PORSCHE

Bachelorarbeit

T2_3300

Messdatenauswertung zur Konzeptionierung von lernenden Fahrzeugfunktionen

im Studiengang Elektrotechnik

in der Studienrichtung Automation

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Moritz Schüler

24.09.2017

Bearbeitungszeitraum:	19.06.2017 – 24.09.2017
Matrikelnummer, Kurs	7735199, STG-TEL14GR3
Ausbildungsfirma:	Dr. Ing. h.c. F. Porsche AG, Stuttgart
Betreuer der Ausbildungsfirma:	Timo Maise, Dipl.-Ing.
Gutachter der Dualen Hochschule:	Winfried Bantel, Dr.

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema: „Messdatenauswertung zur Konzeptionierung von lernenden Fahrzeugfunktionen“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

---------- ----------

Ort, Datum Unterschrift

PORSCHE

Sperrvermerk

Die vorliegende Studienarbeit enthält zum Teil Informationen, die nicht für die Öffentlichkeit bestimmt sind. Alle Rechte an der Studienarbeit einschließlich der Verbreitung auf elektronischen Medien liegen bei der Dr. Ing. h.c. F. Porsche AG.

Abweichend hiervon darf der Inhalt der Arbeit während einer Sperrzeit von 5 Jahren ab dem Abgabedatum mit der ausdrücklichen schriftlichen Genehmigung der Dr. Ing. h.c. F. Porsche AG an Dritte weitergegeben werden. Nach Ablauf der Sperrzeit ist diese Genehmigung nicht mehr erforderlich.

Abstract

This work deals with the further development of a framework for geobased learning vehicle functions. This framework contains a large number of Matlab scripts, which should help to check the feasibility of various ideas and concepts on the basis of measured data. In this thesis, special attention will be given to the function of an automatic suspension changeover. For this purpose, a learning algorithm is designed and its quality is evaluated on the basis of the introduced metrics. Various visualizations should help to understand and interpret the results quickly and easily.

By means of the data, the use of a geobased suspension switchover for a sufficient number of drivers was shown. The simple learning algorithm for the changeover showed a good result for some drivers, but also showed improvement potential in several places. Above all, the handling of more frequent switchovers within a crossing and the absence of forgetting geo-positions were recognized as weak points.

Keywords:

Matlab, Proof of Concept Tool, learning functions, logged data evaluation, geo positions

Kurzfassung

Diese Arbeit beschäftigt sich mit der Weiterführung eines Frameworks für geobasierte lernende Fahrzeugfunktionen. Dabei handelt es sich um eine Vielzahl an Matlab Skripten, welche helfen sollen, anhand von Messdaten verschiedene Ideen und Konzepte auf deren Umsetzbarkeit und Sinn zu überprüfen. Weiterführend wird in dieser Arbeit speziell auf die Funktion einer automatischen Fahrwerksumschaltung eingegangen. Hierfür wird ein Lern-Algorithmus entworfen und anhand der eingeführten Metriken dessen Güte bewertet. Verschiedene Visualisierungen sollen helfen die Ergebnisse schnell und einfach verstehen und interpretieren zu können.

Mithilfe der Daten konnte die Nutzung einer geobasierten Fahrwerksumschaltung für eine ausreichende Anzahl an Fahrern gezeigt werden. Der einfache Lern-Algorithmus zur Durchführung der Umschaltungen zeigte bei einigen Fahrern ein gutes Ergebnis, ließ jedoch auch an mehreren Stellen Verbesserungspotenzial erkennen. Vor allem die Handhabung von häufigeren Umschaltungen innerhalb einer Durchfahrt und das Fehlen des Abtrainierens von Geopositionen wurden als Schwachstellen erkannt.

Schlüsselwörter:

Matlab, Proof of Concept Tool, lernende Fahrzeugfunktionen, Messdatenauswertung, Geopositionen

Inhaltsverzeichnis

Abkürzungen

Abkürzung	Bedeutung
CAD	Computer Aided Design
CAN	Controller Area Network
CSV	Comma-Separated Values
GB	Gigabyte
GPS	Global Positioning System Globales Positionsbestimmungssystem
HS	Hotspot
IoT	Internet of Things Internet der Dinge
KW	Kalenderwoche
LTE	Long Term Evolution
MAT	MATLAB Format
MDF	Measurement Data Format
PC	Personal Computer
ROC	Receiver-Operating-Characteristics
SD Karte	Secure Digital Memory Card
SiL	Software in the Loop
TB	Terabyte

