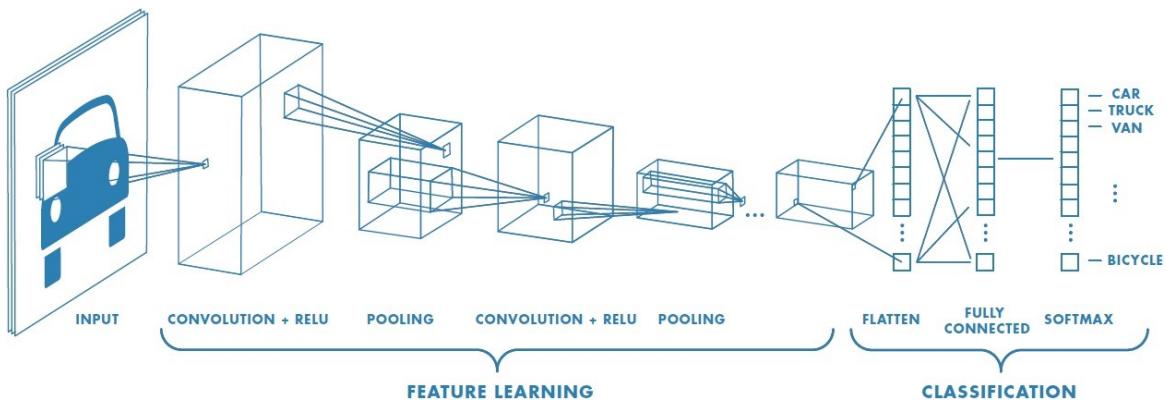


Abbildung 2.3: Aufbau eines CNNs [Patel and Pingel, 2017]

Im Fall der Tensorflow Object Detection API werden die Positionen der erkannten Objekte über sogenannte *Bounding-Boxes* beschrieben. Diese leiten sich aus den ROIs ab, welche vom CNN ermittelt werden.

Um aus den Positionsinformationen, welche aus den einzelnen Video-Frames extrahiert wurden, zusammenhängende Fahrzeugtrajektorien erstellen zu können, müssen diese eindeutig einzelnen Fahrzeugen zugeordnet werden.

¹ Tensorflow - <https://www.tensorflow.org/>

2.2 Positionsbestimmung in Videoaufnahmen

Nachdem die Positionen der Fahrzeuge in 2D-Bildkoordinaten bestimmt wurden, müssen diese in 3D-Weltkoordinaten umgewandelt werden, um die genauen Fahrzeugpositionen auf der Straße zu erhalten. Dank der Verwendung eines statioären Weltkoordinatensystems mit Einheit Meter, lassen sich zudem reale Entferungen in der Aufnahme ermitteln, was beispielsweise für die Bestimmung der Fahrzeuggeschwindigkeiten sehr wichtig ist.

Um eine Bildkoordinate in eine Weltkoordinate überführen zu können, müssen vorher verschiedene Kameraeigenschaften ermittelt werden. Hierzu ist es wichtig ein grundlegendes Verständnis von der Funktionsweise einer Kamera zu besitzen. Anhand eines Lochkameramodells lässt sich die Funktionsweise der Bildprojektion einer Kamera herleiten. Ein solches Modell ist in Abbildung 2.4 dargestellt.

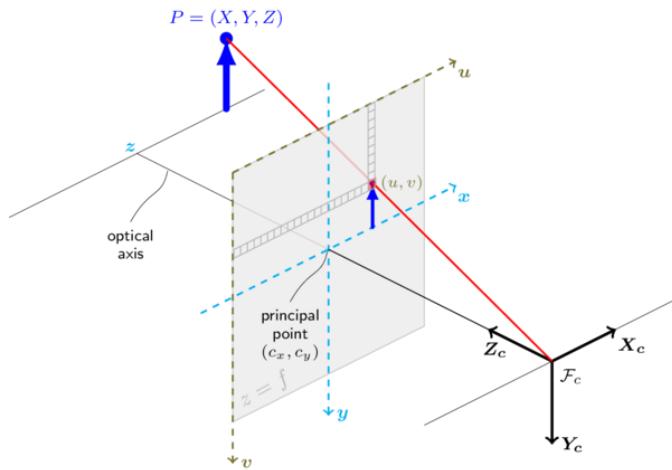


Abbildung 2.4: Lochkameramodell [OpenCV, 2018]

In einer Lochkamera fällt Licht durch eine kleine Öffnung auf eine Projektionsfläche. Es werden keine Linsen zur Bündelung des Lichtes eingestzt. Für die Projektion eines Punktes P auf diese Fläche müssen vier Bezugsysteme berücksichtigt werden:

1. Das **Weltkoordinatensystem** ist ein statioäres Bezugsystem, dessen Ursprung sich an einem beliebigen Punkt im aufgenommenen Raum befindet. Über dieses System werden Punkte $P = (X, Y, Z)$ definiert.
2. Das **Kamerakoordinatensystem** hat seinen Ursprung im Punkt F_c , der Öffnung des Kamerasytems. Es definiert Punkte $P = (x_c, y_c, z_c)$ relativ zur Kamera.
3. Das **Projektionskoordinatensystem** hat seinen Ursprung im Punkt $C = (c_x, c_y)$ der Projektionsfläche. Über es werden zweidimensionale Punkte $P' = (x_p, y_p)$ definiert.
4. Der Ursprung des **Bildschirmkoordinatensystem** ist die linke obere Ecke der Projektionsfläche. Über es werden Punkte $P' = (u, v)$ in der Einheit Pixel definiert.

Unter Berücksichtigung dieser Bezugssysteme können die sogenannten “intrinsischen” und “extrinsischen Kameraparameter” bestimmt werden.

Intrinsische Kamera- und Verzeichnungsparameter

Die intrinsischen Kamera- und Verzeichnungsparameter sind abhängig von der Bauart der Kamera und gelten daher für alle Aufnahmen, welche mit der selben Hardware-Konfiguration getätigt werden. Die Parameter c_x und c_y , welche die Bildmitte festlegen und die horizontale und vertikale Brennweite f_x und f_y definieren die intrinsischen Kameraparameter. Mit ihrer Hilfe lässt sich eine Kameramatrix der Form

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

bilden. Die Verzeichnungsparameter sind abhängig von den Linsen, welche in einer Kamera verwendet werden. Diese sorgen für radiale und tangentiale Verzerrungen des Bildes. Die Parameter k_1 , k_2 und k_3 definieren die radiale Verzeichnung und p_1 und p_2 die tangentiale Verzeichnung. [Meißner, 2007]

Die intrinsischen Kamera- und Verzeichnungsparameter können durch ein Kalibrierungsverfahren bestimmt werden. Computervision-Bibliotheken wie OpenCV bieten hierfür Hilfsfunktionen an [OpenCV, 2018].

Extrinsische Kameraparameter

Die extrinsischen Kameraparameter definieren, wie ein Punkt des 3D-Weltkoordinatensystems in einen 3D-Punkt des Kamerakoordinatensystems überführt wird. Die Transformation ist in Gleichung 2.2 beschrieben:

$$P_c = RP_w + t \quad (2.2)$$

R und t sind die extrinsischen Kameraparameter. R ist die sogenannte Rotationsmatrix, welche die Drehung der Kamera um die X, Y und Z-Achse enthält. t ist der Translationsvektor, welcher angibt wieweit der Ursprung des Weltkoordinatensystems von der Kamera entfernt ist.

Die extrinsischen Parameter einer Aufnahme lassen sich mithilfe der intrinsischen Kamera- und Verzeichnungsparameter und mindestens sechs Weltkoordinaten, welche Bildschirmkoordinaten zugeordnet werden können, bestimmen. Bibliotheken wie OpenCV bieten auch hierfür Funktionen an.

Transformation von Bildschirmpositionen in Weltkoordinaten

Die Transformation einer Weltkoordinate in eine Bildschirmposition ist, ohne Berücksichtigung der Verzeichnung, gegeben durch die Formel 2.3. s ist hier ein konstanter Skalierungsfaktor.

$$s \begin{bmatrix} u & v & 1 \end{bmatrix}^T = A \begin{bmatrix} R & t \end{bmatrix} P_w \quad (2.3)$$

Da bei der Projektion vom Weltkoordinatensystem ins Bildschirmkoordinatensystem eine Dimensionsinformation verloren geht, muss bei der Rücktransformation diese extra bestimmt oder geschätzt werden. Liegt ein Punkt auf der Ebene, welche von der X- und Y-Achse des Weltkoordinatensystems beschrieben wird, kann für dessen Höhe $z = 0$ verwendet werden. Die Position eines Punktes P_w kann so mithilfe des folgenden Gleichungssystems bestimmt werden:

$$\begin{bmatrix} x_w & y_w & 0 \end{bmatrix}^T = R^{-1} \left(s A^{-1} \begin{bmatrix} u & v & 1 \end{bmatrix}^T - t \right) \quad (2.4)$$

In der Anwendung *Vehicle-Tracker* werden auf diese Weise alle Bildschirmkoordinaten der Trajektorien in Weltkoordinaten umgewandelt.

2.3 Herausforderungen bei der Rekonstruktion von Fahrzeugtrajektorien

3 Clusteranalyse

Die Clusteranalyse (kurz Clustering) ist ein wichtiges Werkzeug zur Auswertung von Daten unterschiedlichster Art. Sie stellt dabei kein konkretes Vorgehen oder einen Algorithmus dar, sondern beschreibt ein allgemeines Problem, welches auf unterschiedlichste Weise gelöst werden kann. Eine Definition ist nachfolgend gegeben:

Definition 1 *Die Clusteranalyse ist ein Verfahren um Datenobjekte aufgrund ihrer Eigenschaften und Beziehungen untereinander so zu gruppieren, dass sich die Objekte einer Gruppe möglichst stark ähneln und sich von den Objekten anderer Gruppen möglichst stark unterscheiden. Die auf diese Art entstehenden Objektgruppen werden Cluster genannt. Foo*

Je höher die *Homogenität* in einem Cluster und die *Differenz* zwischen den Clustern, desto besser ist die gewählte Clustering Methode. Der Einsatz einer Clusteranalyse ist in vielen Anwendungsgebieten und in den unterschiedlichsten wissenschaftlichen Disziplinen sehr beliebt, um ein Verständnis für Daten zu erhalten, beziehungsweise diese, nach einer Gruppierung, sinnvoll weiter verarbeiten zu können. So kommt die Clusteranalyse unter anderem in den Feldern des maschinellen Lernens, der Mustererkennung, Bildanalyse, der Biologie (Taxonomie) oder im Bereich Data Mining zum Einsatz. [Tan et al., 2007]

Die Clusteranalyse hat viel mit dem Problem der Klassifizierung von Daten gemein, insofern sie Datenobjekten Label zuordnet. Im Gegensatz zu *überwachten Klassifizierungsansätzen*, wie dem heute populären überwachten Lernen, leiten Cluster-Algorithmen die Label allerdings alleine aus den vorhandenen Daten ab. Es kommen keine Vergleichsobjekte mit bekannten, händisch vergebenen Labels zum Einsatz. Aus diesem Grund wird die Clusteranalyse auch häufig als *unüberwachte Klassifizierung* bezeichnet. [Tan et al., 2007]

Das Konzept eines *Clusters* ist nicht genau definiert. Es existieren daher viele unterschiedliche Konzepte und Algorithmen, welche sich jeweils für andere Anwendungsfälle eignen und verschiedene Eigenschaften besitzen. Somit ist das Clustering kein selbstdärfender Prozess, welcher sich in einheitlicher Weise auf unterschiedliche Probleme anwenden lässt. Jedes Problem erfordert die individuelle und sorgfältige Auswahl eines passenden Algorithmus, eines Distanzmaßes und der richtigen Parameter. Die Bestimmung dieser geschieht iterativ und nicht selten nach dem Prinzip des *Trial and Error*. In Abbildung 3.1 ist beispielhaft ein Datensatz (links) mit – für den Menschen intuitiv ersichtlich – 7 unterschiedlichen Clustern (rechts) dargestellt. Nach [Jain, 2010] kann allerdings kein existierender Clustering Algorithmus diese alle erkennen. [Jain et al., 1999; Tan et al., 2007]

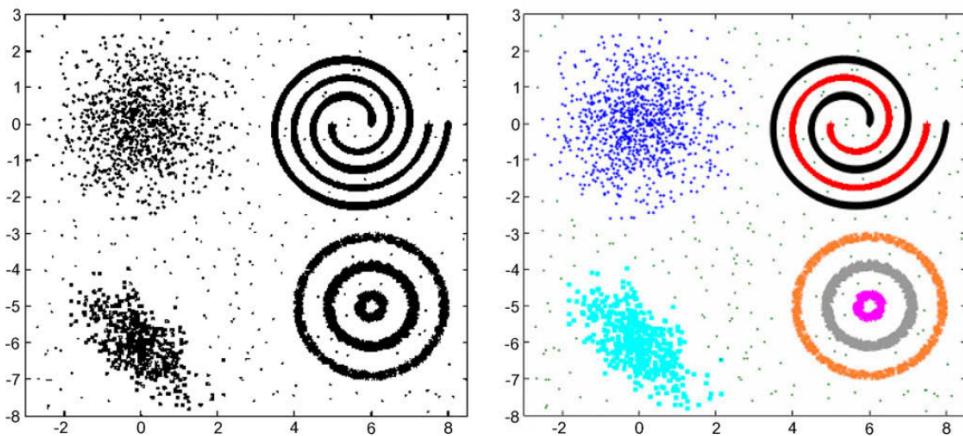


Abbildung 3.1: Rohdaten (links) und erwünschtes Clustering-Ergebnis (rechts) [Jain, 2010]

Aufgrund der Einschränkungen, welche alle Cluster-Algorithmen besitzen, muss der Analyst sich vor deren Anwendung intensiv mit den zu verarbeitenden Daten beschäftigen. Er muss ein Verständnis dafür besitzen, welche Struktur die Daten haben, beziehungsweise annehmen können, und nach welchen Mustern zu suchen ist. Besonders wichtiger ist zudem auch die Auswahl der richtigen, also relevanten, Datenmerkmale (*“Feature Selection”*) und die Wahl deren Repräsentation (*“Feature Transformation”*). Die Selektion und gegebenenfalls Transformation der Daten muss in einem Vorverarbeitungsschritt geschehen, dessen Qualität einen maßgeblichen Einfluss auf das finale Clustering-Ergebnis hat. Basierend auf vorangegangener Beschreibung und [Jain et al., 1999], lässt sich der grundlegende Ablauf einer Clusteranalyse wie folgt darstellen:

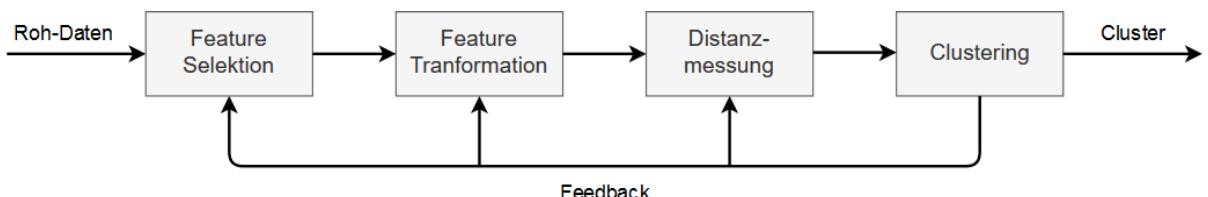


Abbildung 3.2: Ablauf einer Clusteranalyse

[Jain, 2010] nennt einige weitere Herausforderungen, welcher man sich bei der Clusteranalyse bewusst sein muss:

1. Daten können Ausreißer enthalten. Wie sollen diese behandelt werden?
2. Die Anzahl der Zielcluster ist üblicherweise nicht bekannt. Wie kann sie im voraus bestimmt werden, wenn die Analyse es erfordert?
3. Wie können gefundenen Cluster validiert werden?

3.1 Cluster-Sets und Cluster

Cluster-Sets, das heißt die Gesamtheit aller durch eine Analyse gefundenen Cluster, und einzelnen Cluster selbst, können in verschiedene Kategorien unterteilt werden, beziehungsweise unterschiedliche Eigenschaften besitzen. Nachfolgend sind die wichtigsten basierend auf [Tan et al., 2007] und [Jain et al., 1999; Jain, 2010] aufgeführt.

3.1.1 Eigenschaften von Cluster-Sets

Bei Cluster-Sets kann grundsätzlich zwischen nachfolgenden Eigenschaften unterschieden werden.

Hierarchisch vs. Partitioniert Von *hierarchischen* Cluster-Sets wird gesprochen, wenn die einzelnen Cluster verschachtelt sind und dabei eine Baum-Struktur bilden. Cluster sind hingegen *partitioniert*, wenn keine Überlagerungen zwischen ihnen existieren.

Exklusiv vs. Überlappend vs. Fuzzy *Exklusive* Cluster-Sets liegen vor, wenn jedem Datenwert ein oder kein Zielcluster zugeordnet wird. Im Gegensatz hierzu können bei *überlappenden* Cluster-Sets Objekte einer oder mehrerer Gruppen angehören. Bei dem sogenannten *Fuzzy* oder *Soft* Cluster-Sets, gehört ein Datenobjekt einem Cluster mit einer bestimmten Wahrscheinlichkeit oder Gewicht an. Algorithmen, welche Daten eine Wahrscheinlichkeit für die Zugehörigkeit zu einem Cluster zuweisen, werden *probabilistische* Cluster-Algorithmen genannt.

Komplett vs. Partielle Von *kompletten* Cluster-Sets wird gesprochen, wenn jedes Element der Eingangsdaten einem Cluster zugeordnet wird. Bei *partiellen* Sets ist dies nicht der Fall. Hier kann ein bestimmter Anteil an Datenwerten als Ausreißer markiert werden, welche keine Gruppe besitzen.

3.1.2 Eigenschaften von Clustern

Da, wie oben erwähnt, nicht klar definiert ist, was ein Cluster ausmacht, können auch diese unterschiedliche Eigenschaften besitzen. Die wichtigsten Cluster-Arten sind nachfolgend erläutert.

Klar separierte Cluster Unter *klar separierten* Clustern versteht man solche, in welchen jedes Datenelement einen geringeren Abstand zu allen anderen Elementen des Clusters hat, als zu Elementen außerhalb des Clusters. Dargestellt ist dies in Abbildung 3.3 a). Diese idealistische Definition eines Clusters ist nur dann erfüllt, wenn die in den Daten enthaltenen Cluster einen großen Abstand voneinander haben. Dies ist in der Realität allerdings selten der Fall.

Prototyp basierte Cluster Von einem *Prototyp basierten* Cluster wird gesprochen, wenn alle Elemente einer Gruppe einen geringeren Abstand zu einem Prototyp oder Referenzwert des Clusters besitzen, als zu denen anderer Gruppierungen (siehe Abb. 3.3 b)). Ein solcher Prototyp ist üblicherweise der Mittelwert der Datenelemente eines Clusters (*Centroid*).

Graphen basierte Cluster Die Definition eines *Graphen basierten* Clusters kann immer dann verwendet werden, wenn Daten als vernetzter Graph dargestellt werden. In einem solchen sind die Elemente Knoten und die Kanten repräsentieren Beziehungen zwischen ihnen. Ein Cluster in einem solchen Graphen ist definiert als Menge von Knoten, welche untereinander verbunden sind, jedoch keine Verbindungen zu Elementen außerhalb des Clusters haben. Dargestellt ist dies in Abbildung 3.3 c).

Dichte basierte Cluster *Dichte basierte* Cluster sind definiert als Regionen mit einer hohen Dichte an Objekten, welche von Regionen umgeben sind, welche eine geringe Objektdichte besitzen (siehe Abb. 3.3 d)). Elemente, welche in einer solchen Region mit geringen Dichte liegen, werden als Ausreißer interpretiert. Dichte Bereiche werden üblicherweise gefunden, indem die Nachbarschaften von Elementen untersucht werden.

Konzeptionelle Cluster Eine sehr allgemeine Definition eines Clusters ist die der *konzeptionellen* Gruppen. Hiermit ist gemeint, dass die Elemente eines Clusters einige gemeinsame Eigenschaften besitzen. Dies schließt die oben genannten Cluster-Arten mit ein, lässt sich allerdings beliebig erweitern. So sind beispielsweise in Abbildung 3.3 e) konzeptionelle Cluster dargestellt, die die Form zweier Kreise und eines Rechtecks haben. Um solche Muster erkennen zu können, benötigt ein Algorithmus ein spezielles “Verständnis” eines Clusters.

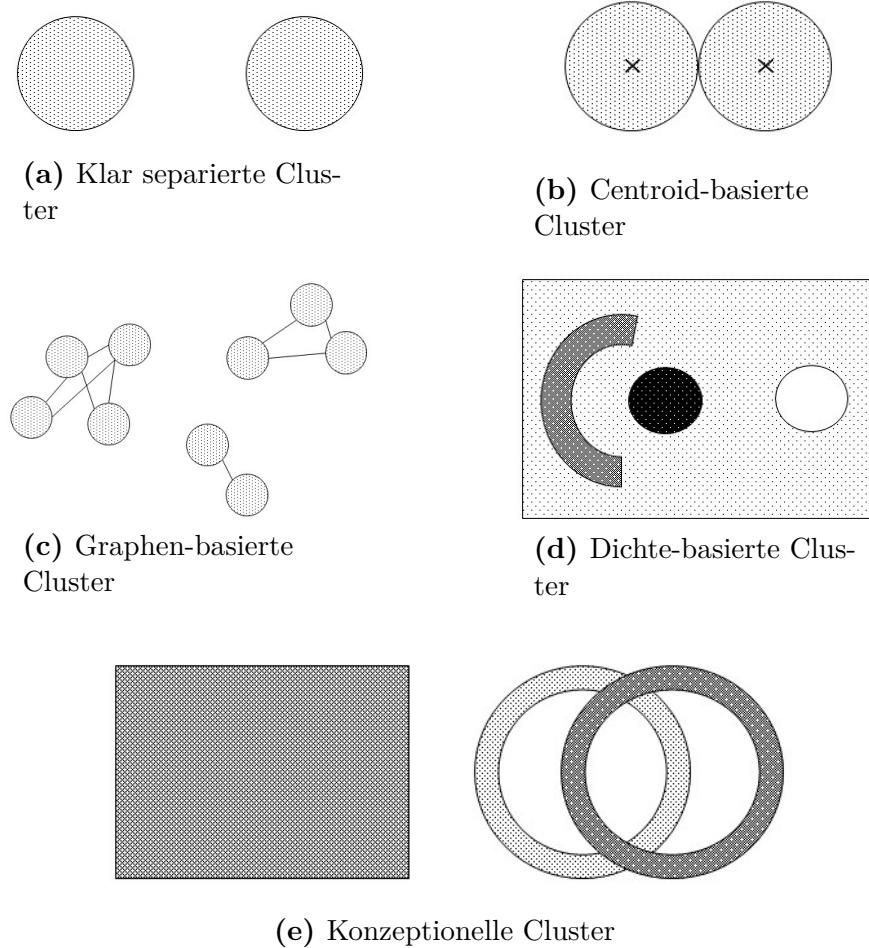


Abbildung 3.3: Visualisierung verschiedener Clusterarten (basierend auf [Tan et al., 2007])

3.2 Cluster-Algorithmen

Um mit den oben beschriebenen unterschiedlichen Cluster-Sets und Cluster Definitionen umgehen zu können, existieren verschiedene Clustering-Modelle. Einige wichtige Clustering-Ansätze sind die Vernetzungs-Modelle, Centroid-basierte-Modelle, Verteilungs-Modelle oder Dichte-Modelle. Für jedes dieser Modelle existieren unterschiedliche Algorithmen. Im Folgenden werden diese Modelle und jeweils exemplarisch ein Algorithmus, der diese vertritt, vorgestellt.

3.2.1 Vernetzungs-Modelle

Vernetzungs-Modelle werden auch häufig *hierarchische Cluster-Modelle* genannt. Sie beruhen auf der Annahme, dass Elemente, welche nahe beieinander liegen, eine höhere Gemeinsamkeit besitzen als solche, welche weiter voneinander entfernt sind. Zur Bestimmung der Nähe zwischen Elementen benötigen Vernetzungs-Modelle, wie auch andere Cluster-Modelle, eine Definition von Distanz. Diese legt ein sogenanntes *Distanzmaß* fest. Zusätzlich ist ein *Link-Kriterium* notwendig, welches bestimmt, wie genau die Entfernung zwischen zwei Clustern ermittelt wird. Übliche Link-Kriterien sind *Minimum-Linkage*, welches die minimale Distanz zwischen den Objekten der Cluster als Distanz verwendet, oder *Maximum-Linkage* beziehungsweise *Average-Linkage*. [Jain et al., 1999; George Seif, 2018]

Grundsätzlich teilen sich hierarchische Cluster-Algorithmen in zwei Gruppen auf: *Agglomerative* (Bottom-Up) und *Divisive* (Top-Down) Algorithmen. Agglomerative Ansätze weisen zu Beginn des Cluster-Vorgangs jedem Datenelement eine eigene Gruppe zu und vereinigen diese anschließend. Bei divisiven Ansätzen werden hingegen zu Beginn alle Elemente in einem Cluster zusammengefasst und diese in den nachfolgenden Schritten geteilt.

Als Beispiel wird anschließend der *agglomerative-hierarchische Cluster-Algorithmus* genauer vorgestellt. Sein Vorgehen lässt sich sehr gut anhand sogenannter Dendrogramme oder geschachtelter Cluster-Diagramme darstellen (siehe Abbildung 3.4).

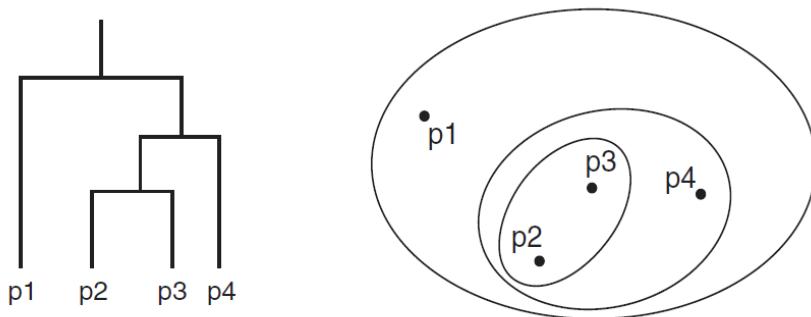


Abbildung 3.4: Agglomeratives Clustering dargestellt als Dendrogramm (links) und geschachteltes Cluster-Diagramm (rechts)

Im ersten Schritt des Algorithmus werden alle Datenpunkte als separate Cluster markiert. Diesen Schritt repräsentieren die Blätter des Dendrogramms. Anschließend muss ein Distanzmaß und ein Link-Kriterium gewählt werden. Das am häufigsten verwendete Distanzmaß ist sicherlich der euklidische Abstand, welcher die Distanz zwischen zwei Punkten oder Vektoren im n -dimensionalen Raum bestimmt. Er ist definiert durch die Formel 3.1.

$$dist(p, q) = \|q - p\|_2 = \sqrt{\sum_{i=1}^n (q_i - p_i)^2} \quad (3.1)$$

Wird als Link-Kriterium beispielsweise *Minimum-Linkage* gewählt, ist dieses definiert als:

$$link(P, Q) = \min\{dist(p, q) \mid p \in P, q \in Q\} \quad (3.2)$$

Hierbei entsprechen P und Q zwei Clustern, welche die Elemente $p \in P$ und $q \in Q$ enthalten. Auf Basis des gewählten Link-Kriteriums kann nun eine Distanz-Matrix für die einzelnen Cluster erstellt werden. Die zwei Cluster mit minimalem Abstand voneinander werden anschließend zusammengeführt und die vorherigen Schritte werden wiederholt, bis nur noch ein Cluster (Wurzel des Dendrogramms) beziehungsweise die gewünschte Clusteranzahl übrig ist. [George Seif, 2018; Tan et al., 2007]

Bei den meisten Varianten des agglomerativen Clusterings muss der Nutzer die Anzahl der Zielcluster im vorraus festlegen, was problematisch ist, da diese meist nicht bekannt ist. Umgangen werden kann dies nur, indem ein Link-Kriterium gewählt wird, das ab einer bestimmten Distanz zwischen den Clustern diese nichtmehr fusioniert [George Seif, 2018].

Die Zeitkomplexität des agglomerativen Clusterings beträgt bestenfalls $O(n^2 \log n)$, weshalb die Menge der Daten, welche mit ihm verarbeitet werden können erheblich begrenzt ist [Tan et al., 2007].

3.2.2 Prototyp-Modelle

Centroid basierte Cluster-Modelle betrachten im Gegensatz zu hierarchischen Modellen nicht die Distanz zwischen Clustern, sondern die Entfernung von Objekten zu Referenzpunkten, sogenannten *Prototypen*. Die am häufigsten verwendeten Prototypen sind *Centroids*, welche den Mittelpunkt eines Clusters darstellen. *Medoids*, welche auch häufig genutzt werden, repräsentieren hingegen den Median eines Clusters.

Ein Beispiel für einen Centroid-Cluster-Algorithmus ist *k-Means*. Dieser ist aufgrund seines Alters, seiner Einfachheit und der vielen Weiterentwicklungen wohl der bekannteste Cluster-Algorithmus überhaupt.