Abbildungs-, Tabellen- und Codeverzeichnis

Abbildungen

Tabellen

Codes

Formelgrößen und Einheiten

Formelzeichen	Einheit	Bezeichnung
f_n		Falsch negativ
f_p		Falsch positiv
$F_ß$		F-Score
P		Wahrscheinlichkeit
r_n		Richtig negativ
r_p		Richtig positiv
β		Gewichtungsfaktor Beta

1 Einleitung

„Schon seit einiger Zeit bereitet sich die Menschheit auf die nächste digitale Revolution vor, welche viele Bereiche des alltäglichen Lebens grundlegend verändern wird: Das „Internet of Things“ (IoT) [1]. In seiner Abhandlung „The Computer for the 21st Century“ sprach Mark Weiser im Jahr 1991 erstmals von diesem Zukunftsszenario. Ziel ist eine dichte Vernetzung von Geräten zu erreichen, die sowohl untereinander, als auch mit den Menschen kommunizieren und das Leben der Benutzer mit Hilfe individuell angepasster Entscheidungen vereinfachen [2].“

Diese neue Welle an Automatisierung hat nun auch die Automobilbranche erreicht. Aufgrund neuer technologischer Fortschritte träumt man hier von der Verwirklichung des autonom fahrenden Fahrzeugs. Um dies zu erreichen werden aktuell immer mehr Bereiche und Funktionen von Automobilen „intelligenter“ gestaltet. Dies zeigt sich vor allem an den zunehmenden Sicherheitsfunktionen für neue Fahrzeuge, wie Notbremsassistent, intelligenter Stauwarner und weitere. Allerdings werden nicht nur in diesem Bereich Systeme vorangetrieben, die automatisch ins Geschehen eingreifen.

Da der Kunde immer im Vordergrund steht, werden auch viele Funktionen zur Verbesserung dessen Alltags entwickelt. So gibt es auch im Bereich der Komfortfunktionen neue Ansätze bei denen sich die Systeme an den Kunden anpassen. Ein Beispiel ist die personenbezogene Sitz- und Lenkradeinstellung, die bereits beim Aufschließen des Fahrzeugs die für den Fahrer vordefinierte Fahrposition von Lenkrad und Sitz einnimmt. Ein weiteres Beispiel ist das automatische Erkennen des Fahrstils, bei dem der beliebte Fahrmodus bereits vor Antritt der Fahrt oder Streckensequenzen voreingestellt wird, um auf der einen Seite lästiges Schalten zu vermeiden und zum anderen die Aufmerksamkeit des Fahrers weg von der Bedienung zurück zum Straßenverkehr zu lenken.

2 Aufgabenstellung

Für die Entwicklung neuer intelligenter Fahrzeugfunktionen soll im Rahmen dieser Arbeit basierend auf Messdaten ein Framework weiterentwickelt werden. Dieses soll für die systematische Konzeptbestätigung von lernenden Fahrzeugfunktionen verwendet werden, die sich Regelmäßigkeiten im Verhalten des Fahrers aneignen. Das erlernte Verhalten kann im Folgenden beispielsweise zur Automatisierung von Fahrzeugbedienungen genutzt werden. In dieser Arbeit soll im Speziellen auf die Umschaltung der Fahrwerkseinstellungen eingegangen werden. Hierfür muss zunächst ein Gütemaß definiert werden, welches zur Beurteilung des Lern-Algorithmus dient. Anschließend soll ein Algorithmus zum Erlernen von Fahrwerksumschaltpositionen konzeptioniert und prototypisch umgesetzt werden. Anhand des Gütemaßes soll dann eine Verbesserung oder Verschlechterung durch Parametervariation erkennbar sowie die Schritte für eine systematische Verbesserung der lernenden Fahrzeugfunktionen aufgezeigt werden.

Zusammengefasst sollen folgende Aufgaben bearbeitet werden:

- Weiterbearbeitung eines Frameworks zur Konzeptionierung lernender Fahrzeugfunktionen
- Definition eines Gütemaßes zur Bewertung lernender Algorithmen
- Konzeptionierung und prototypische Umsetzung eines Algorithmus zum Erlernen von Fahrwerksumschaltungen
- Bewertung des Algorithmus anhand des Gütemaßes und der vorliegenden Messdaten
- Systematisches Vorgehen zum Umgang mit dem Framework zur Verbesserung von lernenden Fahrzeugfunktionen

3 Theoretische Grundlagen

In diesem Kapitel wird dargestellt, woher die Messdaten stammen und wie sie gewonnen wurden. Des Weiteren wird ein kurzer Überblick über die Vorverarbeitung der Daten sowie dem grundsätzlichen Aufbau eines Frameworks für lernende Funktionen geschaffen. Zum Schluss wird das Rapid Prototyping Konzept vorgestellt, welches eine neuartige Entwicklungsmethode in der Automobilindustrie darstellt.

3.1 Messtechnik

Die verwendeten Messdaten für die Entwicklung des Frameworks stammen von Testfahrten, die im Rahmen eines Projekts der Fachabteilung Fahrwerk zur Bestimmung einer Reibwertkarte verwendet werden. Dabei wurden 27 verschiedene Fahrzeuge jeweils wochenweise an verschiedene Mitarbeiter vergeben, die damit ihren Weg zur Arbeit zurückgelegt haben. Insgesamt wurden diese Fahrzeuge über einen Zeitraum von Anfang Oktober 2016 bis Anfang März 2017 von 166 verschiedenen Fahrern bewegt. Bei jeder neuen Datenerhebung versucht man so viele Daten wie möglich zu generieren, doch leider ist man dabei noch von der Logging Hardware limitiert. In einem Automobil gibt es zwischen 8 und 12 CAN-Bussen, bei neueren Fahrzeugen meist noch zusätzliche Flexray-Busse, sodass eine Datenmenge von ca. 2 Gigabyte (GB) pro Stunde anfällt. Inzwischen ermöglichen Datalogger zwar ein sog. „Fulllogging“, d.h. Messen und Abspeichern aller CAN-Größen eines Fahrzeugs, jedoch nur für eine kurze Zeit.

Ein Beispiel für solch einen Logger ist der M-LOG V3 von IPETRONIK, der in Abbildung 1 zu sehen ist. Dieser kann bis zu zwölf High-Speed CAN-Busse auf einer bis zu 16 GB großen Secure Digital Memory Card (SD-Karte) mitloggen.

Abbildung 1: Datalogger IPETRONIK M-LOG V3 [3]

Das würde bedeuten, dass man alle acht Stunden stehen bleiben und die SD Karte wechseln müsste. Dadurch ist dieses Konzept nur schwer umsetzbar. Eine Alternative ist das Online Logging. Hierbei wird ebenfalls wie auf klassische Art und Weise auf eine SD Karte geschrieben, allerdings werden die Daten bei ausreichender Verbindung über ein LTE Funkmodul an einen Backend Server übertragen und abgespeichert. Diese neuartigen Logger sind der erste Schritt für ein komfortables Fulllogging, allerdings ist der technologische Stand noch nicht ausgereift. Die Infrastruktur für die Onlineübertragung ist noch zu klein, sodass es oft nicht zur Übertragung kommt und die Daten auf den Loggern schlimmstenfalls überschrieben werden oder keine neuen Daten mehr geloggt werden. Durch die gezielte Auswahl weniger Daten wird diese Lösung allerdings umsetzbar.

Ein Anbieter hierfür ist Car Media Lab mit ihrem Flea 3. Dieser ist untenstehend zu sehen und bietet die Möglichkeit vier High-Speed Can-Busse zu loggen und auf einer Onlineplattform abzuspeichern.

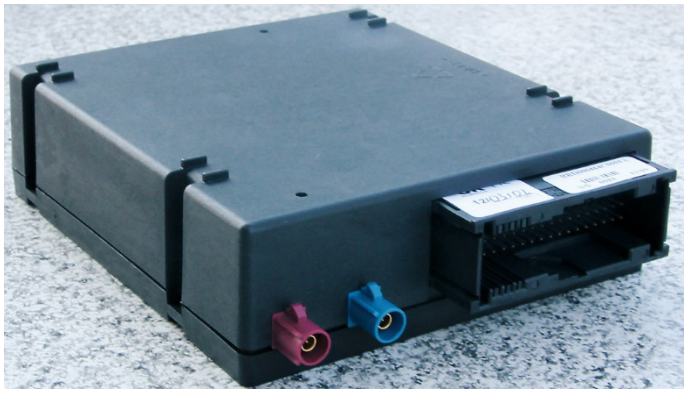

Abbildung 2: Car Media Lab Flea3 [4]

Aufgrund der hohen Datenmenge war ein Online Logging nicht umsetzbar, weshalb das erste Konzept gewählt wurde. Zusätzlich wurde die Datenmenge reduziert, sodass es genügte die Daten einmal pro Woche auszulesen und abzuspeichern. Die wichtigsten Daten wurden auf den CAN Bussen Fahrwerk, Antrieb und Dashboard gefunden, sodass diese drei CAN Busse vollständig geloggt wurden.