Das Ziel von k-Mean ist es, für eine n-dimensionale Punktmenge $X = \{x_1 \dots x_n\}$ ein Cluster-Set $C = \{c_1 \dots c_k\}$ zu finden, welches die Summe der quadratischen Abweichung (Gleichung 3.3) zwischen allein Punkten in einem Cluster und deren Mittelwert μ_k (Centroids) minimiert.

$$J(c_k) = \sum_{k=1}^K \sum_{x_i \in c_k} \|x_i - \mu_k\|^2 \quad (3.3)$$

Eine Lösung für dieses Problem zu finden, ist NP-Schwer. Aus diesem Grund ist k-Means ein approximativer Ansatz, welcher nicht garantieren kann, ein globales Minimum zu finden. Die Funktionsweise des Algorithmus ist in Abbildung 3.5 dargestellt. Die Kreuze entsprechen hierbei den Centroids, welche sich über die Iterationen hinweg verschieben.

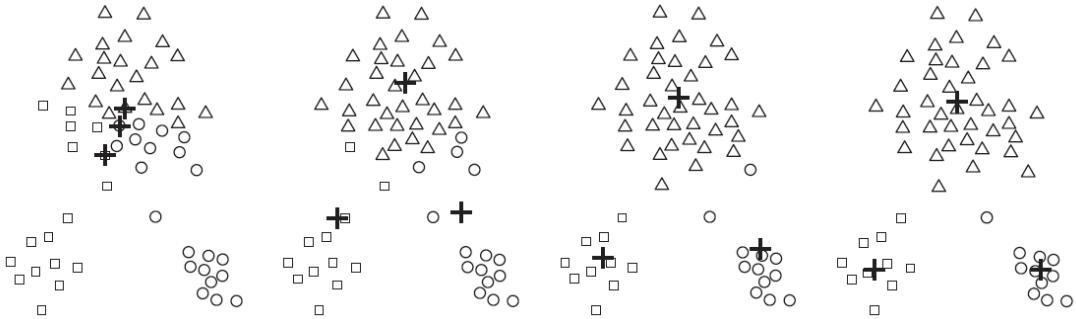


Abbildung 3.5: Funktionsweise von k-Means [Tan et al., 2007]

Ausgehend von der Punktmenge X und der gesuchten Cluster-Anzahl k , werden im ersten Schritt k zufällig positionierte Centroids μ_k definiert. Anschließend wird für alle Punkte x_i der nächstgelegene Centroid μ_j gesucht.

$$j = \arg \min(\text{dist}(x_i, \mu_j)) \quad (3.4)$$

x_i wird daraufhin Mitglied in Cluster C_j . Als Distanzmaß (dist) kann hier wieder der euklidische Abstand (Gleichung 3.1) verwendet werden oder aber auch beliebige andere sinnvolle Metriken. Nachdem alle Punkte x_i einem Cluster zugewiesen wurden, werden die Centroid Positionen neu bestimmt. Hierzu wird der Durchschnitt aller Punkte eines Clusters berechnet:

$$c_j = \frac{1}{n} \sum_{x_j \in C_j} x_j \quad (3.5)$$

Diese zwei Schritte werden mehrfach wiederholt, bis das Ergebnis konvergiert, das heißt die Zuweisungen sich nur noch geringfügig ändern. [Jain, 2010]

Der primäre Nachteil des k-Means Algorithmus ist, dass auch bei ihm die Anzahl der Zielcluster spezifiziert werden muss. Des Weiteren ist sein Ergebnis aufgrund der zufälligen Initialisierung der Centroids nicht deterministisch. Vorteil von k-Means ist hingegen, dass seine Zeitkomplexität bei $O(n)$ liegt.

Um die genannten Nachteile, zumindest in Teilen, umgehen zu können, existieren diverse Weiterentwicklungen des k-Mean Algorithmus. So stammen beispielsweise von [Hamerly et al.] und [Pelleg et al.] die Algorithmen *g-Means* beziehungsweise *x-Means*, welche die Clusteranzahl k auf Basis mehrerer k-Means Durchläufe und statistischer Kennzahlen bestimmen.

3.2.3 Distributions-Modelle

Distributions-Cluster-Modelle basieren auf der Verwendung von statistischen Wahrscheinlichkeitsverteilungen wie beispielsweise der Gauß-Verteilung. Cluster werden darüber definiert, wie wahrscheinlich es ist, dass Objekte der selben Verteilung angehören. Problematisch ist die Verwendung dieser Cluster-Methodik, da sie anfällig für das Problem des “*Overfitting*” ist, wenn die Komplexität der verwendeten Modelle nicht beschränkt wird. Zudem ist die Annahme, dass vielen realen Datensätzen ein statistisches Verteilungsmodell zugrundeliegt, gefährlich. Ist diese These jedoch berechtigt, haben die Modelle den Vorteil, dass sie neben der Zuweisung von Objekten zu Clustern auch Korrelationen zwischen einzelnen Attributen aufzeigen können. [Anders Drachen, 2014]

Nachfolgend wird der bekannteste Vertreter der Distributions-Cluster-Algorithmen vorgestellt: das *Expectation–maximization* (EM) Verfahren unter Verwendung sogenannter *Gaussian-Mixture-Models* (GMM). Die Funktionsweise des EM-Algorithmus hat grundsätzlich viel gemein mit der des k-Mean Ansatzes. Es wird ebenfalls mit einer festen Anzahl zufällig initialisierter Modelle gestartet, welche anschließend über mehrere Iterationen an die Daten angepasst werden. Im Gegensatz zu k-Means, sind die gewählten Modelle hingegen Gauß-Verteilungen, welche zwei Parameter besitzen: ihren Mittelwert und die Standardabweichung. Das Vorgehen des EM-Algorithmus ist nachfolgend, basierend auf [George Seif, 2018], beschrieben und in Abbildung 3.6 grafisch dargestellt.

- 1) Wahl der Clusteranzahl k und Initialisierung der Gauß-Modelle für die entsprechenden Cluster.
- 2) Berechnung der Wahrscheinlichkeit, dass ein Datenpunkt zu einem Cluster gehört. Je näher ein Datenpunkt dem Zentrum einer Gauß-Verteilung ist, desto höher die Wahrscheinlichkeit für dessen Zugehörigkeit.
- 3) Basierend auf den Wahrscheinlichkeiten werden die Parameter der Verteilungen neu berechnet. Hierzu wird die gewichtete Summe der Datenpunkt-Positionen errechnet. Die Gewichte entsprechen dabei den Wahrscheinlichkeiten, dass ein Element zu einem Cluster gehört. Hierdurch werden die Gauß-Modelle automatisch den in den Daten enthaltenen Clustern angepasst.
- 4) Wiederholung der Schritte 2) und 3), bis das Clustering-Ergebnis konvergiert.

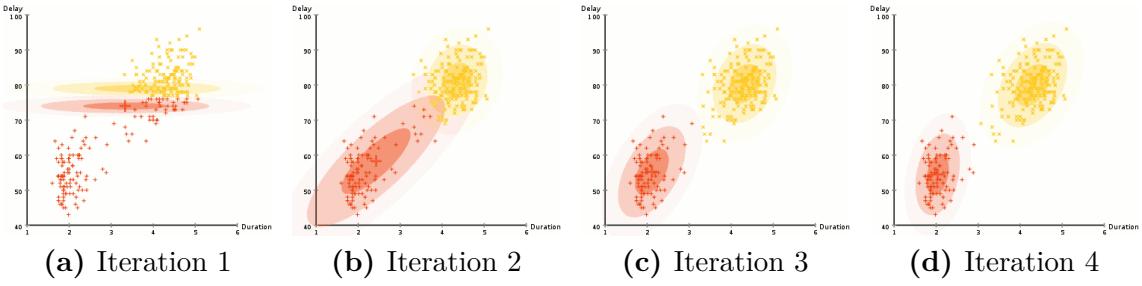


Abbildung 3.6: Darstellung des EM-Cluster-Algorithmus über mehrere Iterationen [George Seif, 2018]

Ziel des EM-Algorithmus ist es, die Parameter der Gauß-Modelle so zu optimieren, dass diese die Verteilung der Daten bestmöglich beschreiben. Am Ende des Clusterings besitzt jeder Datenwert die Zugehörigkeits-Wahrscheinlichkeiten für die einzelnen Cluster. Ein Element wird jenem Cluster zugeordnet, für welches es die höchste Wahrscheinlichkeit besitzt.

3.2.4 Dichte-Modelle

Dichte basierte Cluster sind, wie oben beschrieben, definiert als Regionen hoher Objektdichte, welche von Bereichen geringer Dichte umgeben sind. Dichte-Clustering-Modelle suchen nach eben solchen Regionen. Ein großer Vorteil der Algorithmen dieser Klasse ist, dass sie Cluster beliebiger Formen finden können, nicht auf die Vorgabe einer Clusteranzahl angewiesen sind und mit Ausreißern umgehen können.

Als Vertreter der Dichte-basierten Ansätze wird nachfolgend der *DBSCAN* Algorithmus (*Density-Based Spatial Clustering of Applications with Noise*), wie in [Gao, 2012] beschrieben, vorgestellt. Er verwendet als Maß für die Dichte einer Region die sogenannte ϵ -Nachbarschaft (*Eps*). Diese selektiert für ein Objekt p alle Objekte, welche innerhalb des Radius ϵ um dieses liegen:

$$N_\epsilon(p) = \{q \mid dist(p,q) \leq \epsilon\} \quad (3.6)$$

Eine ϵ -Nachbarschaft besitzt eine hohe Dichte, wenn in ihr mindestens *MinPts* Objekte liegen.

Basierend auf der Definition von *Eps*, werden die in einem Datensatz vorhandenen Elemente in drei Klassen unterteilt. Sie sind entweder *Kern-*, *Rand-* oder *Ausreißer-* Objekte. Ein Kernobjekt hat mindestens *MinPts* andere Punkte in *Eps*. Randobjekte besitzen weniger als *MinPts* in *Eps*, liegen aber in der Nachbarschaft eines Kernobjektes. Ausreißerobjekte sind weder Kern- noch Randobjekte.

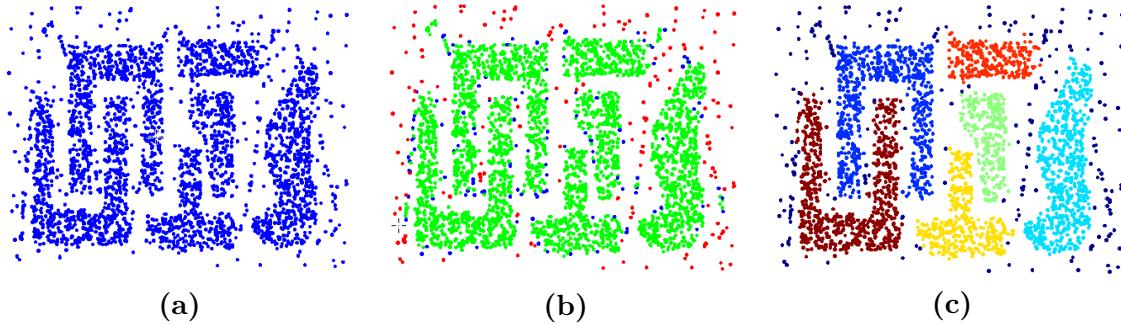


Abbildung 3.7: Schritte des DBSCAN Algorithmus, a) Rohdaten, b) Klassifizierung in Kern- (grün), Rand- (blau) und Ausreißer- (rot) Punkte, c) Cluster Ergebnis [Gao, 2012]

Auf Basis der drei Objektklassen, lässt sich das Prinzip der dichte-basierten *Erreichbarkeit* definieren. Ein Objekt q ist von p *direkt* erreichbar, wenn p ein Kernobjekt ist und q in dessen Eps liegt. In Abbildung 3.8 gilt dies beispielsweise für p und p_2 . Zwei Elemente sind *indirekt* erreichbar, wenn sie über eine Reihe von Zwischenschritten (direkte Relationen) verbunden sind (transitiv). Dies ist in Abbildung 3.8 für q und p der Fall.

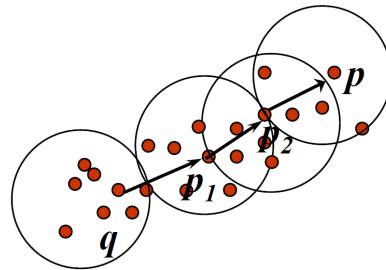


Abbildung 3.8: Erreichbarkeit in DBSCAN [Gao, 2012]

Der DBSCAN Algorithmus lässt sich, basierend auf den obigen Definitionen, informell wie folgt beschreiben:

- 1) Unterteilung der Objekte in die drei Objektklassen. (Abb. 3.7 b))
- 2) Aussortierung der Ausreißer-Objekte.
- 3) Wahl eines nicht zugewiesenen Kernobjektes.
- 4) Erstellung eines neuen Clusters für das Kernobjekt und alle von ihm ausgehend direkt oder indirekt erreichbaren Objekte.
- 5) Wiederholung der Schritte 3) und 4), bis alle Kern- und Randobjekte einem Cluster zugewiesen sind. (Abb. 3.7 c))

DBSCAN besitzt die oben beschriebenen Vorteile Dichte-basierter Cluster-Algorithmen. Dank einer Zeitkomplexität von $O(n \log n)$ kann er außerdem auch auf große Datensätze angewendet werden. Nachteil des Ansatzes ist hingegen, dass er schlecht mit Clustern umgehen kann, welche unterschiedliche Dichten besitzen.

3.3 Distanzmaße zum Vergleich von Fahrzeugtrajektorien

Bei der Clusteranalyse ist neben der Wahl des passenden Cluster-Algorithmus insbesondere die Entscheidung, welches Distanzmaß verwendet wird, ausschlaggebend. Im obigen Abschnitt wurden bereits die euklidsche Distanz (Gleichung 3.1) als ein mögliches Distanzmaß definiert. Dieses kann jedoch nur zur Bestimmung der Distanz zwischen n -dimensionalen Punkten im euklidschen Raum verwendet werden. Dies gilt ebenso für andere einfache Maße wie die Manhatten-Distanz oder die Pearson-Distanz.

Um Fahrzeugtrajektorien korrekt gruppieren zu können, ist ein Distanzmaß notwendig, welches je nach Anforderungen die unterschiedlichen Aspekte der Trajektorien vergleicht. Häufig werden die Eigenschaften Lage, Form und Länge hierzu herangezogen. In der Literatur werden diverse Maße zum Vergleich von Trajektorien vorgestellt. Diese besitzen alle unterschiedliche Eigenschaften, Vor- und Nachteile.

Nachfolgend werden exemplarisch drei Distanzmaße vorgestellt, anhand welcher ersichtlich ist, welche Abwägungen bei der Wahl des Maßes gemacht werden müssen. In allen drei Fällen werden die Trajektorien als Reihen 2-dimensionaler Punkte mit Länge n interpretiert: $t_i = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$. Der n -te Punkt einer Trajektorie ist gegeben über $t_i(n)$ und deren Punkt-Länge über $\text{len}(t_i)$. Die Menge der zu vergleichenden Trajektorien ist $T = \{t_1, t_2, \dots, t_m\}$. Abbildung 3.9 zeigt eine Auswahl möglicher Trajektorien.

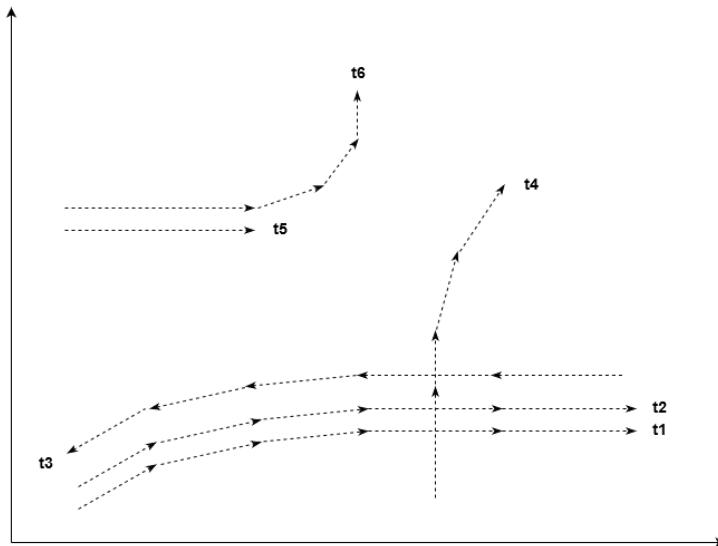


Abbildung 3.9: Trajektorien im 2-dimensionalen Raum

3.3.1 HU-Distanz

Die HU-Distanz wurde erstmals in der Arbeit “*Similarity based vehicle trajectory clustering and anomaly detection*” von [Hu et al., 2005] verwendet. Es ist ein sehr einfaches Distanzmaß, welches auf der mittleren euklidischen Distanz zwischen zwei Trajektorien basiert. Berechnet wird die HU-Distanz für zwei Trajektorien t_1 und t_2 wie folgt:

$$D_{HU}(t_1, t_2) = \frac{1}{N} \sum_{n=1}^N dist(t_1(n), t_2(n)) \quad (3.7)$$

$$\text{wobei } N = \min(\text{len}(t_1), \text{len}(t_2)) \quad (3.8)$$

Aus dieser Formel lassen sich sowohl die Vor- als auch Nachteile der HU-Distanz ableiten. Der klare Vorteile der HU-Distanz ist deren Einfachheit und die Effizienz von $O(n)$. Nachteil ist hingegen, dass das Distanzmaß nur gut funktioniert, wenn die Trajektorien bestimmte Kriterien erfüllen. So sollten Trajektorien, welche einem Cluster angehören, auch immer möglichst auf selber Höhe beginnen, damit deren mittlerer Abstand nicht, aufgrund einer Verschiebung, erhöht wird. Außerdem ist es notwendig, die Abstände zwischen den Punkten der Trajektorien auf die selbe Länge zu bringen, damit beim paarweisen Vergleich immer Elemente verglichen werden, welche gleichweit vom Start der Spuren entfernt sind. Diese Eigenschaften der Trajektorien müssen über einen Vorverarbeitungsschritt geschaffen werden. Problematisch bei der Verwendung der HU-Distanz ist außerdem, dass beim Vergleich zweier Trajektorien immer nur die ersten N Punkte (s. Gleichung 3.8) betrachtet werden. Die kann dazu führen, dass zwei Trajektorien, welche zu Beginn fast identisch sind und später auseinanderlaufen, trotzdem einen hohen Ähnlichkeitswert besitzen (siehe t_5 und t_6 in Abbildung 3.9).

Die HU-Distanz kann aufgrund der genannten Einschränken nur in speziellen Fällen oder unter Verwendung eines Vorverarbeitungsschrittes angewandt werden. Sie liefert ansonsten schlechte Clustering Ergebnisse.

3.3.2 Hausdorff-Distanz

Die Hausdorff-Distanz ist ein komplexeres Maß zur Bestimmung der Ähnlichkeit zwischen zwei Trajektorien. Sie misst grundsätzlich den Abstand zwischen zwei nicht-leeren, ungeordneten Teilmengen A und B und ist für Trajektorien definiert über die Gleichungen [Atev et al., 2010]:

$$D_{HD}(t_1, t_2) = \max(h(t_1, t_2), h(t_2, t_1)) \quad (3.9)$$

$$h(t_1, t_2) = \max_{i \in t_1} \min_{j \in t_2} dist(i, j) \quad (3.10)$$

$h(t_1, t_2)$ wird als gerichtete Hausdorff-Distanz *von* t_1 *nach* t_2 bezeichnet. Sie findet die maximale Distanz einer Trajektorie zum nächsten Punkt der anderen Trajektorie [Huttenlocher et al., 1993]. Da h gerichtet ist, gilt $h(t_1, t_2) \neq h(t_2, t_1)$. Aus diesem Grund wird die Hausdorff-Distanz *zwischen* zwei Trajektorien mittels D_{HD} bestimmt. $dist$ kann ein beliebiges Maß für die Distanz zweier Punkte sein, wie beispielsweise die euklidische Distanz. Grundsätzlich lässt sich über die Hausdorff-Distanz die Form zweier Trajektorien vergleichen. Diese sind ähnlich, wenn jeder Punkt einer Trajektorie einen nahegelegenen Punkt in der Vergleichsbahn besitzt.

Vorteil der Hausdorff-Distanz im Vergleich zur HU-Distanz ist, dass diese immer vollständige Trajektorien vergleicht und nicht nur Teile. Außerdem ist bei ihrer Verwendung keine Vorverarbeitung in Form von Resampling et cetera notwendig. Problematisch ist das Distanzmaß hingegen, da es mit ungeordneten Sets arbeitet und somit im Fall von Trajektorien, deren Orientierung nicht beachtet. Zwei parallel aber in entgegengesetzte Richtungen laufende Trajektorien, wie beispielsweise die Trajektorien t_2 und t_3 in Abbildung 3.9, würden nach Hausdorff daher eine hohe Ähnlichkeit besitzen. Zudem kann das Distanzmaß schlecht mit Ausreißern umgehen, da bereits ein einzelner dieser Punkte, bei ansonsten identischen Trajektorien, zu einer beliebig kleinen Ähnlichkeit führen kann. Von Nachteil ist auch, dass die Zeitkomplexität der Hausdorff-Distanz bei $O(n m)$ liegt.

3.3.3 Longest-Common-Subsequence

Das *Longest-Common-Subsequence* (LCSS) Distanzmaß basiert auf dem allgemeinen Problem der Findung einer längsten gemeinsamen Subsequenz zwischen zwei Sequenzen. Da Trajektorien, nach obiger Definition, lediglich Punktfolgen sind, lässt sich das Verfahren sehr gut auf diese anwenden. Aufgrund einiger kleiner Erweiterungen des Basisalgorithmus, besitzt das LCSS Distanzmaß einige besondere Eigenschaften. Der LCSS Algorithmus für Trajektorien ist grundsätzlich wie folgt definiert [Vlachos et al., 2002]:

$$LCSS_{\epsilon, \delta}(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 \text{ or } t_2 \in \emptyset \\ 1 + LCSS_{\epsilon, \delta}(t'_1, t'_2) & \text{if } dist(t_1(n), t_2(m)) < \epsilon \\ & \wedge |n - m| \leq \delta \\ max(LCSS_{\epsilon, \delta}(t'_1, t_2), LCSS_{\epsilon, \delta}(t_1, t'_2)) & \text{otherwise} \end{cases} \quad (3.11)$$

Hierbei gilt $t'_1 = \{t_1(0), \dots, t_1(n-1)\}$. Die Parameter ϵ und δ bestimmen das Vergleichs-Verhalten des Algorithmus. Über ϵ wird definiert, wieweit zwei Punkte maximal voneinander entfernt liegen können, um immer noch als “übereinstimmend” zu gelten. δ bestimmt hingegen, wieweit man in beide zeitliche Richtungen sucht, um einen übereinstimmenden Punkt zu finden. Da die obige LCSS Funktion nur ein diskretes Zählmaß definiert, ist das eigentliche LCSS-Distanz üblicherweise gegeben als [Vlachos et al., 2002]:

$$D_{LCSS}(\delta, \epsilon, t_1, t_2) = 1 - \frac{LCSS_{\delta, \epsilon}(t_1, t_2)}{\min(\text{len}(t_1), \text{len}(t_2))} \quad (3.12)$$

Vorteile der LCSS Ähnlichkeitdefinition sind, dass sie mit kompletten Trajektorien arbeitet und robust gegenüber Ausreißern ist, da nicht für alle Punkte Übereinstimmungen in den Trajektorien gefunden werden müssen. Über ϵ und δ kann die "Strenge" des Algorithmus geregelt werden. Zudem berücksichtigt das LCSS Maß die Orientierung der Trajektorien, solange δ nicht zu hoch gewählt wird. Die rekursive Definition des LCSS Algorithmus aus Gleichung 3.11 lässt sich mittels dynamischer Programmierung mit Zeitkomplexität $O(n m)$ berechnen.

Übersicht Distanzmaße

Anhand der drei ausgewählten und oben exemplarisch beschriebenen Distanzmaße, ist bereits ersichtlich, dass die Wahl eines passenden Maßes nicht trivial ist. Es muss die Qualität und Form der Daten berücksichtigt werden und abgewogen werden, in wieweit es möglich beziehungsweise gewünscht ist, die Daten vorzuverarbeiten. Das primäre Auswahlkriterium ist allerdings natürlich die situationsabhängige Definition von "Distanz": Sind sich Trajektorien ähnlich, wenn sie lediglich die selbe Form haben und ansonsten irgendwo im Raum liegen? Sind sie sich ähnlich, wenn sie die selbe Form haben und im Raum nahe beieinander liegen? Ist ihre Orientierung relevant? Dies sind wichtige Fragen, welche vor der Wahl eines Distanzmaßes geklärt werden müssen. Da die Maße als Distanzfunktionen bei der Clusteranalyse verwendet werden, ist ihr Verhalten ausschlaggebend für den Erfolg der Gruppierung von Trajektorien.

Wichtige Eigenschaften einiger in der Literatur häufig verwendeten Vergleichsmaße, inklusive der drei oben beschriebenen, sind nachfolgend nochmals in tabellarischer Form festgehalten.

Tabelle 3.1: Eigenschaften verschiedener Distanzmaße

Distanzmaß	gerichtet	PreProc. nötig	Ausreißer-resistant
HU [Hu et al., 2005]	✓	✓	✓
PCA [Bashir et al., 2003]	✓	✓	✓
DTW [Keogh and Pazzani, 2000]	✓	✗	✗
HD [Chen et al., 2011]	✗	✗	✗
mod. HD [Atev et al., 2006]	✓	✗	✓
PF [Piciarelli and Foresti, 2006]	✓	✗	✗
LCSS [Vlachos et al., 2002]	✓	✗	✓

4 Verwandte Arbeiten

Das folgende Kapitel gibt eine Überblick über wichtige und interessante wissenschaftliche Arbeiten, welche sich mit der Analyse von Trajektoriedaten und insbesondere Fahrzeugtrajektorien beschäftigen. Zu Beginn werden diverse Arbeiten vorgestellt, welche sich mit der Clusteranalyse von Trajektorien befassen. Anschließend wird betrachtet, wie in der Literatur die Erkennung von Fahrspuren, vorzugsweise auf Basis von Trajektorien, umgesetzt wird. Am Ende des Kapitels werden Defizite der existierenden Lösungen festgehalten und analysiert, welche spezifischen Neuerungen für die Umsetzung dieser Arbeit nötig sind.

4.1 Clusteranalyse von Trajektorien

Aufgrund der großen Menge an Informationen, welche sich auf Basis von Trajektoriedaten ermitteln lassen, ist ihre Analyse schon seit geraumer Zeit Gegenstand wissenschaftlicher Untersuchungen. Nachfolgend werden einige Arbeiten vorgestellt, welche sich mit der Clusteranalyse von Trajektorien beschäftigen. Die Auswahl zeigt prototypisch, wie unterschiedliche die Anwendungsszenarien und Ziele bei solchen Analyse sind.

Similarity based vehicle trajectory clustering and anomaly detection

Eine Arbeit, welche ein sehr typisches Anwendungszenario behandelt, stammt von [Hu et al., 2005]. Die Autoren beschreiben in dieser Veröffentlichung ein Verfahren zur Clusteranalyse von Fahrzeugtrajektorien. Ziel dieser ist es, auf Basis der entdeckten Spur-Cluster, anormale Verkehrsmanöver in Live-Aufnahmen von Straßenabschnitten detektieren zu können. Solche Manöver sind beispielsweise “Fahren abseits der üblichen Bahnen” oder “zu schnelles/langsames Fahren”. Die Fahrzeugtrajektorien sind in dieser Arbeit als Sequenzen zweidimensionaler Punkte definiert. Um sie zu gruppieren, setzen Hu et al. auf klassische Clusterverfahren und die Verwendung eines einfachen, metrischen Distanzmaßes. Dieses Maß, bekannt als HU-Distanz (siehe Abschnitt 3.3.1), vergleicht Trajektorien über den mittleren Abstand zwischen zusammengehörigen Punktpaaren. Da dies nur zuverlässig möglich ist, wenn die Trajektorien einige Bedingungen erfüllen, müssen die Autoren diese vorverarbeiten. Sie vereinheitlichen daher die Abstände der Punkte einer Trajektorie und erweitern sie zudem in Richtung der Szenen-Grenzen.

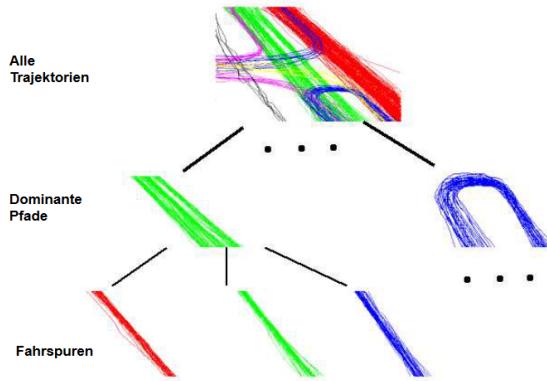


Abbildung 4.1: Zweistufiger Clustering-Vorgang von [Hu et al., 2005]

Unter Verwendung des definierten Distanzmaßes werden die Trajektorien in einem zweistufigen Verfahren verarbeitet. In den zwei Phasen werden, wie in Abbildung 4.1 dargestellt, zuerst dominante Fahrpfade extrahiert, welche anschließend weiter in einzelne Fahrspuren untergliedert werden. Als eigentliche Cluster-Algorithmen vergleichen die Autoren den *Spectral-Clustering* Ansatz [Ng et al., 2002] mit einem *Fuzzy-k-Means* Verfahren [Xie and Beni, 1991]. Die Untersuchungen zeigen, dass der Spectral Clustering Ansatz nicht nur bessere Ergebnisse liefert, sondern dieser über mehrere Durchläufe hinweg auch stabil sind, wohingegen die Resultate des Fuzzy-Ansatzes variieren.