Da es bei dem Projekt darum geht aus den vorhandenen Daten eine Karte zu erstellen, die mit Metadaten versehen wird, ist vor allem die Geoposition eine wichtige Messgröße. Aufgrund der Ungenauigkeit des Global Positioning Systems (GPS) kann es allerdings zu Problemen kommen. Das Standardverfahren zur Verbesserung der Genauigkeit von GPS heißt Map Matching. Hierbei werden die durch Sensoren gewonnen GPS-Koordinaten eines Objekts mit den Ortsinformationen einer digitalen Karte abgeglichen. Es gibt verschiedene Verfahren, die von der Art und der Charakteristik der Sensoren (Genauigkeit, Auflösung), sowie der Güte der Karte abhängen:

- Next Neighbour Verfahren (Ermittlung des nächsten Kartenpunktes zum gemessenen Punkt)
- Vergleich der zurückgelegten Strecke zwischen Karte und Messung durch verschiedene Transformationsalgorithmen
- Karteneinpassung über Vergleich der Winkel- und/oder Krümmungsbilder

Je nach Sensor und Karte muss ein anderes Verfahren oder eine Kombination aus verschiedenen Verfahren gewählt werden [5]. Der grundsätzliche Ablauf ist jedoch bei allen ähnlich. Zunächst werden die Messdaten und die Kartendaten aufbereitet und beispielsweise die Winkelbilder abgeleitet, wie in Abbildung 3 zu sehen. Nach der Aufbereitung wird durch Zuordnung der Messung auf jede Trasse eine Übereinstimmung bestimmt. Zum Schluss ist die wahrscheinlichste Trasse abhängig der Zuordnungsgenauigkeit und der Plausibilität der geschätzten Parameter zu wählen.