Multi Feature Path Modeling for Video Surveillance

Eine weitere Arbeit welche das Ziel hat, anormale Bewegungsmuster auf Basis von Trajektorien zu entdecken, stammt von [Junejo et al., 2004]. In diesem Fall geht es den Autoren allerdings nicht um das Finden von Fahrzeug-Fahrspuren, sondern um die Extraktion von Laufpfaden von Fußgängern. Die Bewegungsbahnen der Passanten werden aus Aufnahmen stationärer Überwachungskameras gewonnen und als zwei-dimensionale Punktreihen repräsentiert. Zum Vergleich der Trajektorien, verwenden die Autoren die Hausdorff-Distanz als Distanzmaß. Die üblicherweise negativen Eigenschaften dieses Vergleichkriteriums (siehe Abschnitt 3.3.2), konkret die Missachtung der Trajektorie-Orientierung, sind bei diesem Anwendungsfall kein Nachteil, sondern gewünscht. Da Fußgänger auf einem Pfad oder Weg in entgegengesetzte Richtungen gehen können, muss die Orientierung ihrer Trajektorien ignoriert werden. Auf Basis der Hausdorff-Distanz erstellen Junejo et al. einen vollständigen Graphen, in welchem die Knoten Trajektorien und die gewichteten Kanten den Distanzen zwischen Trajektorien entsprechen. Sie zerlegen diesen Graphen mit Hilfe eines rekursiven *min-cut*-Graphen-Algorithmus, welcher sich an der Arbeit von [Boykov and Kolmogorov, 2004] orientiert, und erhalten so die Cluster für die extrahierten Fußgänger-Trajektorien.

Clustering of Vehicle Trajectories

In [Atev et al., 2010] ist das Ziel der Autoren, ein Verfahren zu finden, mit welchem Fahrzeugtrajektorien bestmöglich gruppiert werden können, ohne diese im Voraus anpassen zu müssen. Sie vergleichen hierzu die Performance von drei unterschiedlicher Distanzmaße unter Verwendung von zwei Cluster-Algorithmen. Primäres Augenmerk legen die Autoren auf ein von ihnen bereits in [Atev et al., 2006] entwickeltes Distanzmaß, welches auf der Hausdorff-Distanz basiert und sowohl die Orientierung von Trajektorien berücksichtigt als auch robust gegenüber Ausreißern ist. Dieses neue Maß ist für zwei Trajektorien P und Q wie folgt definiert:

$$h_{\alpha, N, C}(P, Q) = \text{ord}_{p \in P}^{\alpha} \left\{ \min_{q \in N_Q(C_{P, Q}(p))} d(p, q) \right\} \quad (4.1)$$

Hierbei entspricht $C_{P, Q}$ einem Mapping $P \rightarrow Q$, welches einem Punkt $p \in P$ einen entsprechenden Punkt $q \in Q$ zuweist, welcher die selbe relative Position in Q besitzt wie p in P . N_Q definiert ein Subset von Q als Nachbarschaft des Punktes q . Zusammen definieren N_Q und $C_{P, Q}$ eine Struktur, in welcher der Abgleich der Trajektorien stattfindet. Dieses Vorgehen wird in Abbildung 4.2 visualisiert. Der Operator $\text{ord}_{p \in P}^{\alpha} f(p)$ selektiert jenen Wert aus $f(p)$, welcher größer ist als α -Prozent der Werte.

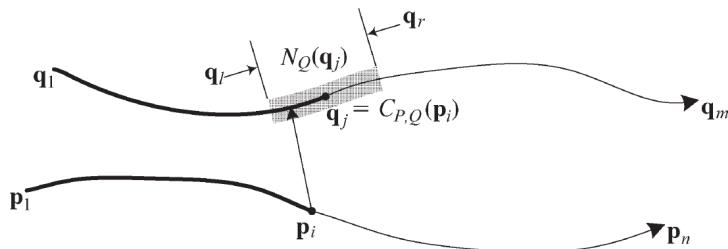


Abbildung 4.2: Funktionsweise der modifizierten Hausdorff-Distanz [Atev et al., 2010]

Dank dieser Modifizierungen eignet sich das Distanzmaß gut für den Vergleich von Trajektorien: $C_{P, Q}$ sorgt für den Einbezug der Orientierungen und über die Nachbarschaft N_Q und $\text{ord}_{p \in P}^{\alpha} f(p)$ kann der Einfluss von Ausreißern minimiert werden.

Dieses Distanzmaß vergleichen Atev et al. unter Verwendung eines Spectral und eines Agglomerativen Cluster-Algorithmus mit der *Longest-Common-Subsequence* (LCSS) und der *Dynamic-Time-Warping*-Distanz (DTW). Die Ergebnisse der Untersuchungen für vier verschiedene Datensätze zeigen, dass die beste Cluster-Performance mit Hilfe der modifizierten Hausdorff-Distanz und des Spectral-Clustering erreicht wird. Unter Verwendung der LCSS und DTW Distanzmaße, könnten die Autoren nicht die selben Resultate erzielen.

Dass das von Atev et al. vorgeschlagene Distanzmaß sehr gute Clusterergebnisse produziert, wurde auch von [Morris and Trivedi, 2009] bestätigt. In ihrer Untersuchung waren allerdings die Ergebnisse, welche mithilfe des LCSS Maßes erreicht wurden, ebenso gut oder teilweise besser.

Clustering of trajectories based on Hausdorff Distance

Eine weitere interessante Arbeit zur Clusteranalyse von Trajektorien stammt von [Chen et al., 2011]. Die Autoren haben das Ziel, Muster in den Bewegungsbahnen von Hurrikans, welche im Zeitraum von 1850 bis 2010 über den Atlantik zogen, zu erkennen. Sie verwenden hierzu einen angepassten DBSCAN Cluster-Algorithmus und das Hausdorff-Distanzmaß. Um die Missachtung der Orientierung kompensieren zu können, und zudem auch Ähnlichkeiten in Sub-Trajektorien zu erkennen, wählen die Autoren eine etwas andere Darstellung der Trajektorien. Sie definieren eine Bewegungsbahn als eine Folge sogenannter „*Flow-Vektoren*“, welche neben Positions- auch Richtungsinformationen enthalten. Ein solcher Vektor ist definiert über:

$$f_i = (x_i, y_i, dx_i, dy_i) \quad (4.2)$$

wobei gilt:

$$dx_i = (x_{i+1} - x_i) / \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad (4.3)$$

$$dy_i = (y_{i+1} - y_i) / \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad (4.4)$$

Die Distanz zwischen zwei *Flow-Vektoren* ist ihr euklidischer Abstand. Auf diese Weise wird bei der Berechnung der Hausdorff-Distanz (siehe Abschnitt 3.3.2) auch die Richtung der Trajektorien berücksichtigt. Um ähnliche Sub-Trajektorien entdecken zu können, teilen Chen et al. die Trajektorien an den Positionen „charakteristischer“ Vektoren. Diese beschreiben Richtungsänderungen in einer Bewegungsbahn und werden identifiziert über die Abweichungen in den Richtungskomponenten zweier aufeinanderfolgender Flow-Vektoren. Dies ist anschaulich in Abbildung 4.3 dargestellt.

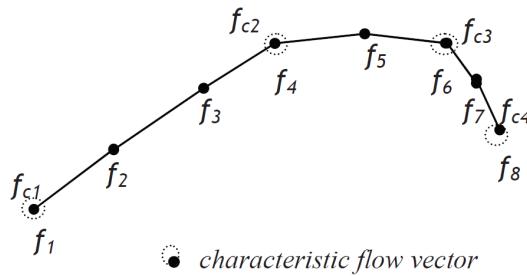


Abbildung 4.3: Zerlegung einer Trajektorie in Sub-Trajektorien [Chen et al., 2011]

Die auf diese Weise erhaltenen Sub-Trajektorien, werden von den Autoren mittels eines DBSCAN Algorithmus gebündelt. Sie können so die üblichen Bewegungsbahnen von Hurrikans über dem Atlantik bestimmen.

Discovering Similar Multidimensional Trajectories

Die Arbeit [Vlachos et al., 2002] thematisiert nicht direkt die Clusteranalyse von Trajektorien sondern beschäftigt sich mit dem Vergleich von Bewegungsbahnen im drei-dimensionalen Raum. Konkret ist ihr Ziel, Trajektorien vergleichen zu können, welche etwa die Handbewegungen beim Ausführen von Zeichensprache beschreiben. Hierzu definieren die Autoren erstmals die Grundversion des LCSS Distanzmaßes, welches in vielen Arbeiten, unter anderem in [Atev et al., 2006], [Buzan et al., 2004] und [Chen et al., 2005] zum Einsatz kommt. Auf dessen Basis erstellen sie ein Distanzmaß, welche es ermöglicht formgleiche aber im Raum verschobene Trajektorien zu finden. Die Grundversion der LCSS-Distanz und ein darauf basierendes, einfaches Distanzmaß ist, nach Vlachos et al., bereits in Abschnitt 3.3.3 vorgestellt worden. Dieses Maß erweitern die Autoren zudem wie folgt:

$$D_{LCSS}(\delta, \epsilon, A, B) = 1 - \max_{f_{c,d} \in F} D_{LCSS}(\delta, \epsilon, A, f_{c,d}(B)) \quad (4.5)$$

Hierbei ist F eine Menge von Translations-Funktionen, welche die Trajektorien entlang der Achsen verschieben. Sie besitzen die Form

$$f_{c,d}(A) = ((a_{x,1} + c, a_{y,1} + d), \dots, (a_{x,n} + c, a_{y,n} + d)) \quad (4.6)$$

Abbildung 4.4 veranschaulicht die Funktionsweise des Distanzmaßes. Es eignet sich immer dann, wenn Trajektorien mit ähnlicher Form gefunden werden sollen, welche zudem eine gewisse räumliche Verschiebung aufweisen können. Diese kann über die Größe von F gesteuert werden.

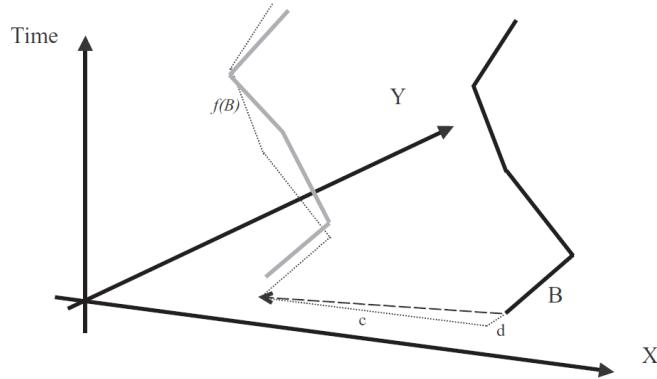


Abbildung 4.4: Verschiebung einer Trajektorie im Raum [Vlachos et al., 2002]

Lane Detection in Video-Based Intelligent Transportation Monitoring via Fast Extracting and Clustering of Vehicle Motion Trajectories

[Ren et al., 2014] stellen in ihrer Arbeit ein Vorgehen zur Clusteranalyse von Trajektorien vor, welches auf der *Rough-Set*-Theorie beruht. Sie extrahieren Fahrzeugpositionen aus Aufnahmen stationärer Überwachungskameras und stellen diese, wie die meisten Autoren, als Sequenzen zweidimensionaler Punkte dar. Ihr Ziel ist anschließend, anhand einer Gruppierung der gewonnenen Fahrzeugtrajektorien, die Spurmittelpunkte der Fahrbahnen zu bestimmen. Da ein nicht unerheblicher Anteil der Trajektorien Spurwechselvorgänge enthält, welche eine Extraktion der Mittellinien erschweren, verwenden Ren et al. einen iterativen *Rough-k-Means* Algorithmus zur Gruppierung der Trajektorien. Hierbei wird jedes Cluster über eine obere und untere Approximation beschrieben. Die untere Approximation enthält dabei die Trajektorien, welche eindeutig der Spur zugeordnet werden können. Die obere Näherung hingegen auch jene, welche Spurwechsel et cetera beschreiben. Bei der Berechnung der Spurmittten, werden die Trajektorien der unteren Approximation höher gewichtet, als die der oberen. Ein sehr ähnlicher Cluster-Ansatz wurde bereits in [Lingras et al., 2004] vorgestellt. Die initialen Mittellinien bestimmen die Autoren anhand einer *Aktivitäts-* oder *Heat-Map*, welche sie während der Extraktion der Fahrzeugpositionen erstellen. Die Clusteranzahl k muss händisch definiert werden.

Als Maß für die Distanz zwischen einer Trajektorie A_x und einer Spurmitte c_i verwenden Ren et al. die Hausdorff-Distanz $h(A_x, c_i)$. Für eine Trajektorie wird somit die nächste Mittellinie wie folgt gefunden:

$$h(A_x, c_m) = \min_{i=1 \dots k} h(A_x, c_i) \quad (4.7)$$

Hieraus ergibt sich die nachfolgende Definition für die Zuordnung der Bewegungsbahnen zu den Cluster-Näherungen:

$$\begin{cases} A_x \in \overline{C_m} \wedge A_x \in \overline{C_j} & \text{if } j \neq m \wedge \frac{h(A_x, c_j)}{h(A_x, c_m)} \leq \lambda \\ A_x \in \underline{C_m} & \text{otherwise} \end{cases} \quad (4.8)$$

Für den Grenzwert λ gilt $1 \leq \lambda \leq 1.5$. $\overline{C_m}$ und $\underline{C_m}$ entsprechen der oberen und unteren Näherung des m -ten Clusters und $\underline{C_m} \subseteq \overline{C_m}$. Nachdem in jeder Iteration des Cluster-Vorgangs die Näherungen auf diese Weise bestimmt wurden, werden die neuen Mittellinien anhand Gleichung 4.9 errechnet.

$$c_i = \begin{cases} \frac{w_l \sum_{A_x \in \underline{C_i}} A_x}{|\underline{C_i}|} + \frac{(1-w_l) \sum_{A_x \in (\overline{C_i} - \underline{C_i})} A_x}{|\overline{C_i} - \underline{C_i}|} & \text{if } \overline{C_i} \neq \underline{C_i} \\ \frac{\sum_{A_x \in \underline{C_i}} A_x}{|\underline{C_i}|} & \text{otherwise} \end{cases} \quad (4.9)$$

Als Gewichtungen w_l verwenden Ren et al. Werte im Bereich $[0.5, 1]$. $|\cdot|$ entspricht hier der Kardinalität einer Menge.

Unter Verwendung dieser Clustering-Methode ist es den Autoren von [Ren et al., 2014] möglich, auch bei einer hohen Anzahl von Ausreißern und Spurwechselvorgängen, stabile Spurmittellinien zu bestimmen. Ergebnisse, welche dies zeigen, sind in Abbildung 4.5 dargestellt.

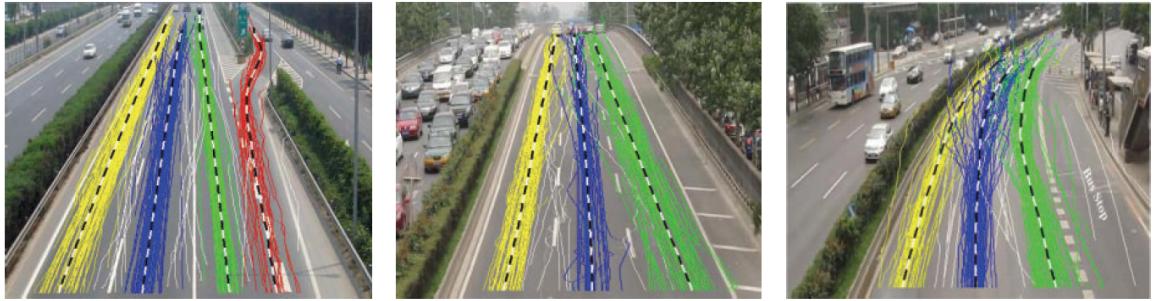


Abbildung 4.5: Ergebnisse der Spurmittellinien-Erkennung [Ren et al., 2014]

4.2 Erkennung und Definition von Fahrspuren

Primäres Ziel dieser Arbeit ist es, Fahrspuren zuverlässig in Videoaufnahmen zu erkennen zu. Dieser Abschnitt geht daher auf Arbeiten mit ähnlichen Ziel ein.

Die meisten Veröffentlichungen in diesem Bereich, löst das Problem, indem sie nach visuellen Merkmalen, primär Spurtrennlinien, in Videoaufnahmen suchen. Die Aufnahmen werden dabei entweder von stationären Kameras, von bemannten oder unbemannten Luftfahrzeugen oder einem Fahrzeug selbst erstellt. Zur Extraktion der Merkmale werden üblicherweise Methoden aus den Gebieten des maschinellen Sehen (CV) oder maschinellen Lernens (ML) verwendet. Arbeiten, welche CV-basierte Ansätze verfolgen, stammen beispielsweise von [Lai and Yung, 2000], [McCall and Trivedi, 2006] oder [Aly, 2008]. ML-gestützte Arbeiten wurden dahingegen unter anderem von [Kim, 2008] und [Gopalan et al., 2012] veröffentlicht.

Da oben genannte Ansätze, wie bereits zu Beginn der Arbeit erläutert, aufgrund von Verdeckungen oder Änderungen in der Belichtung problematisch sind, werden nachfolgend ausschließlich Arbeiten vorgestellt, welche Fahrspuren aus Trajektoriedaten extrahieren. Hierbei setzten die meisten als ersten Schritt auf eine Clusteranalyse von Trajektorien. In der Repräsentation und Extraktion der Fahrspuren variieren die Ansätze hingegen.

A System for Learning Statistical Motion Patterns

Die Arbeit [Weiming Hu et al., 2006] basiert auf dem bereits früher veröffentlichten Artikel [Hu et al., 2005] der selben Autoren, welcher oben beschrieben wurde. In dieser Publikation gehen Hu et al. nun genauer darauf ein, wie sie auf Basis der Ergebnisse der Clusteranalyse, statistische Informationen über die Fahrbahnen der Fahrzeuge berechnen. Hierzu wird für jedes Cluster zuerst eine Referenz-Trajektorie T_r bestimmt. Dies ist jene Bewegungsbahn, bei welcher die Summe der Distanzen zu allen anderen Trajektorien des Clusters minimal ist. Das verwendete Distanzmaß ist hierbei das selbe, welches auch bei der Clusteranalyse zum Einsatz kam. Die Autoren berechnen anschließend für jede Trajektorie-Gruppe eine Kette Gausscher-Wahrscheinlichkeits-Verteilungen $\{\varphi_1, \varphi_2, \dots, \varphi_l\}$, wobei l die Anzahl der Punkte der Referenz-Trajektorie ist. Für jedes φ_i wird der Mittelwert und die Kovarianz, basierend auf den Trajektorie-Punkten, welche dem i -ten Punkt von T_r am nächsten liegen, berechnet. Anhand der Kovarianz-Werte erstellen die Autoren Hüllen für die Fahrbahnen. Ergebnisse dieses Vorgehens sind in Abbildung 4.6 dargestellt.

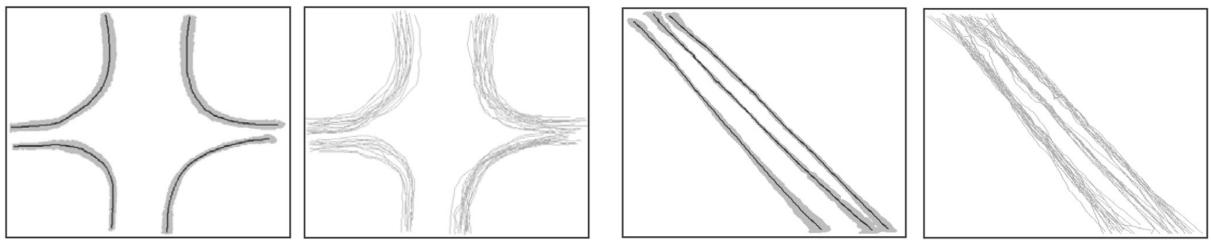


Abbildung 4.6: Spurhüllen in [Weiming Hu et al., 2006]

Anzumerken ist, dass die extrahierten Spurhüllen nicht die tatsächliche Form, insbesondere die Breite, einer Fahrspuren wiederspiegeln. Sie sind deutlich schmäler.

Learning Semantic Scene Models From Observing Activity in Visual Surveillance

In [Makris and Ellis, 2005] extrahieren die Autoren übliche Bewegungsbahnen von Fußgängern aus stationären Videoaufnahmen. Sie verwenden hierzu keine klassische Clusteranalyse, sondern ein iteratives und adaptives Online Verfahren, welches neue Bewegungsstrajektorien automatisch bestehenden Routen zuordnet oder neue initialisiert, falls zu hohe Abweichungen zwischen der Trajektorie und den existierenden Bahnen bestehen. Routen definieren Makris et al. hierbei als eine Sequenz von Knoten, welche folgende Merkmale besitzen:

- 2D-Mittelpunkt
- Gewichtung (Anzahl der Trajektorien im Bereich des Knoten)

- Zwei Hüll-Punkte (Maximale- beziehungsweise Standardabweichung der Trajektorien der Route im Bereich des Knoten)

Abbildung 4.6 a) veranschaulicht nochmals die Definition einer Route. Um aus einfachen Bewegungsbahnen Routen zu erstellen, verwenden die Autoren das nachfolgend beschriebene Verfahren, welches dem agglomerativen Cluster-Ansatz ähnelt.

1. Die erste Trajektorie initialisiert die erste Route.
2. Neue Trajektorien werden mit bestehenden Bahnen abgeglichen und
 - a) bei Übereinstimmung wird Route aktualisiert, oder
 - b) bei keiner Übereinstimmung wird neue Route erzeugt.
3. Aktualisierte Routen werden auf definierten Knotenabstand r resampled.
4. Alle Routen werden miteinander verglichen und
 - a) bei einer Überlagerung der Routen werden diese fusioniert.

Als Vergleichsmaß für die Trajektorien und Routen, verwenden Makris et al. die maximale Distanz zwischen einer Trajektorie und einer Routen-Hülle. Diese Distanz muss sich unterhalb eines bestimmten Grenzwertes befinden, damit eine Trajektorie einer Route zugeordnet wird. Geschieht dies, dann werden alle Hüll- und Mittelpunkte neue berechnet. Ein Ergebnis des Verfahrens ist in Abbildung 4.7 b) dargestellt.

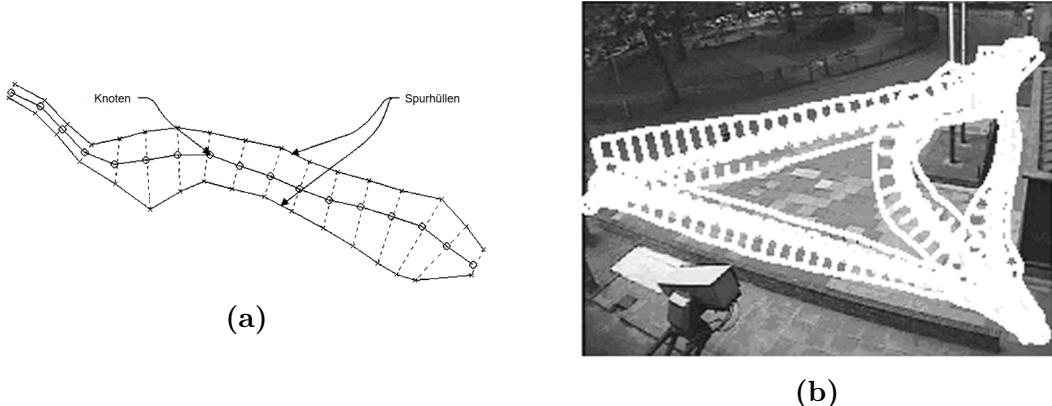


Abbildung 4.7: a) Routen Definition, b) Ergebnisse Routen-Erkennung [Makris and Ellis, 2005]

Trajectory Learning for Activity Understanding: Unsupervised, Multilevel, and Long-Term Adaptive Approach

In [Morris and Trivedi, 2011] stellen die Autoren ein dreistufiges Framework zur Auswertung von Fahrzeugtrajektorien vor. Ziel des Frameworks ist es, die Bewegungsmuster von Objekten mittels eines erlernten Vokabulars beschreiben zu können sowie Aktivitäten vorhersagen und Anomalien erkennen zu können. In einem ersten Schritt werden daher sogenannte *Points of Interests* (POI) identifiziert. Die von Morris et al. untersuchten POI's sind die Eintritts-, Austritts- und Stop-Zonen innerhalb einer Szene. Nach der deren Bestimmung identifiziert das vorgestellte Framework übliche Bewegungsmuster in den Trajektorien und definiert auf deren Basis anschließend Pfade.

Sich häufig wiederholende Bewegungsmuster werden von den Autoren über eine Clusteranalyse ermittelt. Zum Vergleich der Trajektorien kommt das in Abschnitt 3.3.3 vorgestellte LCSS Distanzmaß zum Einsatz und als Clusteralgorithmus das Spectral-Clustering. Morris et al. verwenden statt des im Spectral-Clustering üblicherweise eingesetzten k-Mean-Algorithmus, einen Fuzzy-C-Mean Ansatz, welcher für jede Trajektorie einen Zugehörigkeitswert $u_{ik} \in [0, 1]$ zum Cluster k bestimmt. Anhand der Zugehörigkeiten der Bewegungsbahnen zu den Clustern werden Referenz-Trajektorien bestimmt. Diese ergeben sich für jedes Cluster als gewichteter Durchschnitt aller Trajektorien. Als Gewichte werden die Werte u_{ik} verwendet. Die so erstellten Referenz-Linien sind in Abbildung 4.8 a) zu sehen.

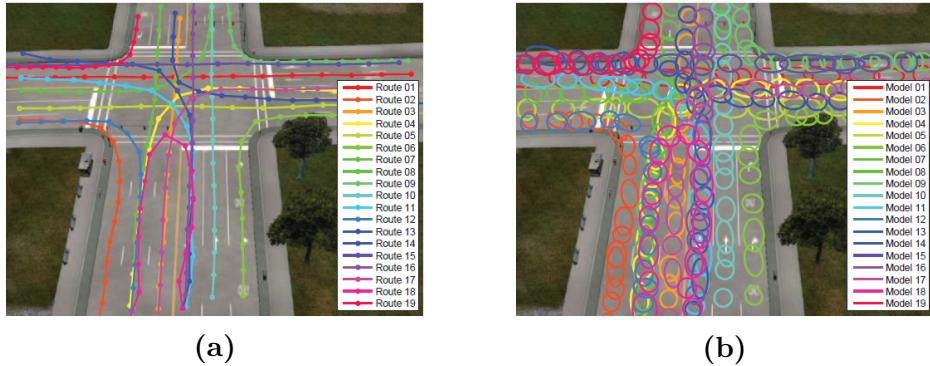


Abbildung 4.8: a) Referenz-Trajektorien, b) Pfade basierend auf HMM's [Morris and Trivedi, 2011]

Basierend auf den Clustern definieren Morris et al. außerdem Pfade, welche die räumliche und zeitliche Dynamik der Fahrzeuge an einer bestimmten Stelle abbilden. Sie verwenden hierzu Hidden-Markov-Modelle (HMM), welche sie mittels der Baum-Welch-Methode trainieren. Die Ergebnisse dieses Ansatzes sind in Abbildung 4.8 b) dargestellt. Die Ellipsen repräsentieren die HMM's, welche angeben, wie die erwartete Bewegung eines Fahrzeugs in einem bestimmten Bereich aussieht. Die in [Morris and Trivedi, 2011] bestimmten Pfade

ermöglichen es diverse Aussagen über das Verhalten der Fahrzeuge zu treffen. Sie repräsentieren allerdings nicht die realen Spur-Geometrien eines Straßenabschnittes.

Automatic Traffic Surveillance System for Vehicle Tracking and Classification

[Hsieh et al., 2006] stellen in ihrer Arbeit ein System zur Extraktion von Fahrzeugpositionen aus Aufnahmen stationärer Verkehrskameras vor. Auf diesen aufbauend definieren sie einen Algorithmus, mit dessen Hilfe es möglich ist, Spurmittel- und Begrenzungs-Linien zu entdecken. Hierzu erzeugen Hsieh et al. ein Histogramm $H_{vehicle}(x,y)$, welches die Häufigkeit abbildet, dass ein Fahrzeug sich über eine bestimmte Pixel-Position (x, y) bewegt. Ein solches ist in Abbildung 4.9 a) dargestellt. Die Autoren gehen davon aus, dass die Fahrzeugpositionen sich mehrheitlich in der Mitte einer Spur befinden und ein Histogramm, welches für eine Fahrbahn mit L_n Spuren erstellt wurde, L_n Maxima in jeder Zeile besitzt, welche die Spurmitten darstellen. Sie isolieren daher die Maxima in $H_{vehicle}$ als Mittellinien. Es sei anschließend $C_{L_k}^j$ die k -te Spurmittellinie in Zeile j des Histogramms. Daraus berechnen Hsieh et al. die Spurbegrenzungslinien. Alle innen liegenden Begrenzungen ergeben sich aus Gleichung 4.10. DL_k^j entspricht hierbei einem Punkt in der j -ten Reihe der k -ten Begrenzung.

$$DL_k^j = \frac{1}{2}(C_{L_{k-1}}^j + C_{L_k}^j) \quad (4.10)$$

Die Breite $w_{L_k}^j$ der k -ten Fahrspur ergibt sich zudem wie folgt:

$$w_{L_k}^j = |C_{L_k}^j - C_{L_{k-1}}^j| \quad (4.11)$$

Die Positionen der äußeren Spurbegrenzungen einer Fahrbahn ergeben sich für die j -te Zeile des Histogramms aus den Gleichungen 4.12 und 4.13.

$$(x_{DL_0^j}, j) = (x_{DL_1^j - w_{L_0}^j}, j) \quad (4.12)$$

$$(x_{DL_{N_L}^j}, j) = (x_{DL_{N_{L-1}}^j + w_{L_0}^j}, j) \quad (4.13)$$

Ein Beispiel für die Ergebnisse der Spurerkennung von Hsieh et al. ist in Abbildung 4.9 b) dargestellt. Das Verfahren funktioniert gut, wenn Fahrspuren nebeneinander und parallel zueinander liegen. Die Dimensionen sich kreuzender Fahrspuren können mit dem Verfahren beispielsweise allerdings nicht bestimmt werden.