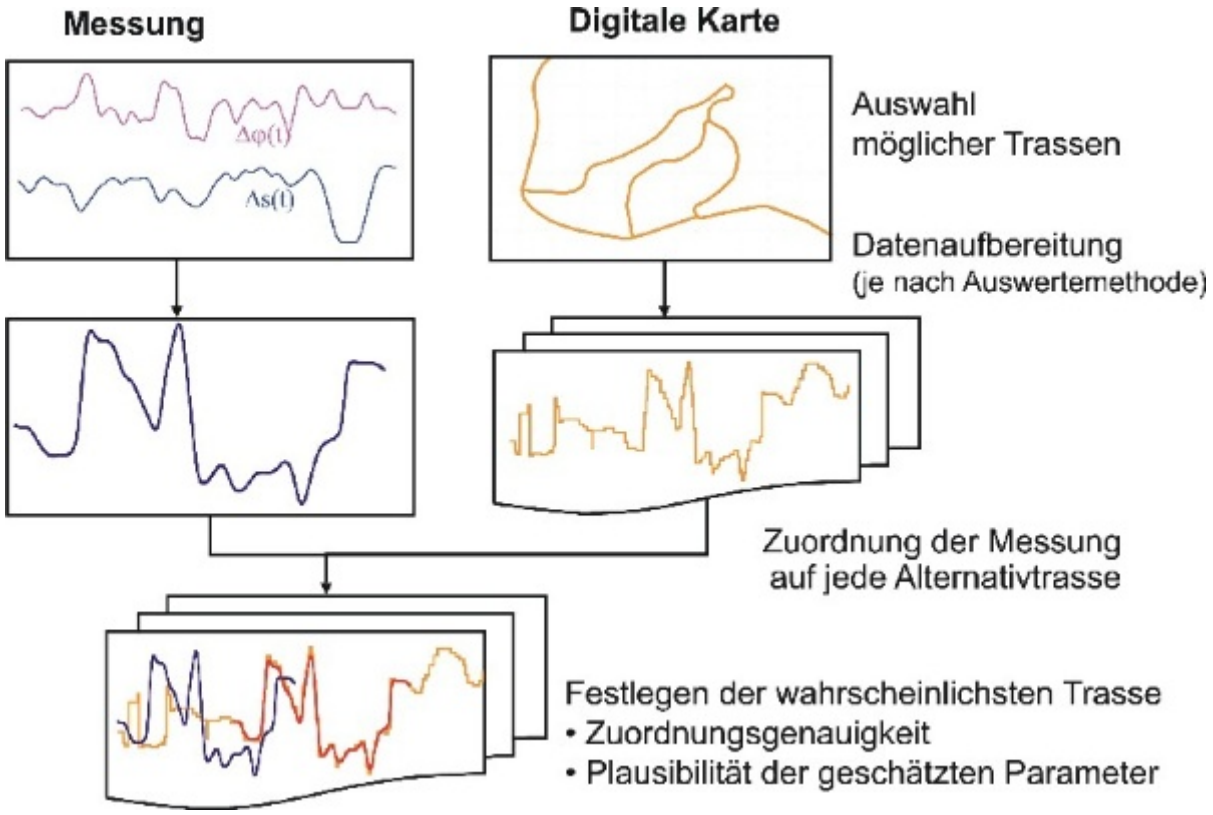

Abbildung 3: Map Matching Ablaufschema [6]

Ein herkömmliches GPS Modul eines Fahrzeugs erreicht eine Genauigkeit von 15 Metern. Durch Map Matching konnte die Position nach verschiedenen Versuchen mit Kraft- und Schienenfahrzeugen bis auf einen Meter genau bestimmt werden [7].

Ein weiteres Problem, welches bei der Messdatenverarbeitung immer auftritt, ist die Synchronisation der Abtastzeiten der Daten. Relevante notwendige Daten werden in unterschiedlichen Zeitabständen erfasst. Um jedoch einen Bezug und wertvolle Schlüsse aus den Daten ziehen zu können, müssen diese erst durch bestimmte Verfahren aufeinander angepasst werden. Zwei Verfahren, die dafür verwendet werden nennen sich „Upsampling“ und „Downsampling“. Hierbei wird im Grunde das wichtigste Signal (Key Signal) ausgewählt und alle anderen Daten werden auf die Samplezeit dieses Signals angepasst. Beim Upsampling geschieht die Abtastratenkonvertierung durch Interpolation während beim Downsampling nur jeder N-te Wert verwendet wird. Hierbei entspricht N der Abtastrate des Key Signals [8].

3.2 Aufbau des Frameworks

Nachdem erläutert wurde, wie die Daten zustande gekommen sind, soll nun der Gesamtaufbau des Frameworks übersichtlich in seinen einzelnen Schritten dargestellt werden. Diese sind die Folgenden:

1. Daten loggen
2. Daten konvertieren
3. Fahrer zuordnen
4. Daten bereinigen & anreichern
5. Lern-Algorithmus ausführen
6. Hotspots finden & vereinen
7. Durchfahrten finden
8. Gütemaß berechnen
9. Visualisieren
10. Gesamtgüte berechnen

Der erste Punkt ist im vorherigen Kapitel bereits beschrieben, weshalb hier nicht weiter darauf eingegangen wird. Die Daten liegen in einem MDF-Format vor, welches zum einen nicht direkt in Matlab nutzbar ist und zum anderen auch sehr viel Speicher benötigt. Deshalb werden die Daten im zweiten Schritt zunächst in ein für Matlab nutzbares Format gebracht, das MAT-Format. Diese Konvertierung ist sehr zeitintensiv, weshalb an dieser Stelle noch einmal genau selektiert werden muss,

welche Daten aus dem Fahrzeug benötigt werden. Da das Nutzerverhalten analysiert und gelernt werden soll, müssen die Daten, die sich bis dahin noch auf Fahrzeuge beziehen mittels einer Excel Tabelle den jeweiligen Fahrern zugeordnet werden.

Der letzte Schritt der Datenvorverarbeitung befasst sich mit dem Bereinigen und dem Anreichern der Messdaten. Das Bereinigen ist vor allem bei GPS Daten nötig, da das System beim Starten, bis es einen passenden Satelliten gefunden hat, falsche Daten sendet, welche mehrere hundert Kilometer vom eigentlichen Standort entfernt sein können. Beim Anreichern geht es darum vor der eigentlichen Datenverarbeitung zusätzliche nützliche Informationen aus den bereits vorhandenen Daten zu generieren. Ein Beispiel hierfür ist die Berechnung eines Richtungssignales zu jeder GPS Position.