Abbildung 4.9: a) Fahrzeugpositions Histogramm, b) Spurmittellinien und Spurbegrenzungslinien [Hsieh et al., 2006]

4.3 Defizite vorhandener Lösungen und benötigte Neuerungen

Es existiert eine große Anzahl von Arbeiten, welche sich mit der Clusteranalyse von Trajektoriedaten befasst. Die in Abschnitt 4.1 vorgestellten Veröffentlichungen stellen nur eine kleine Auswahl dar. Die Arbeiten entwickeln Lösungen in verschiedensten Anwendungsbereichen wie der Verkehrs-, Wetter und Verhaltensanalyse und suchen daher in den Trajektoriedaten nach unterschiedlichsten Mustern. Die geforderte Genauigkeit der Clustering-Ergebnisse unterscheidet sich je nach Anwendungsfall ebenfalls stark. Im Rahmen dieser Masterarbeit muss ein Verfahren identifiziert und entwickelt werden, welches Fahrspur-Cluster in Trajektoriedaten zuverlässig identifizieren kann. Die Fahrspuren können hierbei unterschiedlichste Geometrien aufweisen.

Ein Defizit vieler vorhandener Arbeiten ist, dass diese meist mit wenigen unterschiedlichen Trajektorie-Datensätzen arbeiten und die verwendeten Daten selten Defekte wie Unterbrechungen, Ausreißer oder inkorrekte Positionsinformationen aufweisen, welche durch Fehler in der Fahrzeugverfolgung entstehen. In einigen Arbeit wird teilweise nur mit generierten Trajektoriedaten gearbeitet. Da in der vorliegenden Thesis davon ausgegangen werden muss, dass all diese Probleme auftreten können, muss ein zuverlässiges Verfahren zur Bereinigung von Trajektoriedaten entwickelt werden. Ohne einen solchen Vorverarbeitungsschritt, wäre ein akkurate Clustering nicht möglich.

Deutlich weniger Veröffentlichungen gibt es zum Thema der Identifikation von Fahrspuren. Zwar existieren einige Arbeiten, welche Fahrspuren auf Basis von Trajektorien ermitteln, diese entsprechen aber in den allermeisten Fällen nicht den realen Verläufen der Fahrbahnen. Üblicherweise werden Fahrspuren anhand statistischer Verfahren, wie in [Weiming Hu et al., 2006] oder in [Teng et al., 2015] beschrieben, ermittelt. [Hsieh et al., 2006] ermitteln in ihrer Arbeit Spur-Begrenzungslinien, allerdings nur für kurze, parallele Fahrspuren eines Autobahnabschnittes. In dieser Arbeit sollen reale Spurgeometrien für unterschiedlichste Straßenabschnitte bestimmt werden, weshalb das hierzu eingesetzte Verfahren selbst entwickelt werden muss.

Es muss zudem eine Lösung zur Partitionierung von Fahrspuren entwickelt werden, da sich erkannte Spuren in dieser Arbeit nicht über einen längeren Bereich hinweg überlagern dürfen. Es muss zudem für jede automatisch erkannte Spur überprüft werden, ob es sich bei ihr tatsächlich um eine real Fahrspur handeln kann, oder ob ihr Verlauf im Verhältnis zu dem der anderen Bahnen nicht plausibel ist.

5 Konzeption des Spurerkennungs-Moduls

In diesem Kapitel wird das zu entwickelnde Teilmodul “*Spurerkennung*” der MEC-View *TrackerApplication* konzipiert. Hierzu wird zuerst dessen Rolle und Position im Gesamtkontext der Anwendung betrachtet. Anschließend werden die Anforderungen und der Entwurf des Moduls vorgestellt.

5.1 Überblick über das Gesamtsystem

Das Modul *Spurerkennung* dient der Erreichung der in Abschnitt 1.2 definierten Ziele. Erstellt wird es im Rahmen des MEC-View Teilprojektes *Luftbeobachtung* als Teilmodul der Anwendung *TrackerApplication*. Abbildung 5.1 gibt einen Überblick über das System. Es werden hierbei jene Module beziehungsweise Schritte vorgestellt, welche mit der Spurerkennung in Zusammenhang stehen.

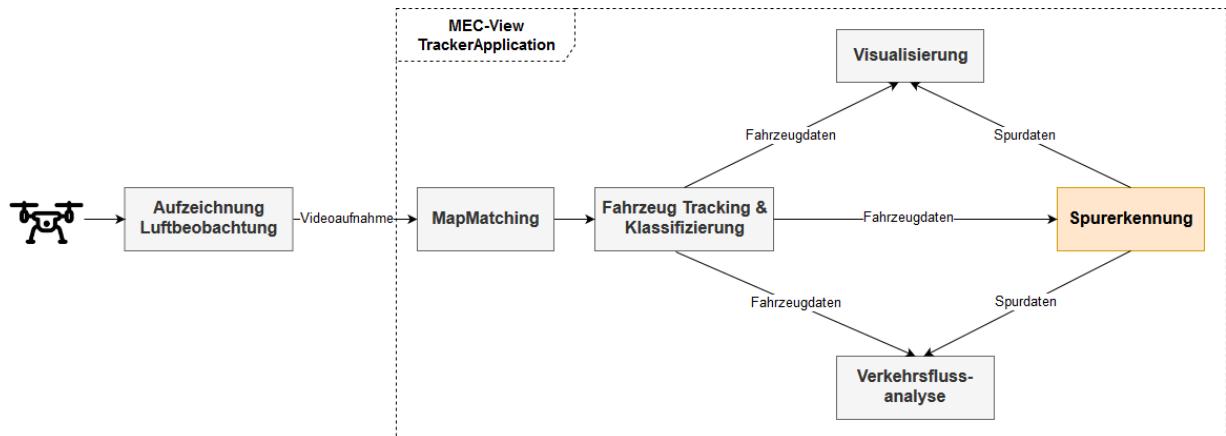


Abbildung 5.1: Kontext des Moduls Spurerkennung

Die mithilfe von Drohnen erstellten Videoaufnahmen können in der MEC-View *TrackerApplication* verarbeitet und analysiert werden. In einem ersten Schritt “*MapMatching*”, wird hierzu ein Welt-Koordinatensystem in Metern definiert. Anschließend können die Positionen der Fahrzeuge bestimmt werden. Diese ersten zwei Schritte sind in Abschnitt 2 genauer beschrieben. Die Fahrzeuginformationen, insbesondere die Positionsinformationen, dienen anschließend dem *Spurerkennung*-Modul als Eingabe. Aus ihnen extrahierte

Spurdaten können anschließend in der Anwendung visualisiert werden oder in Kombination mit den Fahrzeuginformationen zur Analyse des Verkehrsflusses eingesetzt werden.

5.2 Anforderungen an das Modul

In diesem Abschnitt werden die wichtigsten funktionalen und nicht funktionalen Anforderungen an das Modul Spurerkennung festgehalten.

5.2.1 Funktionale Anforderungen

Anforderung 1000 (Top-Level) Das *Spurerkennungs-Modul* soll es ermöglichen, mithilfe der *TrackerApplication* automatisch Fahrspuren aus den Positionsinformationen von Fahrzeugen in Luftaufnahmen abzuleiten.

Anforderung 2000 Das Modul soll die Erkennung von Fahrspuren in den in XXX vorgestellten Straßentopologien unterstützen.

Anforderung 2100 Das Modul soll unabhängig vom Aufnahmewinkel der Kamera Fahrspuren zuverlässig aus Videoaufnahmen ableiten können.

Anforderung 2200 Das Modul soll Fahrspuren bei Überlagerungen sinnvoll partitionieren können.

Anforderung 2300 Das Modul soll die Plausibilität von Fahrspuren bewerten können und nicht plausible Spuren entfernen.

Anforderung 2400 Das Modul soll die Enden benachbarter und paralleler Fahrspuren aneinander angleichen.

Anforderung 2500 Das Modul soll es ermöglichen, die aus den Trajektorien abgeleiteten Fahrspuren in der *TrackerApplication* zu visualisieren.

5.2.2 Nicht funktionale Anforderungen

Anforderung 3000 Das *Spurerkennungs-Modul* muss robust mit Ausreißern und Tracking-Fehlern in den Trajektorien umgehen können.

Anforderung 3100 Die Performance des Spurerkennung-Vorgangs ist nicht von höchster Priorität. Eine Erkennung sollte allerdings dennoch maximal wenige Minuten dauern.

5.3 Entwurf des Moduls

In diesem Abschnitt wird, basierend auf den Erkenntnissen der Literaturrecherche und den Anforderungen, ein grober Entwurf des *Spurerkennung*-Moduls vorgestellt.

Das Modul definiert primär einen Algorithmus, welcher aus Fahrzeuginformationen wie der Position oder Geschwindigkeit von Fahrzeugen, Fahrspuren ableitet. Die Grundfunktionsweise dieses Algorithmus ist in Abbildung 5.2 in Form eines Aktivitätsdiagramms dargestellt.

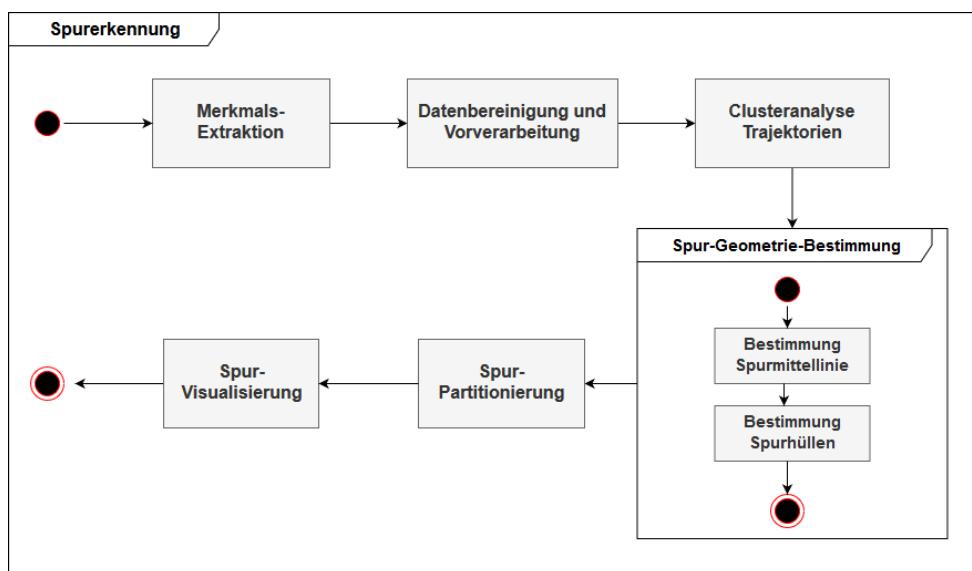


Abbildung 5.2: Grundstruktur des Spurerkennungs-Algorithmus

Die einzelnen Schritte des Algorithmus werden im *Spurerkennungs*-Modul als einzelne Komponenten implementiert. Die *TrackerApplication* ist in Java und Scala implementiert. Ihre Benutzeroberfläche basiert auf JavaFX. Das in dieser Arbeit erstellte Modul wird komplett mit Scala umgesetzt.

Die nachfolgenden Kapitel beschreiben, wie die einzelnen Schritte des Algorithmus aus Abbildung 5.2 realisiert werden und welche Probleme hierbei überwunden werden müssen.

6 Realisierung der Fahrspurerkennung

In diesem Kapitel wird die Realisierung der Spurerkennung thematisiert. Es wird zuerst darauf eingegangen, wie die Trajektoriedaten im Algorithmus repräsentiert und, vor der Clusteranalyse, vorverarbeitet werden. Anschließend werden zwei untersuchte und getestete Ansätze zur Clusteranalyse von Trajektorien vorgestellt. Den Schluss des Kapitels bildet die Erläuterung des Vorgehens zur Bestimmung der Spur-Geometrien.

6.1 Repräsentation und Vorverarbeitung der Trajektoriedaten

Nachfolgend wird beschrieben, welche Repräsentation für die Trajektoriendaten in dieser Arbeit gewählt wurde und welche verschiedenen Schritte die Trajektorien vor der Clusteranalyse durchlaufen.

6.1.1 Trajektorie-Repräsentation

Die Resultate der Fahrzeugverfolgung (siehe Abschnitt 2) werden in der *TrackerApplication* in Form sogenannter *TrackedObjects* gespeichert. Ein solches Objekt repräsentiert eine zusammenhängende, nicht unterbrochene Verfolgung eines Fahrzeugs. Wird eine Verfolgung, beispielsweise aufgrund einer Überdeckungen, unterbrochen, so existieren für ein Kraftfahrzeug mehrere Objekte, welche sich nicht einander zuordnen lassen. Die wichtigsten Informationen, die ein *TrackedObject* beinhaltet, sind eine eindeutige ID, die Frame-Positionen des Starts und Endes der Verfolgung und die Objekt-Klasse des Fahrzeugs. Es wird zwischen den vier Klassen “Auto”, “Lastwagen”, “Transporter” und “Zweirad” unterschieden. Für jedes verfolgte Objekt können die zugehörigen Positions-, Geschwindigkeits-, Beschleunigungs- und Größen-Informationen abgerufen werden. Diese werden für jedes Video-Frame, welches zwischen dem Start- und End-Frame des Objektes liegt, bestimmt.

Da für die Ableitung von Fahrspuren aus Trajektorien lediglich die positionsbezogenen Eigenschaften der Fahrzeuge relevant sind, werden Bewegungsbahnen in dieser Arbeit nur über jene definiert. Geschwindigkeit, Beschleunigung und Größe der Fahrzeuge wird in der Clusteranalyse nicht berücksichtigt.

Zur Erzeugung der Trajektorien werden die “Front-Positionen” der verfolgten Objekte verwendet. Diese markieren den vordersten Punkt eines Fahrzeugs in einem bestimmten Video-Frame und befinden sich immer auf dessen Bounding-Box. Idealerweise liegt die

Front-Position auf der Stoßstange eines Fahrzeugs. In Abbildung 6.1 sind beispielhaft zwei verfolgte Objekte dargestellt. Die Front-Positionen entsprechen den runden Markern.

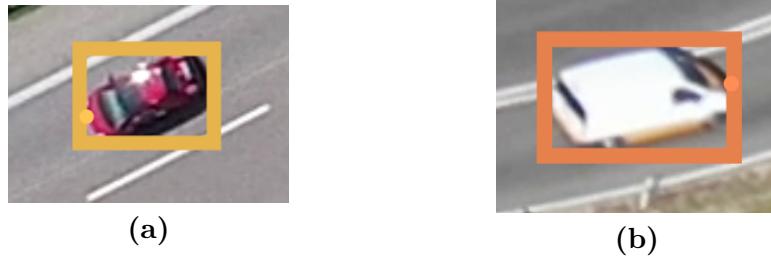


Abbildung 6.1: Erkannte Fahrzeuge und deren Front-Position

Der Vorteil, welcher sich bei der Verwendung der Front-Positionen ergibt, ist, dass diese auch bei niedrigen Aufnahmewinkeln sich nicht zu weit vom Mittelpunkt der Fahrspur, auf welchem sich das Fahrzeug befindet, entfernen. Dies ermöglicht eine bessere Bestimmung der Fahrspuren bei niedrigen Aufnahmewinkeln.

Abbildung 6.2 zeigt den Aufbau einer Trajektorie im Modul *Spurerkennung*.

Trajectory
+ id: Int + objectClass: String + positions: Vector[Point] + pointLength: Int + distanceToStart: Vector[Double] + realLength: Double + clusterLabel: Option[Int]

Abbildung 6.2: Aufbau Trajektorie-Klasse

Die Felder *id* und *objectClass* werden aus dem der Trajektorie zugrundeliegenden *Tracked-Object* übernommen. Die Positionen eines Fahrzeugs werden in Form von 2D-Welt-Koordinaten (siehe Abschnitt 2) in *positions* gespeichert und die Anzahl der Koordinaten zusätzlich in *pointLength*. Die Sequenz *distToStart* enthält für jeden Punkt der Bewegungsbahn dessen Distanz zum Start der Trajektorie in Metern. Die Werte ergeben sich aus Formel 6.1, wobei p_n dem n -ten Punkt in der Trajektorie entspricht und *dist* der euklidischen Distanz zwischen zwei Punkten.

$$distToStart(p_n) = \begin{cases} 0 & \text{if } n = 0 \\ dist(p_n, p_{n-1}) + distToStart(p_{n-1}) & \text{otherwise} \end{cases} \quad (6.1)$$

Aus $distToStart$ ergibt sich zudem die Gesamtlänge einer Trajektorie, welche extra gespeichert wird. Das Feld $clusterLabel$ ordnet jede Trajektorie nach der Clusteranalyse einem bestimmten Cluster zu. Zuvor enthält es keinen Wert.

Zur Untersuchung der Fahrzeugtrajektorien ist es hilfreich diese zu visualisieren. Abbildung 6.3 b) zeigt so beispielsweise 1240 Trajektorien, welche aus einer Aufnahme des Stuttgarter Neckartors extrahiert wurden. In Abbildung 6.3 a) ist ein Ausschnitt der entsprechenden Aufnahme zu sehen.

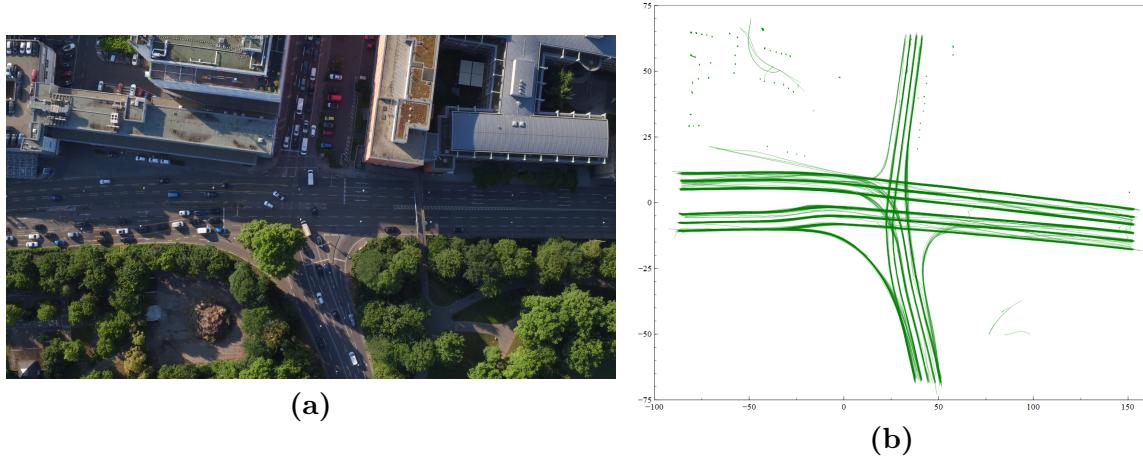


Abbildung 6.3: Stuttgarter Neckartor a) und Trajektorien b)

In Abbildung 6.3 b) sind die verschiedenen Bewegungsbahnen der Fahrzeuge für den menschlichen Betrachter bereits klar erkennbar. Direkt fallen aber auch die Trajektorien der stehenden oder sich auf Parkplätzen bewegenden Autos im oberen Bereich der Aufnahme ins Auge. Diese dürfen nicht in die Clusteranalyse mit einbezogen werden. Bei genauerer Untersuchung der Trajektorien zeigen sich weitere Probleme, welche das Clustering negativ beeinflussen würden. Zwei sind in nachfolgender Abbildung dargestellt. 6.4 a) zeigt, wie Fahrzeuge Punktewolken beim Stillstand vor Lichtsignalanlagen bilden. Abbildung 6.4 b) veranschaulicht, dass in manchen Bereichen sehr viele Trajektorie-Unterbrechungen auftreten. In diesen Bereichen werden die Fahrbahnen üblicherweise von Bäumen, Brücken et cetera überlagert, wodurch die Fahrzeugverfolgung unterbrochen wird.

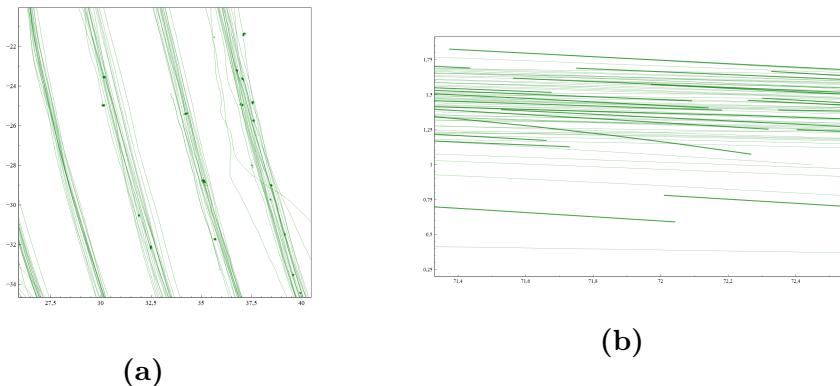


Abbildung 6.4: Punktwolken vor Lichtsignalanlagen a), Unterbrechungen aufgrund von Überdeckung b)

Um von diesen und weiteren Effekten bei der Clusteranalyse nicht beeinflusst zu werden, durchlaufen die “Roh-Trajektorien” einen Vorverarbeitungsschritt. Dieser wird im nächsten Abschnitt vorgestellt.

6.1.2 Vorverarbeitung der Trajektorien

Die verschiedenen Schritte, welche zur Vorverarbeitung und Bereinigung der Roh-Trajektorien angewandt werden, sind in diesem Abschnitt erläutert.

Das primäre Ziel der Vorverarbeitung ist es, Ausreißer aus der Trajektorie-Menge zu entfernen, welche das Ergebnis der Clusteranalyse negativ beeinflussen würden. Idealerweise sollen nur jene Trajektorien beibehalten werden, welche eine komplette Bewegung eines Fahrzeugs auf einem bestimmten Straßenabschnitt repräsentieren. Anhand dieser Trajektorien kann anschließend die Geometrie der realen Fahrspuren ermittelt werden. Ausreißer wie stehende oder unterbrochene Trajektorien liefern hingegen keine verwendbaren Informationen für die Spurerkennung.

In nachfolgender Auflistung sind die drei wichtigsten Vorverarbeitungsschritte und ihre Anwendungsreihenfolge aufgeführt:

1. Resampling von Trajektorien auf minimale Punktdistanz,
2. Entfernung zu kurzer Trajektorien und
3. Entfernung unterbrochener Trajektorien

Die einzelnen Schritte und ihr Hintergrund werden anschließend noch genauer erläutert.

Resampling von Trajektorien auf minimale Punktdistanz

Der erste Vorverarbeitungsschritt reduziert die Anzahl der Koordinaten, welche eine Bewegungsbahn beschreiben, erheblich, ohne dabei jedoch wichtige Informationen zu verlieren. Insbesondere dann, wenn sich Fahrzeuge mit niedrigen Geschwindigkeiten bewegen oder teilweise vor Ampeln et cetera stehen, bestehen die Roh-Trajektorien aus sehr vielen Punkten, welche beinahe identische Positionsinformationen darstellen, das heist nur sehr geringe Abstände voneinander haben. Für die Beschreibung einer Bewegungsbahn ist diese Punktdichte nicht notwendig und sogar kontraproduktiv, da sie die Performance der nachfolgenden Schritte stark erhöht. Hiervon ist insbesonders die Clusteranalyse betroffen.

Aus diesen Gründen werden im ersten Vorverarbeitungsschritt Trajektorien resampled, so dass die Distanz aufeinanderfolgender Punkte mindestens 1.5 m beträgt. Der hierzu verwendete Algorithmus ist in Listing 6.1 dargestellt. Er verwirft alle aufeinanderfolgende Punkte, welche von einem Referenzpunkt weniger als den geforderten Abstand haben.

```

1 algorithm resampleTrajectory:
2   input:  lastRefPoint, newTrajPoints, oldTrajPoints
3   output: resampled trajectory points
4
5   while oldTrajPoints is not empty do:
6     nextPoint := Head(oldTrajPoints)
7     remPoints := Tail(oldTrajPoints)
8
9     if dist(lastRefPoint, nextPoint) < 1.5:
10       resampleTrajectory(lastRefPoint, newTrajPoints, remPoints)
11     else:
12       resampleTrajectory(nextPoint, newTrajPoint ++ nextPoint, remPoints)
13     end
14   end
15
16  return newTrajPoints

```

Listing 6.1: Pseudocode Trajektorie Resampling

Nach Anwendung des Algorithmus ist im Fall der Neckartor Aufnahme die durchschnittliche Punktlänge der Trajektorien von 1094 Koordinaten auf 65 gesunken. Die realen Längen der Bewegungsbahnen bleiben hingegen nahezu identisch. Die Punktwolken, welche stehende Fahrzeuge erzeugen, werden mittels dieses Schritts ebenfalls entfernt (siehe Abb. 6.5 b)).

Entfernung von zu kurzen Trajektorien

Der zweite Verarbeitungsschritt kann viele Trajektorien von beispielsweise stehenden Fahrzeugen oder kurz auftretende Tracking-Fehler entfernen. Hierzu wird die Länge aller Trajektorien überprüft und jene entfernt, welche unter einem bestimmten Grenzwert liegen. Die hierzu verwendete boolesche Überprüfung ist in Gleichung 6.2 gegeben. Als Standardeinstellung für \minLength wurde experimentell der Wert 20m bestimmt, da er für alle

Testaufnahmen gute Ergebnisse liefert. Bei Bedarf kann der Anwender den Parameter beim Start des Spurerkennungs-Jobs in der Oberfläche anpassen.

$$isShortTrajectory(t) = \begin{cases} 1 & \text{if } t.realLength < minLength \\ 0 & \text{otherwise} \end{cases} \quad (6.2)$$

Das Ergebnis der ersten beiden Vorverarbeitungsschritte ist in Abbildung 6.5 dargestellt. Die Trajektorien stehender Fahrzeuge, sowie die Punktfolgen vor Lichtsignalanlagen und weitere Defekte aufgrund kleiner Tracking-Fehler wurden entfernt.

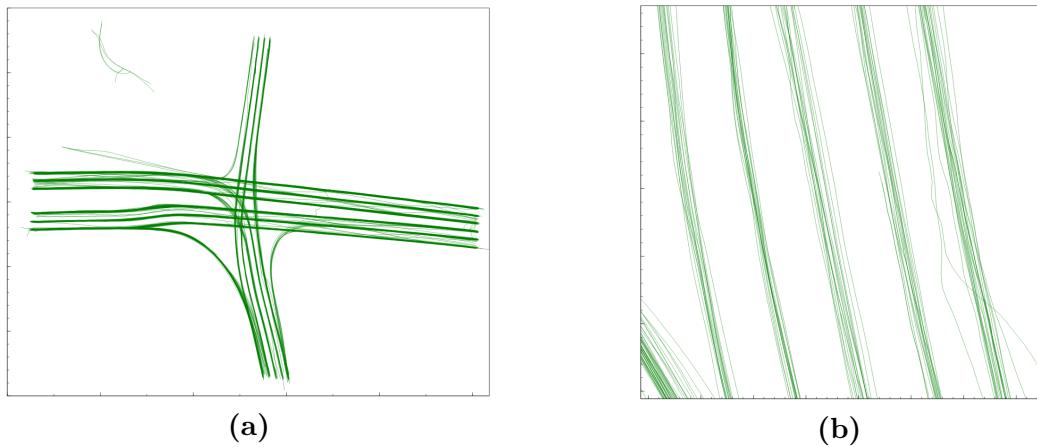


Abbildung 6.5: Ergebnisse der zwei ersten Vorverarbeitungsschritte

Entfernung unterbrochener Trajektorien

Nachdem durch die ersten beiden Vorverarbeitungsschritte bereits stehende Trajektorien entfernt wurden, müssen nun noch unterbrochene Verfolgungen aus gefiltert werden, da diese das Ergebnis der Clusteranalyse negativ beeinflussen.