Nachdem alle Daten in korrekter und passender Form vorliegen, kann die Datenverarbeitung beginnen. Als erstes wird der lernende Algorithmus ausgeführt und versucht das Fahrerverhalten vorherzusehen. Anschließend muss geprüft werden, ob der Algorithmus so arbeitet, wie geplant bzw. an welcher Stelle noch Verbesserungen getätigt werden können.

Der erste Schritt dieser Analyse befasst sich damit, die für den Algorithmus relevanten Punkte zu finden. Diese werden zukünftig als sog. Hotspots bezeichnet. Ein Hotspot wird immer an jener Geoposition angelegt, an welcher der Algorithmus seine Lernmethode ausführt. Da nun mehrere Hotspots an den gleichen Stellen liegen können, müssen diese in einem weiteren Schritt noch vereint werden. Sind alle Hotspots gefunden, muss geprüft werden, wie oft durch diese durchgefahren wurde und in welchem Zustand sich das Fahrzeug jeweils befand. Dafür wird der Schritt der Durchfahrtserkennung genutzt auf den in Kapitel 5.6 noch genauer eingegangen wird.

Um den Algorithmus unabhängig bewerten zu können, benötigt man ein quantitatives Maß zur Bestimmung der Güte. Darum wird im nächsten Schritt eine Maßzahl ausgerechnet, welche angibt, wie gut die Prädiktion in und um die Hotspots herum funktioniert. Welche Maße sich dafür eignen und sich etabliert haben, wird im folgenden Kapitel genauer vorgestellt.

Mithilfe einer Maßzahl und einem Optimierungsalgorithmus können gute Ergebnisse erzielt werden, allerdings ist das für den Anwender selbst recht undurchschaubar. Deshalb werden die Daten und Ergebnisse im nächsten Schritt auf verschiedene Weisen visualisiert, um auch für den Anwender die Resultate transparent darzustellen. Außerdem können damit einfach Verbesserungspotenziale des Lern-Algorithmus aufgezeigt werden. Genaueres zur Visualisierung ist in Kapitel 5.8 zu finden.

Zum Schluss wird die gesamte Datenverarbeitung nacheinander über alle Fahrer iteriert. Mit den erstellten Plots lassen sich jeweils die Probleme der einzelnen Fahrer sehr genau darstellen. Um einen Gesamtüberblick zu erhalten, werden sowohl alle Maßzahlen der Fahrer in einer Tabelle gegenübergestellt, als auch eine Güte über alle Fahrer hinweg berechnet. Diese Tabelle dient insbesondere dazu die Fahrer zu finden, bei denen der Lern-Algorithmus fehlschlägt.

3.3 Gütekriterien

Um die Vergleichbarkeit von Daten sicherstellen zu können, wurden sog. Gütekriterien eingeführt. Ihr Ziel ist es durch Quantifizierung von relevanten Merkmalen ein „Auswertungsverfahren im Hinblick auf mögliche Vergleiche und Abhängigkeiten zugänglich zu machen“ [9]. Ein weiterer Vorteil des Umformens in quantitative Maßzahlen ist ein Reduzieren der umfangreichen Daten auf ein wirklich wichtiges Minimum [9].

Im Falle von lernenden Algorithmen werden spezielle Gütekriterien bevorzugt. Häufig werden die Daten klassifiziert und anschließend diese Klassifikation bewertet. Da es in dieser Arbeit darum geht die Güte eines Lern-Algorithmus zu bewerten, wird nur vertieft auf diese Sonderform der Gütekriterien eingegangen.

Wie zuvor erwähnt müssen die Daten zunächst in zwei Klassen eingeordnet werden. Dabei können die Daten jeweils richtig oder falsch zugeordnet werden, wodurch vier mögliche Klassifizierungen entstehen. Um einen Klassifikator nun bewerten zu können ist es nötig, zu wissen, ob die Daten richtig oder falsch zugeordnet wurden. Zur übersichtlichen Darstellung eignet sich am besten eine Wahrheitsmatrix, wie sie in Tabelle 1 zu sehen ist.

Tabelle 1: Wahrheitsmatrix

Ermittelt/tatsächlich	Klasse 1	Klasse 2
Klasse 1	Richtig positiv (r_p)	Falsch positiv (f_p)
Klasse 2	Falsch negativ (f_n)	Richtig negativ (r_n)

Die beiden grün markierten Felder stellen richtige Zuordnungen dar, während die roten Felder jeweils für falsche Einordnung stehen. Sind alle Daten zugeordnet, wird gezählt, welcher Fall wie häufig aufgetreten ist. Mit diesen Werten lassen sich nun verschiedene Kenngrößen zur Beurteilung des Klassifikators berechnen, die nachfolgend nach den Definitionen von Gigerenzer dargestellt werden [10].