Der erste verwendete Ansatz zur Entfernung von unterbrochenen Trajektorien beruhte auf der Annahme, dass komplettete Trajektorien immer im Bereich der Szenenänder beginnen und enden. Daher wurden alle Trajektorien, welche diese Bedingung nicht erfüllen, entfernt. Es ist jedoch möglich, dass Trajektorien, welche keine Defekte besitzen, auch in der Mitte einer Aufnahme beginnen, wenn die verfolgten Fahrzeuge in diesem Bereich beispielsweise aus einem Tunnel hervorkommen. Unter Anwendung des obigen Ansatzes würden diese Trajektorien fälschlichweise entfernt. Daher wurde ein anderes Vorgehen entwickelt.

Das in der Arbeit final eingesetzte Verfahren basiert auf einer anderen Definition von unterbrochenen Trajektorien: Bewegungsbahnen gelten grundlegend dann als unterbrochen, wenn sich auf Höhe ihrer Starts oder Enden mehrere Trajektorien befinden, welche auf

der entsprechenden Höhe nicht starten oder enden. Ist dies der Fall, dann ist die untersuchte Bewegungsbahn unterbrochen. Die angrenzenden Trajektorien sind potenziell vollständig. Abbildung 6.6 veranschaulicht diese Definition.

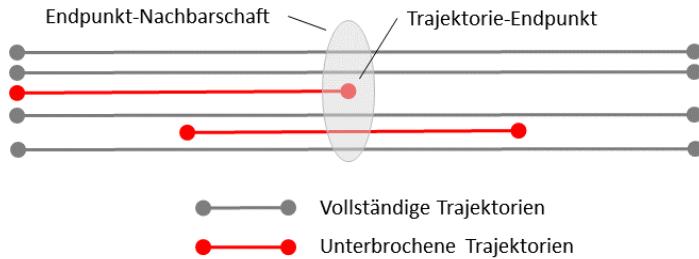


Abbildung 6.6: Identifikation von unterbrochenen Trajektorien

Der Algorithmus zur Entfernung unterbrochener Trajektorien nutzt die oben gegebene Definition. Er untersucht von allen Trajektorien die Nachbarschaften der Start- und Endpunkte. Befinden sich in den Nachbarschaften mindestens 25% der Punkte nicht am Rand der entsprechenden Trajektorien, so wird die untersuchte Bewegungsbahn als unterbrochen gewertet und entfernt. Dieses Vorgehen beruht auf der Annahme, dass für eine Fahrspur neben unterbrochenen auch ununterbrochene Trajektorien vorliegen, welche eine vollständige Bewegung auf der Spur beschreiben.

Nach Ausführung dieses Filter-Schrittes, besteht die Menge der übrigen Trajektorien grundsätzlich nur noch aus kompletten Bewegungsbahnen.

6.2 Clusteranalyse der Trajektorien

Nachdem im vorherigen Abschnitt beschrieben wurde, wie die Roh-Trajektorien vorverarbeitet und gefiltert werden, folgt nun die Erläuterung der verwendeten Ansätze zur Clusteranalyse der bereinigten Trajektorien.

6.2.1 Ansatz Spectral-Clustering und modifizierte Hausdorff-Distanz

Als erster Ansatz für die Clusteranalyse wurde das von [Atev et al., 2010] vorgestellte Verfahren untersucht. Es wurde gewählt, da es sowohl in der Arbeit von Atev et al. selbst, sowie auch in [Morris and Trivedi, 2009], gute Ergebnisse bei der Clusteranalyse lieferte. Das Verfahren beruht auf der in [Atev et al., 2006] vorgestellten modifizierten Hausdorff-Distanz und dem Spectral-Clustering-Algorithmus. Die Grundlagen des Distanzmaßes wurden bereits in Abschnitt 4.1 beschrieben. Nachfolgend wird zuerst genauer beschrieben, wie die modifizierte Hausdorff-Distanz berechnet wird. Anschließend wird auf den eingesetzten Spectral-Cluster-Algorithmus und die erzielten Ergebnisse eingegangen.

Das Verfahren

Grundlegend war die modifizierte Hausdorff-Distanz nach Atev et al., wie in Gleichung 4.1 bereits definiert, gegeben über:

$$h_{\alpha,N,C}(P, Q) = \underset{p \in P}{ord}^{\alpha} \left\{ \min_{q \in N_Q(C_{P,Q}(p))} d(p, q) \right\}$$

Um die Distanz zwischen zwei Trajektorien P und Q zu bestimmen, müssen zuerst die minimalen Distanzen zwischen allen Punkten $p \in P$ und deren Nachbarschaften $N_Q(C_{P,Q}(p))$ in Q bestimmt werden. Eine Nachbarschaft in Q um den Punkt q_0 ist in Abhängigkeit des Parameters w definiert als:

$$N_Q(q_0) = \{q \in Q \mid |\pi_Q(q_0) - \pi_Q(q)| \leq w/2\} \quad (6.3)$$

$\pi_Q(q)$ entspricht hierbei der relativen Position von q in Q , welche in Gleichung 6.4 definiert ist. $|Q_j|$ steht für die Länge einer Trajektorie bis zum Punkt j und $|Q|$ für die Gesamtlänge einer Bewegungsbahn. Diese Längeninformationen sind beide in der verwendeten Trajektorie-Definition aus Abbildung 6.2 enthalten.

$$\pi_Q(q_j) = \frac{|Q_j|}{|Q|} \quad (6.4)$$

Die Nachbarschaften werden um den Referenzpunkt $q \in Q$ gebildet, welcher die selbe relative Position in Q besitzt wie p in P . Der Index dieses Punktes ergibt sich aus dem Mapping $C_{P,Q}$ wie folgt:

$$C_{P,Q}(p) = \arg \min_{q \in Q} |\pi_P(p) - \pi_Q(q)| \quad (6.5)$$

Zur Berechnung der Distanzen $d(p,q)$ zwischen p und allen Punkten $q \in N_Q(C_{P,Q}(p))$, wird die euklidische Distanz verwendet. Wurden auf diese Weise alle minimalen Distanzen zwischen Punkten und ihren Nachbarschaften in der Vergleichs-Trajektorie bestimmt, so wird aus ihnen der finale Distanzwert bestimmt. Der Operator $ord_{p \in P}^{\alpha}$ wählt hierfür jene Distanz, welche größer ist als α -Prozent aller Werte. Mit Hilfe dieses Distanzmaßes, wird eine Distanz-Matrix D konstruiert, welche die Distanzen aller Trajektorie-Kombinationen speichert.

Da im Spectral-Clustering Verfahren die modifizierte Hausdorff-Distanz nicht direkt eingesetzt werden kann, muss ein zusätzliches Affinitätsmaß verwendet werden. Dieses definieren Atev et al. als:

$$k(P,Q) = \exp\left(-\frac{h_{\alpha,N,C}(P,Q) h_{\alpha,N,C}(Q,P)}{2\sigma(P)\sigma(Q)}\right) \quad (6.6)$$

$\sigma(P)$ und $\sigma(Q)$ entsprechen hier Schätzungen für die Streuung der Distanzwerte einer Trajektorie zu allen anderen Trajektorien. Sie ergeben sich aus der Distanz-Matrix D .

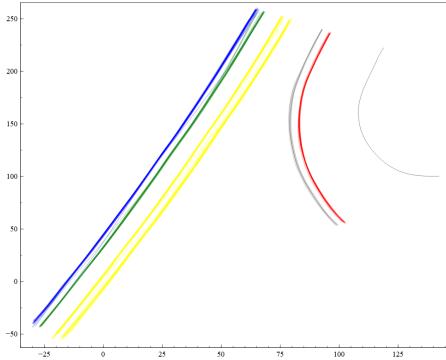
Der in dieser Arbeit und von Atev et al. verwendete Spectral-Cluster-Algorithmus folgt grundsätzlich der Standard-Definition von [Ng et al., 2002]. Für eine feste Clusteranzahl k kann er wie folgt zusammengefasst werden:

- 1) Erstellen einer Affinitätsmatrix $A \in \mathbb{R}^{n \times n}$ basierend auf Affinitätsmaß, wobei $A_{ii} = 0$
- 2) Definieren einer Diagonalmatrix D , deren Elemente an der Stelle (i,i) der Summe der i -ten Zeile von A entsprechen. Basierend auf D wird die Matrix $L = D^{-1/2}AD^{-1/2}$ erstellt.
- 3) Durchführen einer Eigenwert Dekomposition auf L , um die k größten Eigenvektoren $\{x_1, x_2, \dots, x_k\}$ zu finden.
- 4) Erstellen einer Matrix $X = [x_1, x_2, \dots, x_k]$ durch spaltenweises Zusammenführen der Eigenvektoren und Normalisierung der Zeilen auf die Länge 1.
- 5) Gruppierung der Zeilen von Matrix X in k -Cluster unter Zuhilfenahme von k-Means et cetera. Jede Zeile wird als Datenpunkt interpretiert.

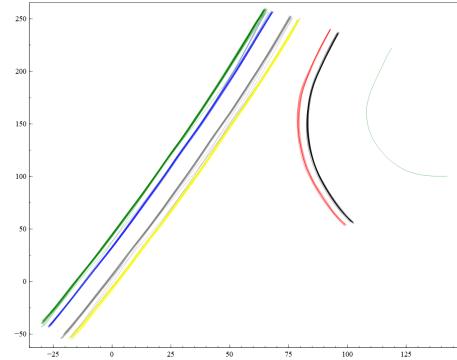
Ein großer Nachteil des Spectral-Clustering-Ansatzes ist es, dass bei seiner Verwendung üblicherweise die Clusteranzahl k bereits im vorraus bekannt und angegeben sein muss. Aus diesem Grund verwenden Atev et al. ein Schätzmaß für k , welches darauf abzielt, die Verzerrung des in Schritt 5) eingesetzten k-Means Algorithmus zu minimieren. Hierzu wird die k-Means Clusteranalyse mit mehreren k 's zwischen den Grenzen $kMin$ und $kMax$ durchgeführt und anschließend jeweils ein Verzerrungs-Maß ρ_k berechnet, welches angibt, wie gut die gefundenen Centroids die Datenpunkte beschreiben. Für k wird daher jener Wert gewählt, welcher das kleinste ρ_k erzeugt. Diese Methode zur Schätzung der Clusteranzahl wurde auch in der vorliegenden Arbeit angewandt.

Auswertung des Verfahrens

Für die Evaluierung des eingesetzten Verfahrens wurde die Clustering-Performance des Ansatzes initial anhand von zwei Trajektorie-Datensätzen untersucht. Die Ergebnisse der Clusteranalyse für den einfacheren Datensatz sind in Abbildung 6.7 dargestellt. An diesem Beispiel lassen sich die Probleme bereits identifizieren. Für die Analyse wurden die Parameter $\alpha = 0.85$ und $w = 1.0$ verwendet. Zur Bestimmung von σ kamen die Grenzwerte $stdMin = 0.5$ und $stdMax = 10.0$ zum Einsatz. Diese haben sich in unterschiedlichen Versuchen als am besten geeignet erwiesen.



(a) mit Schätzung Clusteranzahl



(b) mit fester Clusteranzahl

Abbildung 6.7: Ergebnisse Clusteranalyse B10-Entennest (Ansatz Atev et al.)

Ein erstes Problem des Verfahrens sind die vielen Parameter und Grenzwerte, welche vom Anwender bestimmt werden müssen und nicht alle intuitiv verständlich sind. Neben den oben aufgeführten Parametern, gibt es noch weitere, welche bei der Bestimmung der Clusteranzahl zum Einsatz kommen. Eine Optimierung der Parameter und somit der erzielten Ergebnisse wäre sicherlich möglich gewesen, hierauf wurde aber aufgrund anderer existierender Schwierigkeiten verzichtet.

Problematisch ist das Vorgehen auch, da der Spectral-Cluster-Algorithmus nicht mit Ausreißern umgehen kann, welche trotz der Vorverarbeitung der Trajektorien immer noch in den Datensätzen vorhanden seien können. In Abbildung 6.7 a) und b) ist beispielsweise die einzelne Fahrspur im oberen, rechten Bereich einem Spur-Cluster zugeordnet, was nicht korrekt ist. Diese falsche Zuordnung der Ausreißer, würde die Bestimmung der Spur-Geometrien im nächsten Schritt des Algorithmus aus 5.2 erheblich erschweren.

Die schlechten Clustering-Ergebnisse des Ansatzes sind zudem Folge der nicht zuverlässig funktionierenden Schätzung der Clusteranzahl k . Im Fall des Datensatzes aus Abbildung 6.7, wird $k = 5$ statt korrekterweise $k = 6$ geschätzt. Hieraus ergibt sich, dass in a) zwischen zwei Fahrspuren nicht richtig unterschieden wird. In b) wurde die Clusteranzahl händisch spezifiziert, was in einer korrekten Gruppierung der Fahrspuren resultierte.

Ausschlaggenend dafür, dass der Ansatz von Atev et al. nicht weiter verfolgt und optimiert wurde, war schlussendlich jedoch die Performance der Clusteranalyse. Für den Trajektorie-Datensatz aus Abbildung 6.7, welcher nach der Vorverarbeitung nur 132 Bewegungsbahnen beinhaltet, benötigt die Clusteranalyse bereits über 90 Sekunden. Für einen Datensatz mit über 400 Trajektorien, beträgt die Verarbeitungsdauer über 8 Minuten. Die schlechte Performance lässt sich primär auf das aufwendige Distanzmaß, aber auch auf die mehrfache Durchführung des k-Means Algorithmus zurückführen.

Aufgrund der oben angeführten Problematiken wurde entschieden, dass der Ansatz von Atev et al. nicht weiter verfolgt werden soll. Es wurde ein Ansatz gesucht, welcher mit den

beschriebenen Herausforderungen besser umgehen kann.

6.2.2 Ansatz DBSCAN und LCSS

Nachdem das in Abschnitt 6.2.1 beschriebene Verfahren zur Clusteranalyse in den Untersuchungen schlechte Ergebnisse lieferte, wurde nach neuen Ansätzen gesucht. Kriterien für diese waren primär, dass sie mit Ausreißern sinnvoll umgehen können, weniger Parametrisierung benötigen beziehungsweise diese intuitiver ist, und sie eine bessere Performance besitzen. Aufgrund dieser Anforderungen wurde untersucht, inwiefern sich der DBSCAN-Cluster-Algorithmus in Kombination mit dem LCSS-Distanzmaß für die Clusteranalyse von Trajektorie-Daten eignet.

Vorteil des dichte-basierten DBSCAN-Algorithmus, dessen Funktionsweise bereits in Abschnitt 3.2.4 erläutert wurde, ist, dass er Ausreißer erkennt und die Anzahl der Zielcluster selbst bestimmen kann. Das LCSS Distanzmaß, grundlegend in Abschnitt 3.3.3 beschrieben, ähnelt der modifizierten Hausdorff-Distanz, lässt sich allerdings performanter implementieren und besitzt eine eingängigere Parametrisierung. Es lieferte in den Arbeiten [Morris and Trivedi, 2011] und [Chen et al., 2014] gute Clustering-Ergebnisse.

Das Verfahren

Die Grundgleichung des LCSS Distanzmaßes wurde, basierend auf [Vlachos et al., 2002], bereits in Gleichung 3.11 definiert und ist nachfolgend nochmals dargestellt.

$$LCSS_{\epsilon,\delta}(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 \text{ or } t_2 \in \emptyset \\ 1 + LCSS_{\epsilon,\delta}(t'_1, t'_2) & \text{if } dist(t_1(n), t_2(m)) < \epsilon \\ & \wedge |n - m| \leq \delta \\ max(LCSS_{\epsilon,\delta}(t'_1, t_2), LCSS_{\epsilon,\delta}(t_1, t'_2)) & \text{otherwise} \end{cases}$$

Mit ihrer Hilfe lässt sich bestimmen, wie groß die längste, übereinstimmende Subsequenz zweier Trajektorien t_1 und t_2 ist. Damit Punkte zweier Trajektorien als übereinstimmend gewertet werden, dürfen sie höchstens die Distanz ϵ zueinander haben und ihre Position in den Bewegungsbahnen sich um δ unterscheiden. Wenn das Zahlmaß, wie oben dargestellt, rekursiv implementiert wird, liegt die Zeitkomplexität des Algorithmus bei $O(2^n)$, was für den Vergleich einer größeren Anzahl von Trajektorien nicht geeignet ist. Die Performance kann auf $O(m n)$ verbessert werden, indem das Maß mit Hilfe von dynamischer Programmierung und *Memoization* umgesetzt wird. Der sich so ergebende Algorithmus ist in Listing 6.2 aufgeführt.

```

1  input:  trajectory: t1, trajectory: t2, epsilon, deltaFac
2  output: LCSS distance between t1 and t2
3
4  t1Len := point-length t1
5  t2Len := point-length t2
6  delta := deltaFac * min(t1Len, t2Len)
7  LCS := 2D array with dims. (t1Len+1, t2Len+1)
8
9  for i <- 1 to t1Len do:
10    for j <- 1 to t2Len do:
11      if dist(t1(i-1), t2(j-1)) < epsilon && |i-j| < delta then:
12        LCS(i)(j) = LCS(i-1)(j-1) + 1
13      else
14        LCS(i)(j) = max(LCS(i-1)(j), LCS(i)(j-1))
15      end
16    end
17  end
18
19  return LCS(t1Len)(t2Len)

```

Listing 6.2: Pseudocode LCSS Bestimmung

Der Parameter δ wird in dieser Implementierung nicht, wie meist üblich, fest definiert, sondern er ergibt sich als ein Teil der Länge der kurzeren Trajektorie (siehe Lst. 6.2 Zeile 7). Als Distanzfunktion für das LCSS Zählmaß, wird in dieser Arbeit das von [Vlachos et al., 2002] definierte Maß aus Gleichung 3.12 verwendet:

$$D_{LCSS}(\delta, \epsilon, t_1, t_2) = 1 - \frac{LCSS_{\delta, \epsilon}(t_1, t_2)}{\min(\text{len}(t_1), \text{len}(t_2))}$$

Auf die in Abschnitt 4.1 vorgestellte Erweiterung des Distanzmaßes wird verzichtet, da es nicht gewünscht ist, formähnliche aber im Raum verschobene Trajektorien einem Cluster zuzuordnen.

Der DBSCAN Cluster-Algorithmus wurde in diesem Fall nicht selbst implementiert. Es wurde die Implementierung der *Statistical Machine Intelligence and Learning Engine*¹ (*Smile*-) Bibliothek verwendet. Der dort angebotene DBSCAN Algorithmus kann beliebige Datenobjekte, unter Verwendung eines vom Anwender definierten Distanzmaßes, gruppieren. Zusätzlich muss lediglich die Größe der gewünschten ϵ -Nachbarschaft und *MinPts* definiert werden (siehe Abschnitt 3.2.4).

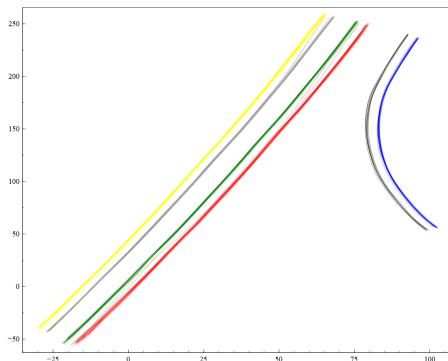
Auswertung des Verfahrens

In Abbildung 6.8 sind die Cluster-Ergebnisse des beschriebenen Verfahrens dargestellt. Diese Resultate ergeben sich unter Verwendung der folgenden Parameter: $\epsilon_{LCSS} = 1.5$, $\delta = 0.1$, $\epsilon_{DBSCAN} = 0.3$ und $\text{min}Pts = 5$.

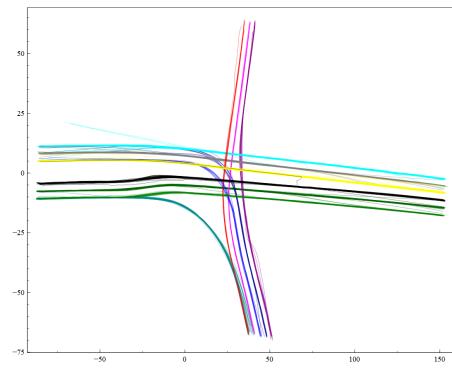
¹ Smile Bibliothek: <https://haifengl.github.io/smile/index.html>

Die Bedeutung der Werte ist in diesem Fall leicht und intuitiv verständlich: Punkte zweier Trajektorien dürfen maximal $1.5m$ voneinander entfernt sein und ihre Position sich um maximal $\delta * \text{minTraLength}$ unterscheiden. Sind diese Bedingungen erfüllt, gelten zwei Trajektorie-Punkte als übereinstimmend. Ein Cluster muss zudem aus mindestens 5 Trajektorien bestehen und die Differenzen der Trajektorie-Distanzen im Cluster untereinander dürfen nicht größer als 0.3 sein. Für ϵ_{DBSCAN} gilt grundsätzlich: Je stärker sich Fahrspuren überlagern, umso kleiner muss der gewählte Wert sein, da ein kleiner Wert für eine striktere Gruppierung der Trajektorien sorgt.

Die Ergebnisse aus Abbildung 6.8 a) und b) zeigen, dass bei Verwendung dieses Verfahrens zwischen allen Fahrspuren korrekt unterschieden wird und die einzelnen Trajektorien richtig gruppiert werden. Zudem funktioniert die Bestimmung der Clusteranzahl zuverlässig. Auch die Performance der Clusteranalyse konnte unter Verwendung des DBSCAN Algorithmus und der LCSS-Distanz verbessert werden. Die benötigte Zeit für den Autobahn-Datensatz sank auf circa 6 Sekunden und die des Kreuzung-Datensatz auf etwa 30 Sekunden. Für den vorliegenden Anwendungsfall eignet sich der Ansatz daher grundlegend gut.



(a) Datensatz B10-Entennest



(b) Datensatz Neckartor

Abbildung 6.8: Ergebnisse Clusteranalyse DBSCAN und LCSS-Distanz

Der DBSCAN Algorithmus markiert nun auch atypische Trajektorien als Ausreißer, was den Vorteil hat, dass diese nichtmehr anderen Spurclustern zugeordnet werden. Problematisch ist nun allerdings, dass Trajektorien welche Abbiegevorgänge beschreiben und in unterschiedliche Spuren einbiegen, teilweise auch als Ausreißer klassifiziert werden. In Abbildung 6.8 b) fehlen so beispielsweise die Trajektorien der zwei Abbiegespuren in der oberen linken und unteren rechten Ecke (vergleiche Abbildung 6.3 a)). Um diese weiterhin erkennen zu können, wurde ein extra Verarbeitungsschritt eingeführt, welcher in Abschnitt 6.2.3 beschrieben wird.

Grundsätzlich überzeugten allerdings die Ergebnisse der Clusteranalyse. Die Qualität des Ansatzes könnte auch durch Anwendung auf weitere Datensätze verifiziert werden. Die

nachfolgend beschriebenen Schritte der Spurerkennung basieren daher auf den Ergebnissen der hier vorgestellten Cluster-Methode.

6.2.3 Erkennung von Abbiegespuren

In Abschnitt 6.2.2 wurde bereits erwähnt, dass, unter Verwendung der gewählten Cluster-Methodik, einige Bewegungsbahnen als Ausreißer klassifiziert werden, dies eigentlich jedoch nicht sind. Diese fälschlicherweise aussortierten Trajektorien beschreiben für gewöhnlich Abbiegevorgänge. Problematisch an diesen Trajektorien, aus Sicht der Clusteranalyse, ist, dass sie üblicherweise auf einer gemeinsamen Spur – der Abbiegespur – beginnen, sich dann jedoch aufteilen und auf mehrere Fahrspuren verteilen. Die Trajektorien haben aus diesem Grund, trotz anfänglich identischem Verlauf, zu hohe Distanzen zueinander, um als ein Cluster identifiziert zu werden. Hinzu kommt, dass die Anzahl der Fahrzeuge, welche eine Abbiegespur benutzen, häufig nicht hoch genug ist, damit eine der Abbiege-Varianten ein Cluster formt. In Abbildung 6.9 a) und b) sind beispielhaft eine Abbiegespur der Neckartor-Kreuzung und die dazugehörigen Trajektorien abgebildet. Anhand der Straßen-topologie und der Trajektorien wird deutlich, dass sich die Fahrzeuge nach dem abbiegen auf drei Fahrstreifen verteilen.

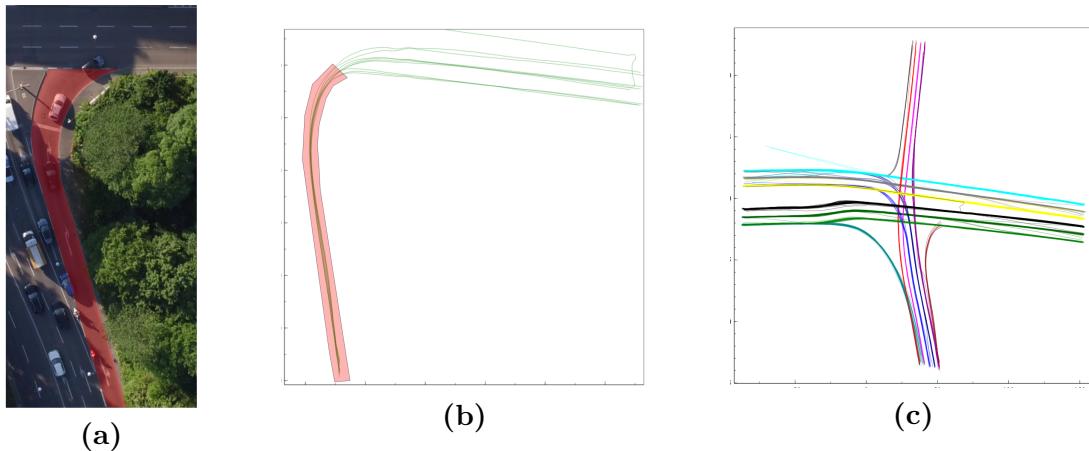


Abbildung 6.9: Abbiegespur a), zugehörige Trajektorien b), Spurcluster mit Abbiegespuren c)

Ziel des nachfolgend beschriebenen Verarbeitungsschrittes ist es, Trajektorien, welche Abbiegevorgänge beschreiben, zu entsprechenden Clustern zusammenzufassen. Anhand dieser kann anschließend die Geometrie der Spuren, wie für alle anderen Fahrspuren auch, bestimmt werden. Die Cluster sollen hierbei allerdings nicht aus den vollständigen Trajektorien gebildet werden, sondern lediglich aus den Teilen, welche anfänglich identisch verlaufen. Dies ist konzeptionell in Abbildung 6.9 b) dargestellt. Notwendig ist dies, da die Trajektorien nach dem Auseinanderlaufen sehr unterschiedliche Bahnen beschreiben können und dies die Spur-Geometrie-Bestimmung negativ beeinflusst.

Um die oben beschriebenen Spurcluster zu finden, werden ausschließlich die Trajektorien untersucht, welche während der Clusteranalyse als Ausreißer markiert wurden. In einem ersten Schritt wird nach möglichen Trajektorie-Nachbarschaften gesucht. Hierzu wird für jede Trajektorie t der Abstand zwischen deren Start und allen anderen Trajektorie-Starts bestimmt. Liegen mehr als fünf Trajektorie-Anfänge in einem Radius $laneEps = 1.5m$ um den Start von t , so bilden die Trajektorien eine mögliche Nachbarschaft. Dass die so gefundenen Trajektorien auch tatsächlich den Verlauf einer Abbiegespur beschreiben und nicht direkt auseinander laufen, wird anschließend geprüft.

Für eine Trajektorie-Nachbarschaft N wird zunächst eine Referenz Bewegungsbahn $refT$ bestimmt, welche den minimalen mittleren Abstand zu allen anderen Trajektorien besitzt:

$$refT_N = N(\arg \min_{t \in N} \left(\frac{1}{|N|} \sum_{tt' \in \{N \setminus t\}} D_{LCSS}(t, tt') \right)) \quad (6.7)$$

Die Referenz-Trajektorie beschreibt die Nachbarschaft. Im Fall der Trajektorien aus Abbildung 6.9 b) ist es beispielsweise eine Bahn, welche auf die mittlere Fahrspur abbiegt. Nachdem $refT$ bestimmt wurde, wird die Trajektorie Punktweise durchwandert. Es werden die Bereiche um jeden Punkt überprüft, um so festzustellen, wo die Trajektorien der Nachbarschaft auseinanderlaufen. Der hierzu eingesetzte Algorithmus ist in Listing 6.3 beschrieben.

```

1 algorithm findSplitPoint:
2   input: neighborhood: N, reference-trajectory: refT
3   output: point where trajectories of neighborhood split up
4
5   for each point in refT do
6     regionAroundP := create circular region around point with r = laneEps
7     neighborsInReg := count intersections of trajectories with regionAroundP
8
9     if neighborsInReg <= 0.75 * |N| && idx(point) > 0.33 * refT.pointLength then
10      return point
11    end
12  end
13
14  return None

```

Listing 6.3: Pseudocode Split-Punkt Bestimmung

Zur Bestimmung der Regionen um einen Punkt und der Schnittpunkte der Trajektorien mit diesem Bereich, wird die JTS Topology Suite¹ (JTS) eingesetzt. Der Split-Punkt einer Nachbarschaft ist jener Punkt, in dessen Umfeld die Anzahl der Trajektorien erstmals unter 75% fällt. Er ist zudem nur gültig, wenn er nicht im ersten Drittel der Referenz-Trajektorien liegt.

¹ JTS Topology Suite: <https://locationtech.github.io/jts/>

Nachdem der Split-Punkt einer Nachbarschaft gefunden wurde, werden alle in ihr beinhalteten Bewegungsbahnen an jenem Trajektorie-Punkt geteilt, der dem Split-Punkt am nächsten ist. Die so neu erstellten Trajektorien bilden ein neues Cluster. In Abbildung 6.9 c) sind nun alle Spurcluster dargestellt, welche nach diesem Schritt existieren. Die zwei Cluster der Abbiegespuren, welche in Abbildung 6.8 b) noch fehlten, sind nun auch vorhanden.

Mithilfe der oben vorgestellten Verfahren können Trajektorie-Cluster entdeckt werden, welche die einzelnen Fahrspuren in einer Aufnahme beschreiben. Im nächsten Abschnitt wird darauf eingegangen, wie aus diesen Clustern Spur-Geometrien abgeleitet werden.

6.3 Spur-Geometrie Bestimmung

Nachfolgend wird beschrieben, wie aus den bereits identifizierten Trajektorie-Clustern Geometrie-Informationen der Fahrspuren abgeleitet werden. Hierzu sind primär drei Schritte notwendig: Bestimmung der Spur-Mittellinien, Bestimmung der Spurhüllen und anschließend die Partitionierung sich überlagernder Spuren. Hinzu kommen weitere kleine Schritte, welche ebenfalls nachfolgend beschrieben werden.

6.3.1 Ausfilterung von Spurwechselvorgängen

Bevor mithilfe der im vorherigen Schritt gewonnenen Trajektorie-Cluster Mittellinien von Fahrspuren bestimmt werden können, müssen diese nochmals vorverarbeitet werden. Die einzelnen Cluster enthalten teilweise Bewegungsbahnen, welche Spurwechselvorgänge oder andere Abweichungen von einer Fahrspur beschreiben. Diese Trajektorien müssen, so weit wie möglich, entfernt werden, da sie die anschließende Geometrie-Bestimmung negativ beeinträchtigen. Die Trajektorien eines Clusters sollten eindeutig einer realen Fahrspur zuzuordnen sein. In Abbildung 6.10 sind beispielhaft zwei Cluster dargestellt, welche eine Vielzahl an Spurwechselvorgängen enthalten.

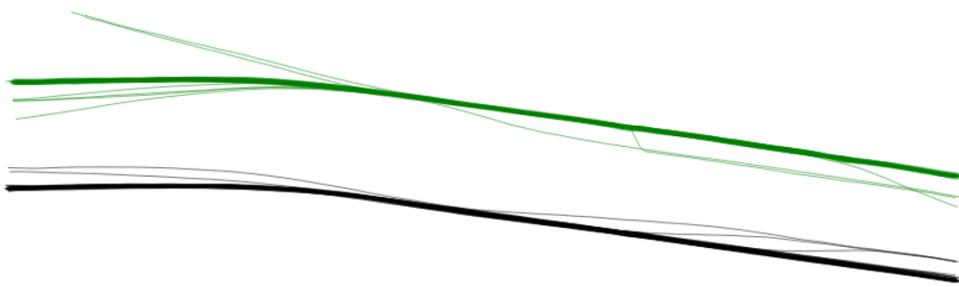


Abbildung 6.10: Trajektorie-Cluster mit Spurwechselvorgängen

Die Grundidee, welche der Ausfilterung der Abweichungen zugrunde liegt, ist, jene Trajektorien aus einem Cluster zu entfernen, welche eine überdurchschnittlich hohe mittlere Distanz zu allen anderen Trajektorien des Clusters besitzen. Als Distanzmaß wird erneut die LCSS-Distanz verwendet. Ein ähnlicher, Distanz-basierter Ausreißer-Detectionsansatz wird in [Mirge et al., 2017] vorgestellt. Der in dieser Arbeit verwendete Algorithmus ist in Listing 6.5 beschrieben.

```

1 algorithm filterCluster:
2   input: unfiltered trajectories of cluster: trajsIn
3   output: filtered trajectories of cluster
4
5   meanTrajectoryDistances :=
6   for each traj in trajsIn do
7     yield mean LCSS distance of traj to all other trajectories of trajsIn

```

```

8     end
9
10    clusterCmpVal := select median of meanTrajectoryDistances as comparison value
11
12    resultTrajs :=
13      for each traj in trajsIn do
14        if meanDist of traj < 1.5 * clusterCmpVal then
15          yield traj
16        end
17      end
18
19  return resultTrajs

```

Listing 6.4: Pseudocode Cluster Post-Processing

Das gewählte Verfahren ist einfach, eignet sich aber gut, um das gewünschte Ziel zu erreichen. Die in Abbildung 6.11 dargestellten Ergebnisse zeigen dies. Aus den in Abbildung 6.10 enthaltenen Clustern wurden alle Trajektorien mit Spurwechselvorgängen oder Abweichungen entfernt. Die Effektivität konnte auch durch die Anwendung auf andere Datensätze bestätigt werden. Wenn in einem Cluster viele Trajektorien Abweichungen von einer Spur besitzen, ist es möglich, dass der Algorithmus diese nicht vollständig entfernt. Bei einer geringen Anzahl von Abweichungen oder Ausreißer arbeitet er allerdings zuverlässig.

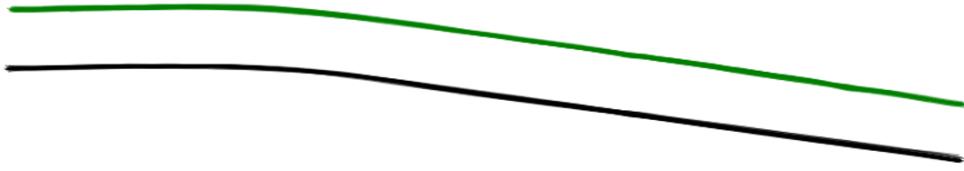


Abbildung 6.11: Trajektorie-Cluster ohne Spurwechselvorgänge

Zu beachten ist, dass die Zeitkomplexität des Verfahrens bei großen Trajektorie-Clustern nicht unerheblich ist. Dies ist auf die Komplexität des LCSS Distanzmaßes zurückzuführen. Bei der Untersuchung alternativer Vorgehensweise wurde allerdings klar, dass es grundlegend schwer ist, das Problem der Ausreißer-Erkennung performant zu lösen. In [Meng et al., 2018] werden hierzu diverse Distanz- und Dichte-basierte Arbeiten vorgestellt und verglichen. Da diese alle allerdings nicht weniger komplex sind und häufig die verfolgten Ziele über die hier geforderten hinausgehen, wurde entschieden den oben beschriebenen Ansatz beizubehalten. Durch die Parallelisierung der Berechnung der mittleren Abstände konnte die Performance des Ansatzes außerdem nochmals deutlich gesteigert werden.

6.3.2 Bestimmung der Spurmittellinien

Nachdem die Trajektorie-Cluster nun weitestgehend von Fahrspurwechseln und anderen Abweichungen bereinigt wurden, beschreiben die verbleibenden Bewegungsbahnen eines

Clusters einer Fahrspur. Anhand dieser Trajektorien können daher nun die Mittellinie der Spuren bestimmt werden.

Die gesuchte Mittellinie verläuft durch die Mitte des entsprechenden Trajektorie-Clusters. Die einzelnen Bewegungsbahnen besitzen jeweils leichte Abweichungen von dieser „*Ideallinie*“. Als Spurmittellinie – wie von Hu et al. [2005] vorgeschlagen – jene Trajektorie zu wählen, welche die kleinste Summe der Distanzen zu allen anderen Trajektorien eines Clusters besitzt, ist nicht praktikabel, da die so gefundene Trajektorie nicht über ihre komplette Länge in der Mitte des Clusters verlaufen muss und auch nicht zwangsläufig so lang ist, wie die anderen Trajektorien des Clusters. Auf eine Bestimmung der Mittellinie mittels linearer oder polynomialer Regression, wie beispielsweise in [Chen et al., 2014] oder [Mélo et al., 2006], wurde ebenfalls verzichtet. Der Grund hierfür ist, dass einerseits die Komplexität und Form der Fahrbahnverläufe nicht bekannt ist und andererseits das Erlernen der Spurrepräsentationen auf diese Weise sehr aufwendig ist.

Der in dieser Arbeit gewählte Ansatz zur Bestimmung der Mittellinien macht sich die von Atev et al. vorgestellte Notation einer relativen Position innerhalb einer Trajektorie zu nutze, welche in Gleichung 6.4 und 6.5 gegeben ist. Die Koordinaten der Spurlinien ergeben sich aus den Mittelwerten von Trajektorie-Punkten, welche sich alle an der selben relativen Position befinden. Vorteil der Verwendung der relativen Positionen ist, dass Punkte auf Trajektorien mit einer Position p_r auch dann auf der selben Höhe liegen, wenn die Bahnen aufgrund von Oszillationen um die Spurmitte unterschiedliche Längen haben. Der verwendete Algorithmus ist nachfolgend beschrieben.

```

1 algorithm calculateCenterline:
2   input:  trajectories of cluster: trajsIn
3   output: centerline of lane
4
5   meanTrajLength := calculate mean point length of trajectories
6   relPositions := define range with relative positions of size meanTrajLength
7       and step-size (1 / meanTrajLength)
8
9   centerline :=
10  for each relP in relPositions do
11    pointsAtRelPos := get points at relP for each traj in trajsIn
12    yield mean of all points in pointsAtRelPos
13  end
14
15 return centerline

```

Listing 6.5: Pseudocode Cluster Post-Processing

Da die Form und der Verlauf von Trajektorien eines Clusters sich nur wenig unterscheiden, liefert dieses Vorgehen gute Ergebnisse. In Abbildung 6.12 a) ist beispielhaft dargestellt, wie eine Mittellinien innerhalb eines Trajektorie-Clusters verlaufen. In 6.12 b) sind alle Spurmittellinien der Neckartor-Kreuzung, welche auf die oben beschriebene Weise bestimmt wurden, abgebildet.

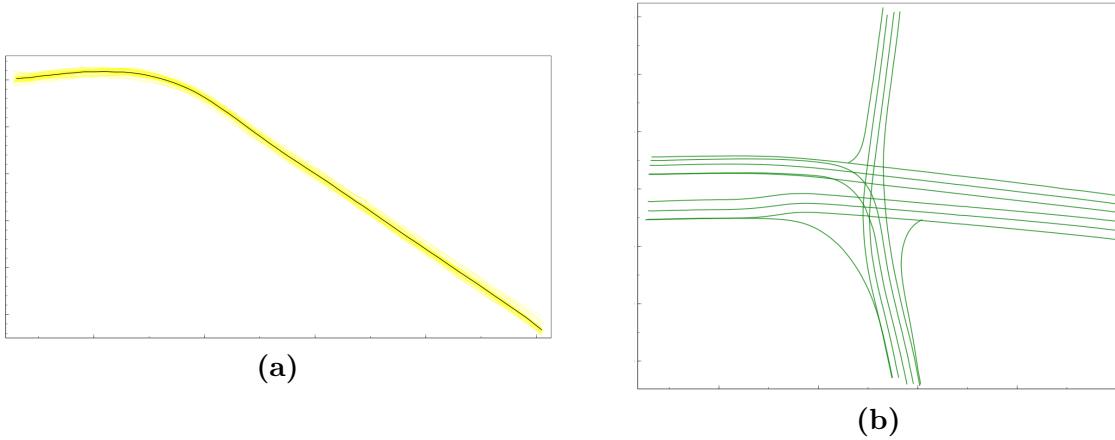


Abbildung 6.12: Spurmittellinie in einem Trajektorie-Cluster a), Spurmittellinien Neckartor-Kreuzung b)

Die Mittellinien werden, da es sich bei ihnen ebenfalls um Sequenzen von 2D-Koordinaten handelt, über Trajektorie-Objekte repräsentiert (siehe Abbildung 6.2).

6.3.3 Angleichung benachbarter Spur-Enden

Bevor für jede Spurmittellinie eine Hülle definiert wird, welche die Geometrie der Fahrspur definiert, werden benachbarte Spur-Enden aneinander angeglichen. Diese sollen immer auf der selben Höhe beginnen beziehungsweise enden, um beispielsweise die Positionen von Fahrzeugen auf benachbarten Spuren bei der Spurwechsel-Analyse besser vergleichen zu können.

Benachbarte Fahrspuren können einen deutlich sichtbaren Versatz an ihren Anfängen und Enden besitzen, was beispielsweise in Abbildung 6.14 a) dargestellt ist. Diese Unterschiede entstehen primär, da Fahrzeuge am Rand der Aufnahme verschieden schnell erkannt werden und so deren Trajektorien unterschiedlich früh beginnen. Für das Angleichen der Spur-Enden sind grundlegend drei Schritte notwendig: Finden von benachbarten Enden, Bestimmung der längsten Spur im Bereich der Enden und schließlich das Angleichen aller Spuren auf die Höhe der vorher bestimmten “äußersten” Mittellinie.

Gruppen benachbarter Spur-Enden werden auf rekursive Weise gefunden. Es wird eine Spur gewählt und im Bereich ihres Starts beziehungsweise Endes nach unmittelbaren Nachbarn gesucht. Das hierzu verwendete Verfahren ist in Abschnitt 6.3.4 beschrieben. Für jede der so gefundenen Nachbarspuren wird wiederum nach deren Nachbarn gesucht. Auf diese Weise ergibt sich eine Spur-Gruppe.

Da das Weltkoordinaten-System in den verwendeten Aufnahmen keine feste Orientierung und keinen fixen Ursprung besitzt, kann die in einer Spur-Gruppe am weitesten außen liegende Spur nicht identifiziert werden, indem die Koordinaten der Start- oder Endpunkte

der Spuren verglichen werden. Bestimmt wird die “äußerste” Mittellinie daher, indem durch jeden Endpunkt der Spuren einer Gruppe eine orthogonale Gerade gelegt wird. Die Spur deren zugehörige orthogonale Gerade die wenigsten Schnittpunkte mit anderen Spur-Mittellinien besitzt, liegt am weitesten außen. Abbildung 6.13 veranschaulicht das Vorgehen.



Abbildung 6.13: Konzept der Spur-Enden-Angleichung

Die so bestimmte Gerade g ist zudem gleichzeitig, die Linie, auf welche die anderen Spur-Endpunkte projiziert werden. Hierzu wird eine Orthogonalprojektion der Endpunkte auf die Gerade g durchgeführt, welche in Gleichung 6.8 definiert ist. \vec{r}_0 entspricht hierbei einem Stützvektor auf der Geraden g und \vec{u} dem Richtungsvektor.

$$P_g(\vec{x}) = \vec{r}_0 + \frac{(\vec{x} - \vec{r}_0) \cdot \vec{u}}{\vec{u} \cdot \vec{u}} \vec{u} \quad (6.8)$$

Das Ergebnis der Spur-Angleichung ist in Abbildung 6.14 b) dargestellt.

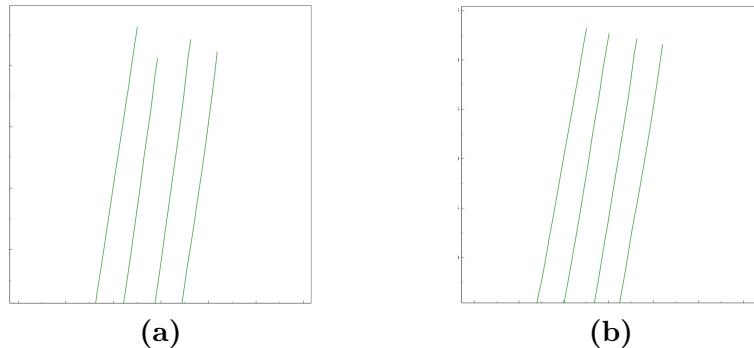


Abbildung 6.14: Mittellinien ohne Angleichung a) und mit Angleichung b)

6.3.4 Bestimmung der Spurhüllen

Nachdem die Mittellinien der Fahrspuren bestimmt und ihre Enden aneinander angeglichen wurden, wird nun für jede Spurlinie eine Hülle definiert. In folgendem Abschnitt wird beschrieben, wie diese erstellt werden.

Die Spurhülle bestimmt die Dimension einer Fahrbahn. In dieser Arbeit verlaufen sie idealerweise entlang realer Begrenzungslinien, welche zwei Spuren voneinander oder eine Fahrbahn von einem Seitenstreifen et cetera trennen. Eine Mittellinie und eine Hülle beschreiben zusammen die Geometrie einer Fahrspur. Diese Geometrie soll die realen Dimensionen einer Spur möglichst genau abbilden. Aus diesem Grund ist es nicht möglich, die Hüllen lediglich auf Basis von statistischen Informationen der Trajektorie-Cluster zu bestimmen, wie das beispielsweise in [Weiming Hu et al., 2006] oder [Morris and Trivedi, 2011] gemacht wird. Der in dieser Arbeit verwendete Ansatz zur Bestimmung der Spurhüllen basiert daher lediglich auf den Spurmittellinien und nutzt ihre relative Lage zueinander. Ansätze hierzu stammen aus den Arbeiten von [Hsieh et al., 2006] und [Makris and Ellis, 2005], welche in Abschnitt 4.2 vorgestellt wurden.

Die grundlegende Idee, auf welcher die Bestimmung der Spurhüllen basiert, ist konzeptiell in Abbildung 6.17 dargestellt. Für zwei parallel zueinander verlaufende Spuren l_1 und l_2 werden die Hüllen über den Abstand zwischen ihren Mittellinien bestimmt. Besitzen die Linien den Abstand d zueinander, so beträgt die Breite der Spuren ebenfalls d . Der Abstand e_d zwischen einer Mittellinie und einer Hüll-Linie beträgt folglich $1/2 d$.

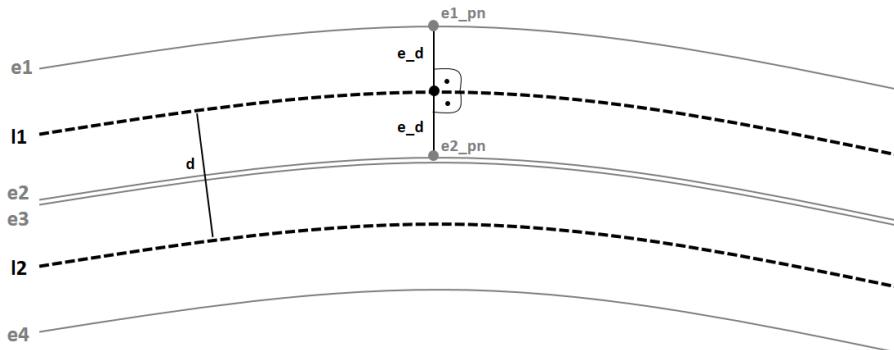


Abbildung 6.15: Bestimmung Spurhüllen

Die oben beschriebene Definition einer Bahn-Geometrie wird im Spurerkennungs-Modul über eine Klasse *LaneGeometry* repräsentiert. Diese ist in Abbildung 6.16 dargestellt. Das Feld *variance* entspricht dem Abstand e_d zwischen der *centerline* und den Hüll-Linien.

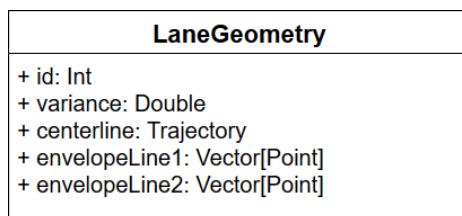


Abbildung 6.16: Aufbau LaneGeometry Klasse

Einen ähnlichen Ansatz zur Bestimmung von Spurbegrenzungslinien verwenden auch Hsieh

et al. in ihrer Arbeit. Sie gehen jedoch, da die von ihnen verwendeten Aufnahmen von einer statischen Kamera über einer Autobahn stammen, davon aus, dass alle Fahrspuren parallel zueinander verlaufen. Da das in dieser Arbeit entwickelte Spurerkennungs-Modul mit unterschiedlichen Aufnahmen und Straßentopologien umgehen können muss, kann diese Annahme nicht getroffen werden. Nachfolgend werden die Schritte vorgestellt, welche angewandt werden, um in einer Menge von Spuren jene zu finden, welche parallel zueinander verlaufen. Auf deren Basis werden anschließend die Spurbreiten bestimmt und die Hüll-Linien definiert.

Identifikation paralleler Fahrspuren

Um in einer Menge von Fahrspuren, welche über Mittellinien repräsentiert werden, jene zu finden, die parallel zueinander verlaufen, werden in einem ersten Schritt benachbarte Spur-Paare gesucht. Zwei Fahrspuren l_1 und l_2 gelten als benachbart, wenn sich der Start oder das Ende von Spur l_2 in einem Bereich mit Radius σ um den Start von l_1 befindet. Der Wert für σ wurde auf Basis der in Deutschland geltenden „*Richtlinien für die Anlage von Autobahnen*“ (RAA) [FGSV, 2008] und der „*Richtlinien für die Anlage von Landstraßen*“ (RAL) [FGSV, 2012] bestimmt. Diese Regelwerke spezifizieren die in der Bundesrepublik derzeit zulässigen Straßenquerschnitte. Die in ihnen festgelegten Spurbreiten variieren zwischen 3.25 und 3.75 Metern. Um auch besonders breite benachbarte Spur-Paare identifizieren zu können, wurde für σ der Wert 4m gewählt.

Die auf diese Weise gefundenen Trajektorie-Paare starten oder enden als benachbarte Spuren. Der Algorithmus welcher prüft, ob zwei Spuren tatsächlich parallel zueinander verlaufen, ist in Listing 6.6 beschrieben. Die zu vergleichenden Mittellinien werden jeweils in eine Reihe von Richtungsvektoren umgewandelt. Anschließend werden paarweise die Winkel zwischen zwei Vektoren berechnet und die Ergebnisse gemittelt. Liegt die sich so ergebende Abweichung unter einem Grenzwert δ , so handelt es sich um parallele Spuren. Zusätzlich wird geprüft, ob die Spuren eine ähnliche Länge besitzen.

```

1 algorithm lanesAreParallel:
2   input: lane-centerline: l1, lane-centerline: l2, delta
3   output: True or False
4
5   inOppositeDirections := check if l1 and l2 run in opposite directions
6   if inOppositeDirections then
7     reverse points of centerline l2
8   end
9
10  l1DirectionVec := calculate direction vectors for l1
11  l2DirectionVec := calculate direction vectors for l2
12  dirDifferences := calculate pairwise the angle between two direction vectors
13
14  meanDiff := calculate mean angle of deviation between l1 and l2
15  lengthDiff := calculate difference between length of l1 and l2
16
17  return meanDiff < delta && lengthDiff >= 0.8

```

Listing 6.6: Pseudocode Überprüfung der Parallelität zweier Mittellinien

Für δ wurde experimentell der Wert 0.1 bestimmt. In den verschiedenen Test-Datensätzen konnten so zuverlässig parallele Spur-Paare identifiziert werden. Anhand dieser werden im nächsten Schritt die Spurhüllen bestimmt.

Berechnung der Hüllen

Bevor für eine Mittellinie die sie umgebende Spurhülle bestimmt werden kann, muss die Breite der Spur ermittelt werden. Diese ergibt sich, für die im vorherigen Schritt bestimmten parallelen Spurpaare, aus deren mittleren Abstand zueinander. Verlaufen zu einer Spurlinie zwei Bahnen parallel, werden die Abstände zu beiden berechnet und der kleinere wird als Spurbreite gewählt. Allen Bahnen, welche keine parallele Nachbarspur besitzen, wird das Minimum der im vorherigen Schritt bestimmten Spurbreiten zugeordnet. Falls sich in einer Aufnahme keine parallelen Spuren befinden, wird für die Spurbreite ein Standartwert von 3.5 Metern verwendet, welcher sich ebenfalls aus den RAA und RAL Richtlinien ableitet.

Nachdem auf die oben beschriebene Weise Breiten für alle Fahrspuren bestimmt wurden, können die Hüll-Linien berechnet werden. Hierzu werden für jeden Punkt der Mittellinie zwei zugehörige Hüll-Punkte berechnet. Das hierzu verwendete Vorgehen ist in Abbildung 6.17 dargestellt.

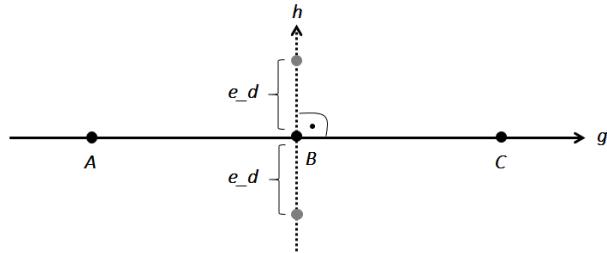


Abbildung 6.17: Bestimmung Spurhüllen

Um die Hüll-Punkte für einen Punkt B einer Mittellinie zu bestimmen, wird eine Gerade g durch die Punkte A und C gelegt, welche sich vor und hinter B befinden.

$$g : \vec{x} = \overrightarrow{OA} + t \cdot \overrightarrow{AC} \quad (6.9)$$

Anschließend wird durch B eine Gerade h gelegt, welche orthogonal zu g verläuft.

$$h : \vec{x} = \overrightarrow{OB} + t \cdot \hat{X} \quad (6.10)$$

Für ihren Richtungsvektor \hat{X} , welcher sich aus \overrightarrow{AC} ergibt, gilt $\overrightarrow{AC} \cdot \hat{X} = 0$. Da \hat{X} ein Einheitsvektor ist, können die Hüll-Punkte, welche auf h liegen und den Abstand e_d von

B besitzen, einfach bestimmt werden, indem e_d beziehungsweise $-e_d$ für t in die Gleichung von h eingesetzt wird.

Auf diese Weise werden die Hüll-Linien für alle Fahrspuren bestimmt. In Abbildung 6.19 sind die Ergebnisse der Spur-Geometrie-Bestimmung dargestellt. Teil a) zeigt die Spurmittellinien mit ihren sie umgebenden Hüllen. Teil b) zeigt die in die TrackerApplication eingefügten Spuren.

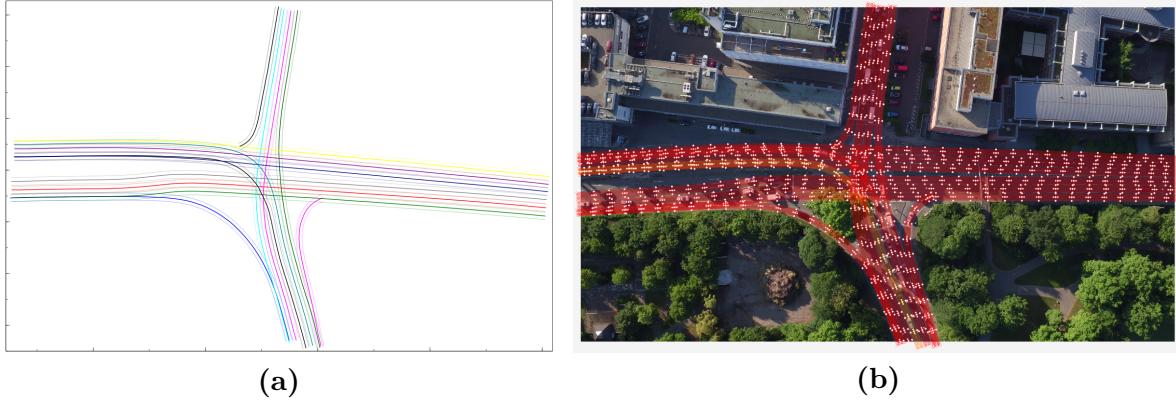


Abbildung 6.18: Plot Spurmittellinien und Hüllen a), Ergebnis Spur-Geometrien in TrackerApplication b)

Die obige Abbildung zeigt, dass die berechneten Spur-Geometrien in den meisten Fällen bereits gut mit den realen Spur-Dimensionen übereinstimmen. Bevor die Spuren partizipiert werden, um Überlagerungen von Fahrspuren zu entfernen, werden in einem weiteren Schritt “unechte” Spur-Geometrien entfernt.

Entfernung unechter Spur-Geometrien

Spur-Geometrien sind grundsätzlich dann “unecht”, wenn sie keine real existierende Fahrspur beschreiben. Am häufigsten treten unechte Fahrspuren in Form von zu kurzen Geometrien auf, oder in Form von Geometrien, welche einen Spurwechsel beschreiben. Diese zwei Arten von Spurgeometrien werden daher identifiziert und entfernt.

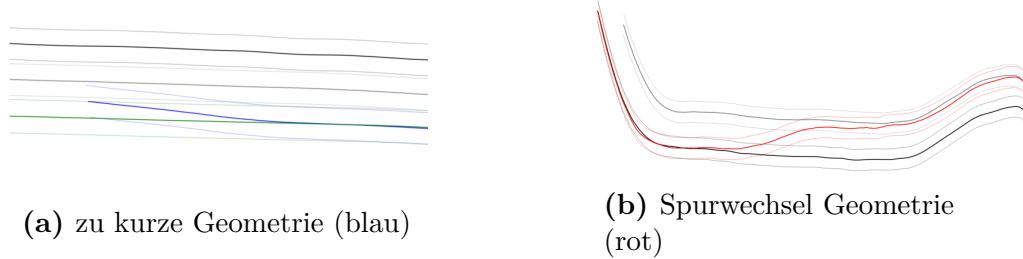


Abbildung 6.19: Beispiele “unechter” Spur-Geometrien

Zu kurz ist eine Spur-Geometrie dann, wenn sie einen Streckenabschnitt beschreibt, welcher gleichzeitig noch von einer anderen, längeren Spur beschrieben wird. Sie wird daher gänzlich oder fast gänzlich überlagert. Diese Art von unechten Spuren ist leicht zu identifizieren. Es wird für jede Spur geprüft, ob eine andere Spur existiert, welche sie zu 95% oder mehr überlagert. Wie die Überlagerung von zwei Spuren identifiziert wird, ist in Abschnitt 6.3.5 beschrieben.