Als Erstes sei die Sensitivität oder auch Trefferquote (engl. recall) erwähnt. Sie gibt an, wie viele der tatsächlich positiven Objekte auch als solche identifiziert wurden.

$$P(recall) = \frac{r_p}{r_p + f_n} \qquad (1)$$

Analog dazu lässt sich die Falsch-Negativ-Rate (miss rate) darstellen. Sie gibt an wie viele Objekte fälschlicherweise als negativ klassifiziert wurden, obwohl sie in Wirklichkeit positiv sind. Sie kann entweder durch subtrahieren der Trefferquote von Eins oder durch nachfolgende Formel berechnet werden.

$$P(miss\ rate) = 1 - P(recall) = \frac{f_n}{r_p + f_n} \qquad (2)$$

Da die oberen Kenngrößen nur Aussagen darüber treffen, wie viele richtige Ergebnisse gefunden, jedoch nicht wie viele falsche Zuordnungen dabei mitberücksichtigt wurden, gibt es mit den nachfolgenden Kriterien eine Möglichkeit dies darzulegen.

Mit der Relevanz oder Genauigkeit (engl. precision) lässt sich ausdrücken, wie viele der insgesamt als positiv klassifizierten Objekte auch wirklich positiv sind [11]. Dies lässt sich durch folgende einfache Formel besser verstehen.

$$P(precision) = \frac{r_p}{r_p + f_p} \qquad (3)$$

Analog zur Relevanz lässt sich mit der Segreganz oder Trennfähigkeit der Anteil der korrekt negativ zugeordneten Ergebnisse aus allen negativ zugeordneten Werten darstellen. Dies lässt sich mit der Formel ebenfalls leichter nachvollziehen.

$$P(Segreganz) = \frac{r_n}{r_n + f_n} \qquad (4)$$

Im Gegensatz zu den beiden erst genannten Gütemaßen lassen sich Genauigkeit und Segreganz nicht zu 1 addieren, da sie jeweils von unterschiedlichen Fällen ausgehen.

Ein weiteres Gütemaß, welches oft verwendet wird, ist die Vertrauens-wahrscheinlichkeit oder Treffergenauigkeit (engl. accuracy). Sie gibt aus allen Objekten diejenigen an, die korrekt klassifiziert werden. Berechnet wird sie nach nachfolgender Formel:

$$P(true\ accuracy) = \frac{r_p + r_n}{r_p + f_p + r_n + f_n} \qquad (5)$$

Neben den richtig zugeordneten Elementen werden häufig auch einige Objekte falsch zugeordnet. Für diese Fehleinschätzungen lässt sich ebenfalls die Wahrscheinlichkeit berechnen. Die Falschklassifikationsrate lässt sich hierfür wie folgt beschreiben:

$$P(false\ accuracy) = \frac{f_p + f_n}{r_p + f_p + r_n + f_n} \qquad (6)$$

Da es nicht möglich ist alle Gütemaße gleichzeitig zu optimieren, hat sich eine kleine Auswahl der obigen etabliert: Precision und Recall. Je nach Anwendungsfall hat jedoch nur eines der beiden Vorrang, deshalb wurden zusätzlich kombinierte Maße entwickelt, wie z.B. der F-Score (F_β) [12].

Er verbindet die beiden Gütemaße und lässt über einen Parameter β eine Gewichtung zwischen Precision und Recall über folgende Formel zu [13]:

$$F_\beta = (1 + \beta^2) \times \frac{(precision \times recall)}{(\beta^2 \times precision + recall)} \qquad (7)$$

Bei einem Wert von β = 1 werden beide Maße gleichermaßen bewertet. Erhöht man den Faktor allerdings, wird die Trefferquote über die Genauigkeit gestellt. Im Umkehrschluss bedeutet ein Wert von β < 1 eine höhere Gewichtung des Precision-Wertes.