Geometrien, welche nicht eine Spur sondern den Wechsel zwischen zwei Spuren darstellen, können existieren, wenn die Trajektorie-Daten viele Spurwechselvorgänge enthalten und in der Cluster-Analyse für diese ein extra Cluster erstellt wird. Um diese Spuren zu identifizieren, müssen Kriterien eines Spurwechsel-Vorgangs definiert werden. Diese zeichnen sich primär dadurch aus, dass sie auf einer Spur *A* beginnen und auf einer Spur *B* enden, wobei die Spuren *A* und *B* parallel zueinander verlaufen. Es kann zudem davon ausgegangen werden, dass sich weniger Fahrzeuge auf einer Spurwechsel-Spur bewegen, als auf den zwei eigentlichen Fahrspuren. Auf Basis dieser Eigenschaften, ergibt sich der nachfolgende Algorithmus.

```

1 algorithm isLaneChangeLane:
2   input: lane-geo: l1, sequence of other lane-geos: otherGeos
3   output: True or False
4
5   overlaysAtStart := find lanes in otherGeos that are overlaid by l1 at its start
6   overlaysAtEnd := find lanes in otherGeos that are overlaid by l1 at its end
7
8   if overlaysAtStart.nonEmpty && overlaysAtEnd.nonEmpty then:
9     a := identify more traveled parallel lanes in overlaysAtStart
10    b := identify more traveled parallel lanes in overlaysAtEnd
11    return a.nonEmpty && b.nonEmpty
12  else return false
13  end
```

Listing 6.7: Pseudocode Identifikation Spurwechsel-Spur

Eine Spurwechsel-Spur muss also im Bereich ihres Starts und ihres Endes jeweils mindestens eine andere mehr befahrene, parallele Fahrspur überlagern. Nachdem auf diese Weise die “unechten” Spur-Geometrien entfernt wurden, werden die übrigen partitioniert.

6.3.5 Partitionierung von Fahrspuren

Fahrspuren, welche mithilfe dieser Arbeit erkannt werden, kommen bei der Verkehrsanalyse zum Einsatz. Unter anderem können mit ihrer Hilfe Spurwechselvorgänge von Fahrzeugen untersucht werden. Damit eine solche Analyse sinnvoll ist, dürfen sich Fahrspuren nicht über einen größeren Bereich hinweg überlagern. Aus diesem Grund werden Spuren, welche in Teilen identisch mit anderen verlaufen, partitioniert. Die identischen Teile werden verworfen und nur die separaten beibehalten. Unproblematisch ist es, wenn sich Spuren lediglich kreuzen, ihre Überlagerung also geringfügig ist. In diesem Fall werden die Spuren nicht partitioniert. Es wird daher zwischen sich überlagernden und sich kreuzenden Spuren unterschieden. Nachfolgend wird das Verfahren zur Identifikation von sich überlagernden Fahrspuren und der Partitionierungs-Vorgang beschrieben.

Um Fahrspuren zuverlässig und sinnvoll partitionieren zu können, müssen grundlegend zwei Probleme bewältigt werden. Es müssen zuerst die sich überlagernden Spur-Paare und deren Schnittpunkte gefunden werden und anschließend muss für jedes Paar entschieden werden, welche Spur partitioniert wird und welche erhalten bleibt. Hierzu wird zuerst die Menge der Spur-Geometrien in drei Kategorien unterteilt:

- isolierte Fahrspuren
- primäre Fahrspuren
- sekundäre Fahrspuren

Isolierte Fahrspuren sind hierbei jene, welche keine Überschneidung mit anderen Spuren besitzen. Primäre Fahrspuren besitzen keine Überschneidungen untereinander, können sich aber mit anderen Spuren kreuzen oder von ihnen überlagert werden. In einer Menge von Spur-Geometrien bildet das größte Subset von parallel zueinander verlaufenden Spuren die primären Spuren. Sekundäre Fahrspuren sind all jene, welche weder isoliert noch primär sind. In Abbildung 6.20 a) sind die primären und sekundären Spur-Geometrien der Neckartor-Kreuzung dargestellt.

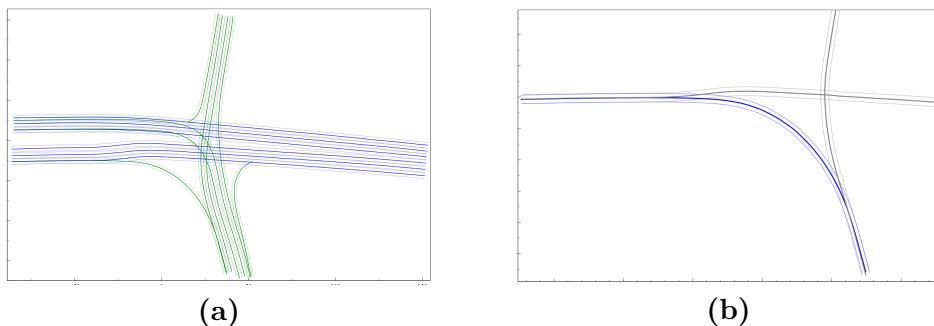


Abbildung 6.20: Primäre (blau) und sekundäre (grün) Spuren a), sich überlagernde und kreuzende Spur-Paare b)

Nachdem die Spuren, anhand der oberen Definition, in die drei Kategorien unterteilt wurden, werden anschließend die primären und sekundären Spuren nach sich überlagernden Paaren durchsucht. Zwei Spuren überschneiden sich, wenn, wie in Abbildung 6.21 zu sehen, die Mittellinie einer Spur innerhalb der Hülle einer anderen Spur liegt. Die Punkte b_1 und b_2 entsprechen hierbei den äußeren und inneren Grenzpunkten der Überschneidung der Mittellinie von l_2 mit der Hülle von l_1 . Zur Bestimmung der Schnittmenge wird wieder die JTS Topology Suite eingesetzt.

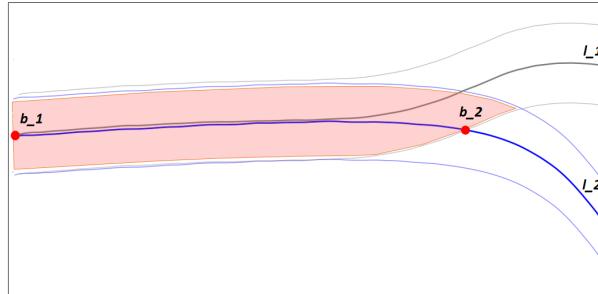


Abbildung 6.21: Überlagerung zweier Spur-Abschnitte

Zwei Spuren l_1 und l_2 kreuzen sich nicht nur, sondern überlagern sich, wenn der Abschnitt zwischen den Grenzpunkten b_1 und b_2 mindestens 10% der Spurlänge ausmacht. Auf diese Weise werden die sich überlagernden Spur-Paare und die zugehörigen Schnittpunkte bestimmt. Aus jeder Überschneidung ergeben sich zwei Spur-Paare. Im Fall von Abbildung 6.21 wird einerseits l_1 von l_2 überlagert und andererseits l_2 von l_1 . Daher enthält Abbildung 6.20 b) beispielsweise vier sich überlagernde Spur-Paare (blau-grau) und zwei sich kreuzende Paare (grau-grau).

Es kann vorkommen, dass in einem Bereich, in welchem sich zwei Spuren schneiden, die Mittellinie der einen Spur sich kurzzeitig außerhalb der Hülle der anderen befindet. Bei der Bestimmung der überlagerten Bereiche, wie oben beschrieben, würde dies dazu führen, dass statt einer zwei Überlagerungen erkannt werden. Um mit einer solchen Situation korrekt umzugehen, werden die identifizierten Schnittmengen zwischen zwei Spuren fusioniert, falls die Distanz zwischen zwei Abschnitten sehr gering ist. Hierrüber wird sichergestellt, dass eine Spur anschließend nicht fälschlicherweise zweimal oder zu früh beziehungsweise zu spät partitioniert wird.

Nachdem auf die oben beschriebene Weise die sich überlagernden Spur-Paare bestimmt wurden, wird anschließend entschieden, welche Spur der Paare partitioniert wird und welche erhalten bleibt. Zuerst wird hierzu überprüft, ob es sich bei einer der Geometrien um eine primäre Fahrspur handelt. Ist dies der Fall, so bleibt diese vollständig. Existiert in einem Paar keine primäre Spur, so werden verschiedene Eigenschaften der Fahrspuren untersucht, um eine Entscheidung zu treffen, welche Geometrie zu partitionieren ist. Der hierzu verwendete Algorithmus ist vereinfacht in Listing 6.8 dargestellt.

Grundlegen wird zuerst das Krümmungsverhalten der zwei Fahrspuren im Bereich ihres Schnittpunktes untersucht. Liegt die Differenz der Krümmungen der beiden Spuren oberhalb eines Grenzwertes δ , so wird jene Spur partitioniert, welche die höhere Krümmung besitzt. Die Annahme ist, dass es sich bei dieser Spur mit hoher Wahrscheinlichkeit um eine Abbiegespur et cetera handelt, welche in eine gerade Spur übergeht. Ist die Differenz der Krümmungen kleiner als δ , so verlaufen die zwei Spuren im Bereich des Schnittpunktes annähernd parallel. In diesem Fall wird entweder anhand der Länge oder der Anzahl der Fahrzeuge entschieden, welche Spur geteilt wird. Der Anwender der Tracking-Application kann das Verhalten des Algorithmus an dieser Stelle über einen Parameter steuern. Standardmäßig wird die Anzahl der Fahrzeuge als Kriterium verwendet.

```

1 algorithm selectLaneGeometryForPartitioning:
2   input: lane-geo: l1, lane-geo: l2, boundPoints: bounds
3   output: l1 or l2 based on curviness around bounds
4
5   innerBoundPoint := select inner bound point based on distance
6     from b1 and b2 to the edges of l1
7
8   l1Subset := get subset of l1 around innerBoundPoint
9   l2Subset := get subset of l2 around innerBoundPoint
10
11  l1CurvMea := estimate curvature of l1Subset
12  l2CurvMea := estimate curvature of l2Subset
13
14  if | l1CurvMea - l2CurvMea | > delta then
15    partition lane with bigger curvMea
16  else
17    if check length? then partition shorter lane
18    if check vehicle-count? then partition lane with less vehicles
19  end

```

Listing 6.8: Pseudocode Auswahl zu partitionierende Fahrspur

Im Fall der sich überlagernden Spurpaare in Abbildung 6.21 ist b_2 der *innerBoundPoint*. Zur Bestimmung der Krümmung einer Fahrspur in einem Bereich, siehe Listing 6.8 Zeile 11 + 12, wird der Winkel φ zwischen den Richtungsvektoren des Anfang und Endes der Teilspur berechnet. Er ergibt sich anhand Gleichung 6.11.

$$\varphi = \arccos \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} \quad (6.11)$$

Nachdem alle zu teilenden Spuren und die zugehörigen Grenzpunkte bestimmt wurden, folgt die eigentliche Partitionierung. Aus den Spur-Geometrien werden alle Bereiche entfernt, welche zwischen den zwei Grenzpunkten einer Überlagerung liegen. In Abbildung 6.21 wird so beispielsweise der rot gekennzeichnete Bereich der Spur l_2 zwischen b_1 und b_2 entfernt.

Abbildung 6.22 zeigt das Ergebnis der Spur-Partitionierung im Fall des Neckartor Datensatzes. Es wurden alle Spur-Überlagerungen entfernt.

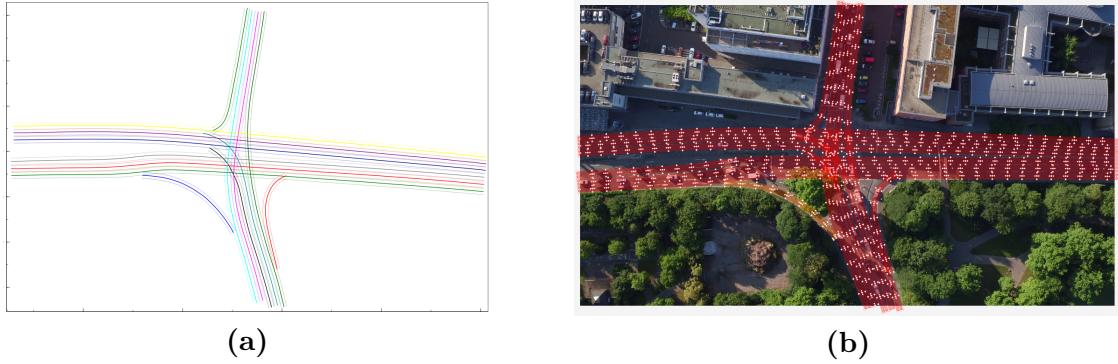


Abbildung 6.22: Plot partitionierte Spur-Geometrien a), Ergebnis in der TrackerApplication b)

Die so ermittelten Spur-Geometrien entsprechen in den meisten Fällen bereits dem gewünschten Ergebnis, da sie die realen Fahrspur-Verläufe gut wiederspiegeln und keine Überlagerungen mehr existieren. In einigen Szenarien kann es jedoch vorkommen, dass nach der Geometrie-Ermittlung und der Partitionierung die Fahrspur-Geometrien die reale Straßen-Topologie noch nicht korrekt abbilden. Aus diesem Grund werden die Spur-Geometrien in einem weiteren Schritt nochmals optimiert.

6.3.6 Optimierung der Spur-Geometrien

Die nach dem Partitionierungs-Schritt vorliegenden Spur-Geometrien besitzen in einigen Fällen noch Defekte. Das am häufigsten auftretende Problem ist, dass die Spur-Geometrien nicht breit genug sind. Abbildung 6.23 zeigt beispielsweise die Spur-Geometrien eines Straßen-Abschnitts, in welchem sich zu Beginn zwei parallele Fahrspuren befinden, welche sich am Ende in vier Spuren aufteilen. Es ist erkennbar, dass die zwei Fahrspuren zu Beginn nicht direkt aneinander angrenzen und daher in diesem Abschnitt nicht breit genug sind. In diesem Fall resultiert die zu niedrige Fahrbahnbreite aus der starken Überlagerung der vier Spur-Geometrien zu Beginn, was eine falsche Schätzung der Spur-Breite verursacht (siehe Abschnitt 6.3.4).

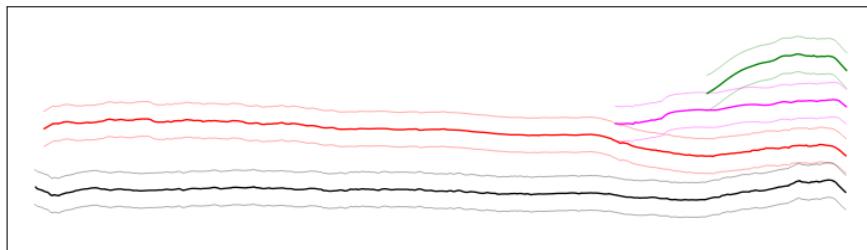


Abbildung 6.23: Beispiel zu schmale Spur-Geometrien

Um Effekte wie diesen zu korrigieren, werden nach der Partitionierung nun nochmals die Breiten aller Spur-Geometrien neu bestimmt. Im Gegensatz zu dem verwendeten Vorgehen aus Abschnitt 6.3.4, bei welchem für jede Spur eine feste Breite berechnet wurde, wird diese nun für jeden Punkt der Spur einzelt berechnet. Hierdurch wird es möglich, dass beispielsweise die rote und grüne Spur aus Abbildung 6.23 zu Beginn breiter sind, als an ihrem Ende.

Der zur Bestimmung der neuen Spur-Breiten eingesetzte Algorithmus hat vereinfacht gesehen den nachfolgenden Ablauf:

- Wähle eine Spur-Geometrie l
- Für jeden Punkt p der Mittellinie von l mit Position i :
 - Suche korrespondierende Punkte $P_k = \{pk_0 \dots pk_n\}$ auf benachbarten Spuren
 - Wähle einen benachbarten Punkt $pk_j \in P_k$
 - Verwende die Distanz zwischen p und pk_j als Spurbreite von l an Position i

Entscheidend bei diesem Verfahren ist es, den richtigen Punkt pk_j auszuwählen. Dieser muss grundsätzlich auf jener Spur liegen, welche im Bereich des Punktes p parallel zu l verläuft und ihr am nächsten liegt.

Nachdem auf diese Weise für eine Spur punktweise die neuen Breiten bestimmt wurden, werden diese noch mithilfe eines gleitenden Mittelwert Verfahrens geglättet. Anschließend werden neue Spurhüllen, mittels dem in Abschnitt 6.3.4 vorgestellten Vorgehen, erzeugt. Das Ergebnis der Geometrie-Optimierung für den Fall der in Abbildung 6.24 enthaltenen Spuren, ist nachfolgend dargestellt.

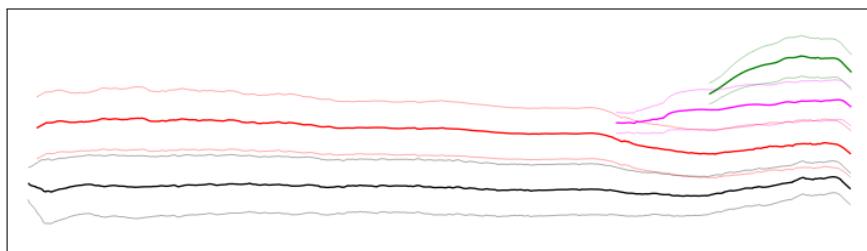


Abbildung 6.24: Beispiel zu schmale Spur-Geometrien

Die Fahrspurerkennung ist nach dem Optimierungsschritt abgeschlossen. Auf Basis der vorliegenden Spur-Geometrien können anschließend die Fahrspuren in die TrackerApplication eingefügt werden.

6.3.7 Anlegen der Fahrspuren in der TrackerApplication

Nachdem die Spur-Geometrien auf Basis der Fahrzeugtrajektorien bestimmt wurden, müssen diese in einem finalen Schritt noch in das von der TrackerApplication verwendete Datenmodell zur Repräsentation von Fahrspuren überführt werden. Dieses Vorgehen ist nachfolgend beschrieben.

Wie bereits zu Beginn der Arbeit erwähnt, wurden Fahrspuren in der TrackerApplication bislang manuell über die Benutzeroberfläche angelegt. Das hierzu verwendete Datenmodell, welches auch weiterhin für die automatisch erkannten Spuren verwendet wird, ist in Abbildung 6.25 dargestellt.

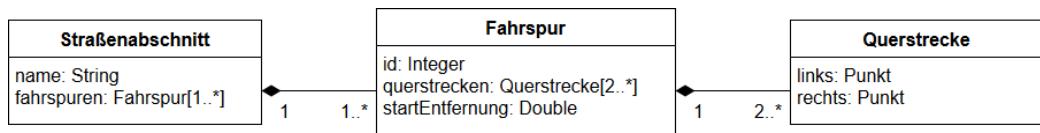


Abbildung 6.25: Datenmodell der Spurdefinition

Eine Fahrspur in einem Streckenabschnitt besteht aus mindestens zwei *Querstrecken*. Jede Querstrecke legt über zwei Punkte die Breite der Fahrspur auf einer bestimmten Höhe der Spur fest. Eine Fahrspur kann aus beliebig vielen Querstrecken bestehen, wodurch es möglich ist, die Fahrspuren beliebig detailliert zu definieren.

Die Fahrspuren und Querstrecken werden im Fall der automatisch erkannten Fahrbahnen anhand der Spur-Hüllen definiert. Über Punkt-Paaren der beiden Hülllinien (siehe Abbildung 6.16), werden Querstrecken erstellt. Da eine Fahrspur möglichst über die kleinste Menge an Querstrecken beschrieben werden soll, welche die Geometrie der Bahn zuverlässig abbildet, wird nicht für jedes Hüll-Punkt-Paar eine Querstrecke definiert. Die Anzahl und Distanz zwischen den Querstrecken richtet sich nach der Krümmung der Spuren. Zur Beschreibung einer geraden Fahrspur werden weniger Querlinien benötigt, als zur Beschreibung einer Kurve. Zur Bestimmung der Querstrecken werden daher die Hülllinien abschnittsweise untersucht und jeweils die Richtungsänderungen zwischen aufeinanderfolgenden Abschnitten betrachtet. Hierzu wird erneut die Gleichung 6.11 eingesetzt. Übersteigt die Richtungsänderung einer Hülle einen Grenzwert von $\phi = 0.02$, so wird eine neue Querstrecke erstellt. Andernfalls verläuft die Spur im untersuchten Bereich annähernd gerade. In Abbildung 6.26 ist das Ergebnis der Spurerstellung dargestellt. Es ist gut zu erkennen, dass im Fall der geraden Fahrspuren der Autobahn nur wenige Querstrecken zur Definition der Spur verwendet werden, und für die gekrümmten Spuren deutlich mehr.

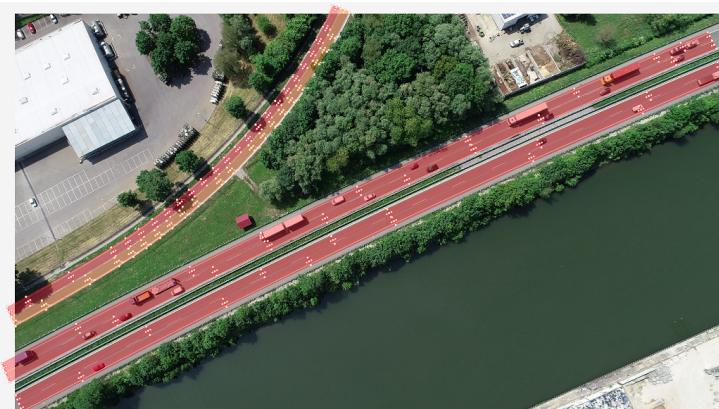


Abbildung 6.26: Angelegte Fahrspuren in der TrackerApplication

Ein Vorteil, welcher sich durch die Weiterverwendung des in Abbildung 6.25 dargestellten Datenmodells ergibt, ist, dass Benutzer weiterhin Spuren manuell anlegen können und automatisch detektierte Fahrbahnen sich nachträglich noch leicht modifizieren lassen. Falls eine vom Algorithmus erkannte Spur also beispielsweise in einem Abschnitt zu schmal oder zu breit ist, kann dies leicht vom Nutzer korrigiert werden.

7 Auswertung und Ergebnisse

Die Stärken, Schwächen und Ergebnisse des entwickelten Algorithmus werden im nachfolgenden Kapitel zusammengefasst, diskutiert und ausgewertet. Es wird zuerst abschnittsweise auf die drei primären Schritte des Verfahrens eingegangen. Anschließend werden weitere Beispielaufnahmen mit erkannten Fahrspuren gezeigt.

7.1 Evaluierung der Datenvorverarbeitung

Die Datenvorverarbeitung ist ein wichtiger Teilschritt bei der Erkennung von Fahrspuren. Nur wenn aus den Roh-Trajektorien die meisten Defekte entfernt wurden, können die nachfolgenden Schritte zuverlässig funktionieren. Die Mehrzahl der Defekte in den Roh-Trajektoriedaten werden durch stehende Fahrzeuge und fehlerhafte oder unterbrochene Fahrzeugverfolgungen verursacht. Die angewandten Schritte zur Entfernung der Anomalien sind in Abschnitt 6.1.2 beschrieben.

Dass die Entfernung von Ausreißern funktioniert, wurde bereits zum Teil in Abbildung 6.5 anhand des Neckartor-Trajektoriedatensatzes gezeigt. In Abbildung 7.1 sind nun die Trajektorien eines weiteren Datensatzes dargestellt, welcher von der Heilbronner-Straße in Stuttgart stammt.

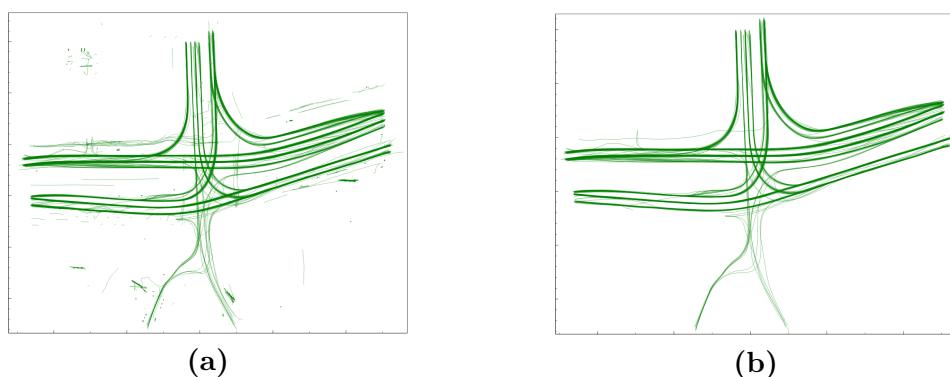


Abbildung 7.1: Ergebnis Vorverarbeitung Heilbronner-Straße

Die zwei obigen Plots zeigen gut, dass die vielen in a) vorkommenden Defekte entfernt wurden. Von den circa 1050 Roh-Trajektorien im ursprünglichen Datensatz bleiben nach

der Vorverarbeitung etwa 450 intakte Bewegungsbahnen übrig. Die Mehrzahl der Defekte in diesem Fall stammt von fehlerhaften Objekterkennungen und unterbrochener Fahrzeug-Detektionen, welche aufgrund der Verdeckung von Fahrspuren durch Bäume entstehen.

Ein problematisches Verhalten des Vorverarbeitungsschrittes wurde beim Testen der Spurkennung anhand eines Datensatzes aus Steinheim deutlich. In Abbildung 7.2 a) ist der untersuchte Straßenabschnitt dargestellt.



Abbildung 7.2: Straßenausschnitt Steinheim a), gekürzte Trajektorien b)

Kritisch an dieser Aufnahme ist, dass die Fahrzeuge, welche sich auf der oben linkes startenden Fahrbahn bewegen, am Anfang beziehungsweise Ende ihrer Fahrt sehr klein sind. Da Fahrzeuge in einer Aufnahme ab einer gewissen Größe nurnoch sehr unzuverlässig detektiert werden, brechen die Trajektorien in diesem Fall auf sehr unterschiedlichen Höhen ab. Problematisch ist nun, dass der in Abschnitt 6.1.2 beschriebene Algorithmus zur Entfernung unterbrochener Trajektorien, aufgrund der stark variierenden Start- und End-Positionen, auch die meisten Trajektorien auf der nach hinten verlaufenden Fahrbahn entfernt und somit die Spuren nicht erkannt werden können.

Da die Fahrzeuge im Bereich des Horizonts nur unzuverlässig erkannt werden, ist auch eine Fahrverhaltensanalyse mithilfe von Fahrspuren hier wenig sinnvoll. Es wurde daher entschieden dem Anwender die Möglichkeit zu geben, eine Horizont-Linie zu definieren. In einem ersten Vorverarbeitungsschritt werden alle Trajektorie-Punkte oberhalb dieser Linie entfernt. Das Ergebnis dieses Verfahrens ist in Abbildung 7.2 b) dargestellt. Dank der Beschneidung der Trajektorien bleiben diese in den nachfolgenden Verarbeitungsschritten erhalten und es können Spuren im gewünschten Abschnitt erkannt werden.

Mit Ausnahme des unerwünschten Verhaltens im Fall von Trajektorien, welche am Horizont sehr unterschiedliche Start- beziehungsweise End-Positionen besitzen, funktioniert die Datenvorverarbeitung gut und wie gewünscht. Mit ihrer Hilfe werden in einem Datensatz jene Trajektorien identifiziert, welche eine Bewegung durch den untersuchten Straßenabschnitt vollständig beschreiben. Voraussetzung dafür, dass nach der Datenvorverarbeitung noch genug Trajektorien vorliegen, welche weiterverarbeitet werden können, ist natürlich, dass auch in den Rohdaten genügend intakte, vollständige Bewegungsbahnen vorhanden sind.

7.2 Evaluierung der Clusteranalyse

Die Clusteranalyse der Trajektorien bildet die Grundlage für die anschließende Bestimmung der Spur-Geometrien. Sie ist daher ein kritischer Bestandteil der Spurerkennung. Der in dieser Arbeit eingesetzte Ansatz zur Gruppierung der Trajektorien und somit Identifikation von Fahrspuren wurde in Abschnitt 6.2.2 vorgestellt. Hier wurden auch bereits Ergebnisse der Clusteranalyse für die Datensätze *Entennest* und *Neckartor* vorgestellt.

Da der angewandte Clustering-Algorithmus die vollständigen Verläufe der Trajektorien vergleicht, ist die wichtigste Voraussetzung zur Identifikation eines Clusters, dass ausreichend Trajektorien vorliegen, welche die identische Bewegung durch einen Straßenabschnitt beschreiben. In Abschnitt 6.2.2 wurde erwähnt, dass ein Cluster mindestens aus fünf Trajektorien bestehen muss. Diese Untergrenze wurde gewählt, um nicht zu viele Cluster zu finden, welche eigentlich keine Fahrspur beschreiben. Würde die Grenze niedriger ange setzt werden, so würden beispielsweise vermehrt Cluster aus Trajektorien gebildet, welche Überholvorgänge beschreiben.

In den meisten Datensätzen anhand derer der in dieser Arbeit entwickelte Algorithmus getestet wurde, existierten ausreichend Trajektorien pro Fahrspur, um diese zuverlässig zu identifizieren. Eine Ausnahme stellt der Datensatz von der Heilbronner-Straße in Stuttgart dar, dessen Trajektorien bereits in Abschnitt 7.1 dargestellt sind. In Abbildung 7.1 b) wird deutlich, dass die Fahrspuren im unteren Bereich des Straßenabschnitts nur wenig befahren sind. Dies wirkt sich auch auf das Ergebnis der Clusteranalyse aus, welches in Abbildung 7.3 a) dargestellt ist. Zwar existieren ausreichend Trajektorien, welche sich von oben nach unten links bewegen, allerdings können keine Cluster aus den Trajektorien gebildet werden, welche sich unten rechts befinden. Die wenigen in diesem Bereich exis tierenden Trajektorien haben sehr unterschiedliche Bewegungsbahnen, weshalb hier kein Cluster identifiziert wird.

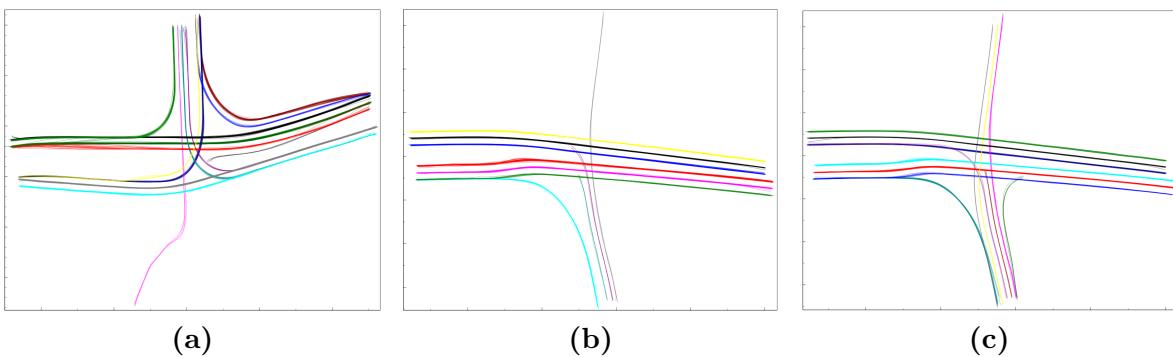


Abbildung 7.3: Trajektorie-Cluster Heilbronner-Straße und Neckator-Kreuzung

Die Plots in Abbildung 7.3 b) und c) zeigen das Ergebnis der Clusteranalyse für den Neckartor-Datensatz, aus welchem vor Anwendung des Algorithmus zufällig 50% der Trajektorien entfernt wurden. Auch hier wird deutlich, dass das Ergebnis maßgeblich von

der Anzahl der Trajektorien pro Spur abhängt. In Plot b) wurden viele vertikal verlaufende Trajektorien entfernt, weshalb hier nur wenige Spurcluster identifiziert werden. In Plot c) wurden hingegen trotz des Fehlens von 50% der Trajektorien fast alle Spurcluster erkannt.

Außer von der Anzahl der Trajektorien hängt das Ergebnis der Clusteranalyse maßgeblich von den gewählten Parametern des DBSCAN Algorithmus und des LCSS Distanzmaßes ab. Die in Abschnitt 6.2.2 definierten Standardparameter liefern in der Mehrzahl der Fälle gute Ergebnisse. Alle oben dargestellten Plots wurden mit diesen Parametern erstellt. Anderer Parameter müssen lediglich dann gewählt werden, wenn sich die Trajektorien unterschiedlicher Fahrspuren in einer Aufnahme stark überlagern. Dies ist beispielsweise im Datensatz Düsseldorf der Fall. Dessen Trajektorien sind in Abbildung 7.4 a) dargestellt. Um die Trajektorien in diesem Datensatz klar in Spurcluster unterteilen zu können, muss statt dem Standardwert $\epsilon_{DBSCAN} = 0.3$ der Wert $\epsilon_{DBSCAN} = 0.1$ verwendet werden. Die Ergebnisse der Clusteranalyse unter Einsatz dieses Parameters sind in Abbildung 7.4 b) dargestellt.

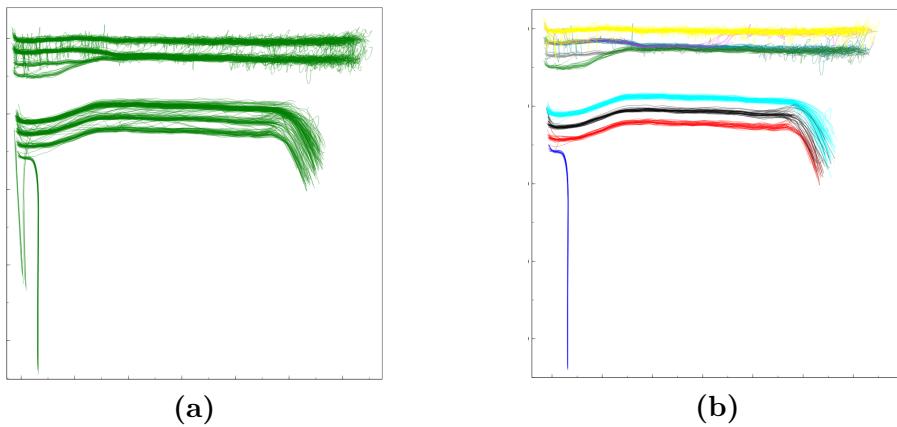


Abbildung 7.4: Trajektorien Düsseldorf Datensatz a), Spurcluster b)

Die Verwendung des Alternativen Wertes für ϵ_{DBSCAN} liefert in Fällen, in welchen sich Spuren stark überlagern, gute Resultate. Um das Verhalten des Algorithmus in dieser Hinsicht steuern zu können, wurde dem Nutzer die Möglichkeit gegeben ϵ_{DBSCAN} beim starten des Spurerkennungs-Jobs anzupassen.

Für die Clusteranalyse kann zusammenfassen festgehalten werden, dass sie gut funktioniert wenn ausreichen Trajektorien zur Beschreibung einer Spur vorliegen. Ihr Verhalten kann bei Bedarf zudem angepasst werden.

7.3 Evaluierung der Spur-Geometrie-Bestimmung

Nach der Clusteranalyse werden anhand der identifizierten Trajektoriecluster die Geometrien der Fahrspuren bestimmt. Ziel dieses Schrittes ist es, Spur-Geometrien zu definieren, welche den realen Spur-Abmaßen möglichst genau entsprechen. Hierzu werden in mehreren aufeinanderfolgenden Schritten die Spurmittellinien und Spurbreiten bestimmt. Die angewandten Verfahren sind in Abschnitt 6.3 beschrieben.

Wurden im vorherigen Schritt Trajektorie-Cluster für die in einer Aufnahme enthaltenen Spuren identifiziert, so können auf Basis dieser meistens sehr zuverlässige Spur-Geometrien abgeleitet werden. In Abbildung 7.5 sind beispielhaft Spurgeometrien aus drei unterschiedlichen Datensätzen dargestellt.

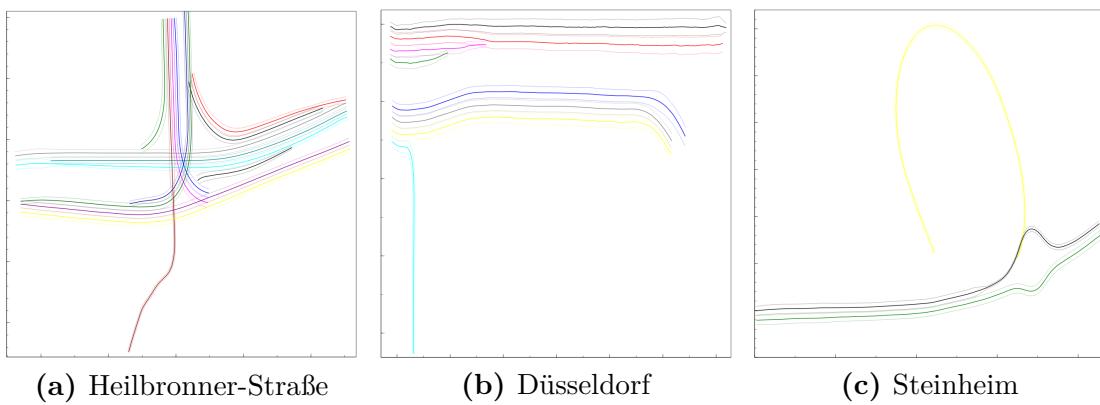


Abbildung 7.5: Ergebnisse Spur-Geometrie-Bestimmung

Anhand der abgebildeten Spurgeometrien ist ersichtlich, dass das in Abschnitt 6.3 beschriebene Verfahren gute Ergebnisse in unterschiedlichen Situationen liefert. Plot a) zeigt die Spur-Geometrien, welche im Fall des Datensatzes der Heilbronner-Straße bestimmt werden. Für jedes Spurcluster, welches in Abbildung 7.3 a) zu sehen ist, wurde eine Geometrie erstellt, deren Mittellinie und Breite den realen Spurverlauf gut abbildet. Es ist zudem zu sehen, dass die Fahrspuren partitioniert wurden, um die Überlagerung von Spuren zu vermeiden. Plot b) veranschaulicht, dass Spurgeometrien auch korrekt bestimmt werden können, wenn in einer Aufnahme Fahrbahnerweiterung existieren. Anhand von Plot c), welcher die für den Datensatz Steinheim berechneten Spur-Geometrien zeigt, wird außerdem deutlich, dass auch die Geometrien kreisförmiger Fahrspuren und Kreisverkehre bestimmt werden können.

Zwei Probleme, welche bei der Bestimmung der Spur-Geometrien auftreten können, sind in Abbildung 7.6 dargestellt. In diesem Fall stammen die Beispiele aus den Geometrien des Datensatzes Heilbronner-Straße.

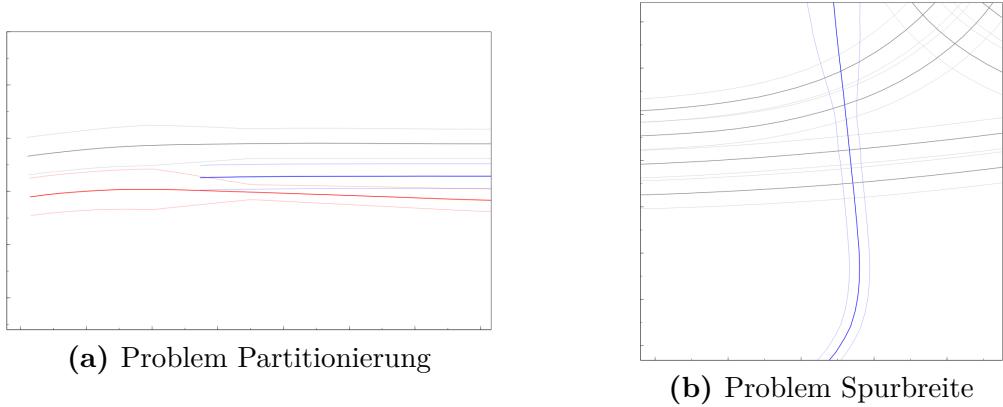


Abbildung 7.6: Auftretende Probleme in den Spur-Geometrien

Teil a) der Abbildung zeigt drei Fahrspuren, welche parallel zueinander verlaufen. Auf dem entsprechenden Straßenabschnitt der Heilbronner-Straße, auf welchem sich die drei Spuren befinden, verschmälert sich am linken Rand die Fahrbahn und die unterste Spur (rot) geht in die mittlere Spur (blau) über. Bei der Partitionierung der Spuren sollte daher eigentlich die mittlere Spur vollständig erhalten bleiben und das Ende der unteren entfernt werden. Wie oben zu sehen ist, wird allerdings die mittlere Spur partitioniert und die untere bleibt erhalten. Hier trifft der in Abschnitt 6.3.5 beschriebenen Algorithmus die “falsche” Entscheidung. In Abbildung 7.6 b) ist zu sehen, dass die blau markierte Spur in ihrem Verlauf nach unten hin schmäler wird. Dies entspricht nicht ihrem realen Verlauf auf der Straße. Diese Schmälerung tritt auf, da der Algorithmus zur Optimierung der Spurbreiten (siehe Abschnitt 6.3.6) an einem bestimmten Punkt den Abstand zur falschen Spur zur Neu-Berechnung der Breite verwendet. Die fehlerhafte Breite wird anschließend für den Rest der Spur weiterverwendet.

Die beiden oben abgebildeten und beschriebenen Fehler treten in ähnlicher Art immer wieder auf. Der Grund hierfür ist, dass es sowohl im Fall der Spur-Partitionierung als im Fall der Bestimmung der Spur-Breite sehr schwierig ist, ein Verfahren zu definieren, welches in allen Fällen die “richtige” Entscheidung trifft. Ein menschlicher Betrachter kann bei der Überlagerung zweier Spuren zwar meist intuitiv feststellen, welche partitioniert werden muss, einem Algorithmus dies beizubringen ist allerdings schwierig.

Die verwendeten Ansätze liefern in der Mehrzahl der Fälle die “richtige” Lösung, was in diesem Anwendungsszenario absolut ausreichend ist. Gerade auch bei der Partitionierung der Spuren gibt es an sich oft kein richtig und falsch. Welche Spur erhalten bleiben soll, ist häufig Interpretationssache. Daher, und da die oben beschriebenen “Fehler” leicht über die Oberfläche der TrackerApplication korrigiert werden können, wurden die verwendeten Algorithmen hier nicht weiter angepasst.

7.4 Ergebnisse der Spurerkennung

Nachdem in den Abschnitten oben auf die Fähigkeiten und Probleme der drei Hauptschritte des Spurerkennungs-Algorithmus eingegangen wurde, werden nun Ergebnisse in Form von Screenshots vorgestellt. In ihnen sind die automatisch erkannten und erstellten Spuren in der TrackerApplication zu sehen.

Die zwei ersten Screenshots, welche in Abbildung 7.7 dargestellt sind, zeigen die erkannten Spuren im Fall der Datensätze *Entennest* und *Düsseldorf*. Anhand der Aufnahmen ist ersichtlich, dass die erstellten Fahrspuren gut mit den realen Verläufen der Spuren auf der Fahrbahn übereinstimmen. Dies gilt auch im Fall des Datensatzes *Düsseldorf*, obwohl die hier detektierten Fahrzeugpositionen aufgrund des niedrigen Aufnahmewinkels nicht immer nahe der Fahrspurmitte liegen. Foobar

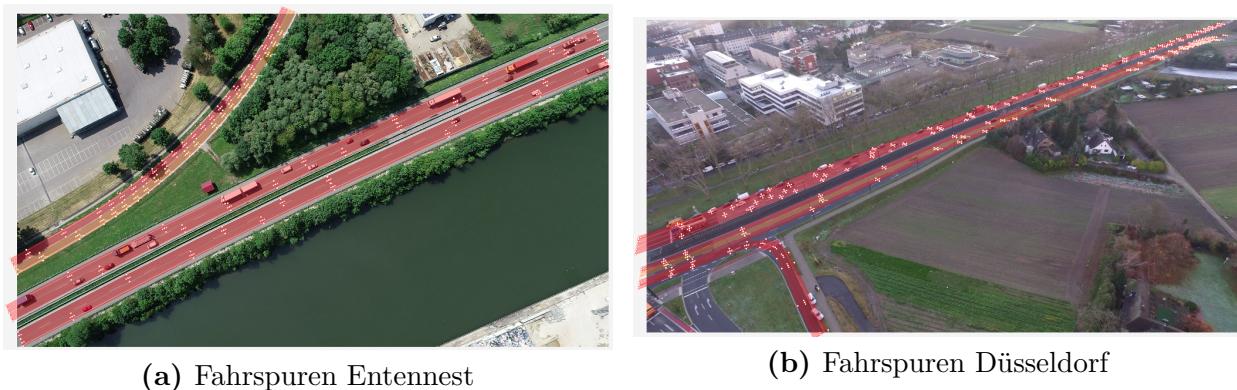


Abbildung 7.7: Erkannte Fahrspuren auf geraden Straßenabschnitten

In Abbildung 7.8 sind die erkannten Spuren der Datensätze *Neckartor* und *Heilbronner-Straße* dargestellt. Auch hier stimmen die Spur-Geometrien gut mit den realen Spur-Ausmaßen überein. Zudem wird deutlich, dass sowohl die Partitionierung der Spuren als auch das angleichen benachbarter Spurenden die gewünschten Resultate liefern. Im Fall der Heilbronner-Straße ist nun auch zu sehen, dass im unteren rechten Bereich keine Spur erkannt wird. Der Grund hierfür wurde in Abschnitt 7.2 beschrieben.



(a) Fahrspuren Neckartor



(b) Fahrspuren Heilbronner-Straße

Abbildung 7.8: Erkannte Fahrspuren auf Kreuzungen

Der Screenshot in Abbildung 7.9 zeigt die erkannten Fahrspuren im Datensatz *Steinheim*. Dieses Beispiel zeigt, dass Fahrspuren auch in Straßenabschnitten mit Kreisverkehren oder kreisförmigen Spuren erkannt werden.



(a) Fahrspuren Steinheim

Abbildung 7.9: Erkannte Fahrspuren Kreisverkehr

8 Fazit und Ausblick

Literaturverzeichnis

- M. Aly. Real time detection of lane markers in urban streets. In *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 7–12, 2008. ISBN 9781424425693. doi: 10.1109/IVS.2008.4621152. URL <https://ieeexplore.ieee.org/abstract/document/4621152/>.
- Anders Drachen. Introducing Clustering II: Clustering Algorithms - GameAnalytics, 2014. URL <https://gameanalytics.com/blog/introducing-clustering-ii-clustering-algorithms.html>.
- S. Atev, O. Masoud, and N. Papanikolopoulos. Learning Traffic Patterns at Intersections by Spectral Clustering of Motion Trajectories. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4851–4856. IEEE, oct 2006. ISBN 1-4244-0258-1. doi: 10.1109/IROS.2006.282362. URL <http://ieeexplore.ieee.org/document/4059186/>.
- S. Atev, G. Miller, and N. P. Papanikolopoulos. Clustering of vehicle trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 2010. ISSN 15249050. doi: 10.1109/TITS.2010.2048101.
- F. Bashir, A. Khokhar, and D. Schonfeld. Segmented trajectory based indexing and retrieval of video data. In *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, volume 3, pages II–623–6, 2003. ISBN 0-7803-7750-8. doi: 10.1109/ICIP.2003.1246757. URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.7768&rep=rep1&type=pdf>.
- Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004.
- D. Buzan, S. Sclaroff, and G. Kollios. Extraction and Clustering of Motion Trajectories in Video. apr 2004. URL <https://open.bu.edu/handle/2144/1545>.
- J. Chen, R. Wang, L. Liu, J. S. , , C. Control, and undefined 2011. Clustering of trajectories based on Hausdorff distance. *ieeexplore.ieee.org*, 2011. URL <https://ieeexplore.ieee.org/abstract/document/6066483/>.
- L. Chen, M. T. Özsü, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05*, page 491, New York, New York, USA, 2005. ACM Press. ISBN 1595930604. doi: 10.1145/1066157.1066213. URL <http://portal.acm.org/citation.cfm?doid=1066157.1066213>.

- Z. Chen, Y. Yan, and T. Ellis. Lane detection by trajectory clustering in urban environments. In *2014 17th IEEE International Conference on Intelligent Transportation Systems, ITSC 2014*, pages 3076–3081, 2014. ISBN 9781479960781. doi: 10.1109/ITSC.2014.6958184. URL <https://ieeexplore.ieee.org/abstract/document/6958184/>.
- G. Cookson, B. P. I. Research, undefined February, and undefined 2017. Inrix global traffic scorecard. *jschultheis.de*. URL http://jschultheis.de/wp-content/uploads/2018/02/INRIX_2017_Traffic_Scorecard_Report__German.pdf.
- Divyansh Dwivedi. Face Detection For Beginners – Towards Data Science, 2018. URL <https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9>.
- FGSV. *RAA - Richtlinien für die Anlage von Autobahnen*. Forschungsgesellschaft für Straßen- und Verkehrswesen (FGSV), Arbeitsgruppe Straßenentwurf., 2008. ISBN 978-3-939715-51-1. URL http://www.fgsv-verlag.de/catalog/product_info.php?products_id=2451.
- FGSV. *RAL - Richtlinien für die Anlage von Landstraßen*. Forschungsgesellschaft für Straßen- und Verkehrswesen (FGSV), Arbeitsgruppe Straßenentwurf., 2012. ISBN 978-3-86446-039-5. URL http://www.fgsv-verlag.de/catalog/product_info.php?products_id=3206.
- J. Gao. Clustering Lecture 4 : Density-based Methods. pages 1–16, 2012. URL https://cse.buffalo.edu/~jing/cse601/fa12/materials/clustering_density.pdf.
- George Seif. The 5 Clustering Algorithms Data Scientists Need to Know, 2018. URL <https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68>.
- R. Gopalan, T. Hong, M. Shneier, and R. Chellappa. A Learning Approach Towards Detection and Tracking of Lane Markings. *Intelligent Transportation Systems , IEEE Transactions on*, 13(3):1088–1098, 2012. ISSN 15249050. doi: 10.1109/TITS.2012.2184756. URL <https://ieeexplore.ieee.org/abstract/document/6155090/>.
- H. Grabner, M. Grabner, H. B. Bmvc, and undefined 2006. Real-time tracking via online boosting. *grabner.family*, 2006. URL <http://grabner.family/helmut/papers/Grabner2006RealTimeTracking.pdf>.
- G. Hamerly, C. E. A. in neural information Processing, and undefined 2004. Learning the k in k-means. *papers.nips.cc*. URL <http://papers.nips.cc/paper/2526-learning-the-k-in-k-means.pdf>.
- P. Hemmerle. Empirische physikalische Eigenschaften des übersättigten innerstädtischen Verkehrs und Energieeffizienz von Fahrzeugen. 2016. URL <https://d-nb.info/1122559259/34>.

- J.-W. Hsieh, S.-H. Yu, Y.-S. Chen, and W.-F. Hu. Automatic Traffic Surveillance System for Vehicle Tracking and Classification. *IEEE Transactions on Intelligent Transportation Systems*, 7(2):175–187, jun 2006. ISSN 1524-9050. doi: 10.1109/TITS.2006.874722. URL <http://ieeexplore.ieee.org/document/1637673/>.
- W. Hu, Z. Fu, and T. Tan. Similarity based vehicle trajectory clustering and anomaly detection. In *Proceedings - International Conference on Image Processing, ICIP*, 2005. ISBN 0780391349. doi: 10.1109/ICIP.2005.1530127.
- J. Huang, V. Rathod, and C. Sun. Tensorflow Object Detection API, 2018. URL <https://github.com/tensorflow/models/tree/master/research/object{ }detection>.
- D. Huttenlocher, G. K. I. T. on . . ., and U. 1993. Comparing images using the Hausdorff distance. *ieeexplore.ieee.org*, 1993. URL <https://ieeexplore.ieee.org/abstract/document/232073/>.
- A. K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, jun 2010. ISSN 0167-8655. doi: 10.1016/J.PATREC.2009.09.011. URL <https://www.sciencedirect.com/science/article/pii/S0167865509002323>.
- A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, sep 1999. ISSN 03600300. doi: 10.1145/331499.331504. URL <http://portal.acm.org/citation.cfm?doid=331499.331504>.
- I. N. Junejo, O. Javed, and M. Shah. Multi feature path modeling for video surveillance. In *Proceedings - International Conference on Pattern Recognition*, volume 2, pages 716–719. IEEE, 2004. ISBN 0769521282. doi: 10.1109/ICPR.2004.1334359. URL <http://ieeexplore.ieee.org/document/1334359/>.
- E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, pages 285–289, 2000. ISBN 1581132336. doi: 10.1145/347090.347153. URL <https://dl.acm.org/citation.cfm?id=347153>.
- Z. W. Kim. Robust lane detection and tracking in challenging scenarios. In *IEEE Transactions on Intelligent Transportation Systems*, volume 9, pages 16–26, 2008. ISBN 1524-9050. doi: 10.1109/TITS.2007.908582.
- A. H. Lai and N. H. Yung. Lane detection by orientation and length discrimination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 30(4):539–548, 2000. ISSN 10834419. doi: 10.1109/3477.865171. URL <https://ieeexplore.ieee.org/abstract/document/865171/>.
- P. Lingras, C. W. J. o. I. I. Systems, and U. 2004. Interval Set Clustering of Web Users with Rough K-Means. *Springer - Journal of Intelligent Information Systems*, 23(1):5–16, 2004. URL <https://link.springer.com/article/10.1023/B:JIIS.0000029668.88665.1a>.

- D. Makris and T. Ellis. Learning semantic scene models from observing activity in visual surveillance. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(3):397–408, 2005. ISSN 10834419. doi: 10.1109/TSMCB.2005.846652. URL <https://ieeexplore.ieee.org/abstract/document/1430826/>.
- J. McCall and M. Trivedi. Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):20–37, mar 2006. ISSN 1524-9050. doi: 10.1109/TITS.2006.869595. URL <http://ieeexplore.ieee.org/document/1603550/>.
- MEC-View. Forschungsprojekt MEC-View. <http://www.mec-view.de>, 2018. Zugriff am: 22.12.2018.
- H. Meißner. Untersuchung und Evaluierung von Methoden zur Kalibrierung optischer Sensoren in Verbindung mit Lagesensoren. *Technische Universität Berlin*, 2007.
- J. Mélo, A. Naftel, A. Bernardino, and J. Santos-Victor. Detection and classification of highway lanes using vehicle motion trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 2006. ISSN 15249050. doi: 10.1109/TITS.2006.874706.
- F. Meng, G. Yuan, S. Lv, Z. Wang, and S. Xia. An overview on trajectory outlier detection, feb 2018. ISSN 15737462. URL <http://link.springer.com/10.1007/s10462-018-9619-1>.
- V. Mirge, K. Verma, and S. Gupta. Outlier Detection in Vehicle Trajectories. *International Journal of Computer Applications*, 171(8):1–6, 2017. URL <https://www.ijcaonline.org/archives/volume171/number8/mirge-2017-ijca-915139.pdf>.
- B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 312–319. IEEE, jun 2009. ISBN 978-1-4244-3992-8. doi: 10.1109/CVPR.2009.5206559. URL [http://ieeexplore.ieee.org/document/5206559/](http://ieeexplore.ieee.org/document/5206559).
- B. T. Morris and M. M. Trivedi. Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2287–2301, 2011. ISSN 01628828. doi: 10.1109/TPAMI.2011.64. URL <https://ieeexplore.ieee.org/abstract/document/5740921/>.
- A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems 14, Vols 1 and 2*, 14:849–856, 2002. ISSN <null>. doi: 10.1.1.19.8100.
- D. OpenCV. Camera Calibration and 3D Reconstruction, 2018. URL https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

- S. Patel and J. Pingel. Introduction to Deep Learning: What Are Convolutional Neural Networks? Video - MATLAB, 2017. URL <https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--148951276577.html>.
- D. Pelleg, A. M. Icmi, and undefined 2000. X-means: Extending k-means with efficient estimation of the number of clusters. *cs.uef.fi*. URL <http://cs.uef.fi/~zhao/Courses/Clustering2012/Xmeans.pdf>.
- C. Piciarelli and G. L. Foresti. On-line trajectory clustering for anomalous events detection. *Pattern Recognition Letters*, 27(15):1835–1842, 2006. ISSN 01678655. doi: 10.1016/j.patrec.2006.02.004. URL <https://www.sciencedirect.com/science/article/pii/S0167865506000432>.
- J. Ren, Y. Chen, L. Xin, and J. Shi. Lane Detection in Video-Based Intelligent Transportation Monitoring via Fast Extracting and Clustering of Vehicle Motion Trajectories. *Mathematical Problems in Engineering*, 2014. ISSN 15635147. doi: 10.1155/2014/156296.
- P.-N. Tan et al. *Introduction to data mining*. Pearson Education India, 2007.
- S. Y. Teng, K. T. Chuang, C. R. Huang, and C. C. Li. Lane detection in surveillance videos using vector-based hierarchy clustering and density verification. In *Proceedings of the 14th IAPR International Conference on Machine Vision Applications, MVA 2015*, pages 345–348. IEEE, may 2015. ISBN 9784901122153. doi: 10.1109/MVA.2015.7153201. URL <http://ieeexplore.ieee.org/document/7153201/>.
- P. Viola, M. J. V. Recognition, Pattern, . CVPR, and undefined 2001. Rapid object detection using a boosted cascade of simple features. *ieeexplore.ieee.org*, 2001. URL <https://ieeexplore.ieee.org/abstract/document/990517/>.
- M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. *Proceedings - International Conference on Data Engineering*, pages 673–684, 2002. ISSN 10844627. doi: 10.1109/ICDE.2002.994784. URL <https://ieeexplore.ieee.org/abstract/document/994784/>.
- Weiming Hu, Xuejuan Xiao, Zhouyu Fu, D. Xie, Tieniu Tan, and S. Maybank. A system for learning statistical motion patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1450–1464, sep 2006. ISSN 0162-8828. doi: 10.1109/TPAMI.2006.176. URL <http://ieeexplore.ieee.org/document/1661547/>.
- X. L. Xie and G. Beni. A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (8):841–847, 1991.