3.4 Rapid Prototyping

Durch die Globalisierung, der großen Anzahl an Start-Ups und etablierten Unternehmen herrscht große Konkurrenz auf dem Weltmarkt. Diese fordert die Teilnehmer immer wieder dazu auf die Produkte zu verbessern. In der letzten Zeit führte der Konkurrenzkampf vor allem auch dazu, die Entwicklungen zu beschleunigen [14]. Gerade in der Softwarebranche werden durch viele amerikanische Start-Ups die Produktentwicklungen beschleunigt, aber auch einige inzwischen etablierten Größen, wie Tesla, treiben den Weltmarkt voran. Dies führt dazu, dass gerade gestandene Unternehmen ihre Entwicklungsstrukturen überdenken und versuchen dem schnellen Trend zu folgen. Ein Aspekt mit dem die Entwicklung beschleunigt werden kann, ist das Verfahren des „Rapid Prototyping“.

Der Begriff „Rapid Prototyping“ beschreibt, wie die Übersetzung ins Deutsche zeigt, eine Art schnellen Modellbaus. Damit ist eine Vorgehensweise gemeint, die es Unternehmen ermöglicht bereits in der Projektplanungsphase erste Testmodelle zu erstellen. Mithilfe dieser „Prototypen“ können schon frühzeitig Probleme und Schwächen aufgedeckt und behoben werden. Dies erspart den Unternehmen vor allem Zeit und Kosten [15].

Unter Rapid Prototyping versteht man jedoch viele Möglichkeiten und Vorgehensweisen einer schnellen Modellbildung. Im Allgemeinen lässt es sich als eine Spezialform des Prototypings, welches aus der Fertigungstechnik stammt, beschreiben. Es dient als Begriff für die Beschreibung einer Vorgehensweise bei welcher mithilfe von digitalen Modellen automatisiert produziert werden kann. Das

gängigste Beispiel für Prototyping sind 3D Drucker. Hierbei werden von der Maschine Computer-Aided-Design-Daten (CAD-Daten) eingelesen und alle weiteren Schritte und Daten aus dem Modell abgeleitet, sodass die Produktion des Teils automatisiert ablaufen kann [14].

Die Anwendungsgebiete sind wie oben erwähnt weitreichend. So erstreckt sich der Begriff nicht nur über die Produktion, sondern auch im Bereich der Softwareentwicklung lässt sich mit dem neuartigen Verfahren viel Entwicklungsaufwand und Geld einsparen. Es bietet die Möglichkeit „neue Funktionen schnell evaluieren und überarbeiten zu können“ und führt so zu einem „effizienten Softwaredesign“ [16]. Die ersten Prototypen können zunächst auf dem PC simuliert (Virtual Prototyping) und am Ende auf dem Zielsteuergerät ausgeführt werden, sodass Ergebnisse aus der Vorentwicklung direkt in die Serienreife einfließen können.

Für die Automobilindustrie bedeutet dies, dass bereits erste Bestandteile und Funktionalitäten mit Vorserienfahrzeugen auf der Zielhardware selbst getestet werden können. Dabei werden wiederum Messdaten gewonnen, die das Modell auf dem PC anreichern und Fehler und Probleme in der aktuellen Software wiederspiegeln. Diese Arbeitsweise, die eine frühzeitige Einbindung neuer Messdaten erlaubt, ermöglicht ein schnelles und verbessertes Fehlermanagement. Viele einfache und kleine Fehler, die eventuell nicht bedacht wurden, können schnell behoben werden, sodass frühzeitig ein lauffähiger Prototyp entsteht. Da in einer frühen Phase bereits erste Langzeittests auf der Zielhardware durchgeführt werden können, lässt sich der Entwicklungsprozess deutlich beschleunigen.

Die eben beschriebene Vorgehensweise des Rapid Prototyping in der Softwaretechnik stellt nur den kleinen Bereich des explorativen Prototypings dar. Hierbei soll gezeigt werden, ob Ideen umgesetzt werden können und ob Spezifikationen vollständig und richtig sind [17]. Es gibt aber noch weitere Arten des Rapid Prototypings in der Softwareentwicklung, welche kurz erläutert werden [17].

Beim evolutionären Prototyping wird zunächst ein Programm mit den Grundfunktionalitäten aufgebaut, welches Stück für Stück erweitert wird. Somit ist das Architekturkonzept zu keiner Zeit festgelegt, wodurch immer wieder neue Anforderungen aufgenommen werden können. Es wird versucht den Prototypen stets

lauffähig zu halten, sodass er bis zur Produktreife erweitert werden kann. Demnach ist das Ergebnis des evolutionären Prototyps auch als das fertige Endprodukt zu sehen. Der Ansatz dieses Konzepts ist schon mehr als 50 Jahre alt, jedoch gibt es noch heute Probleme dabei ihn umzusetzen. Oft sind viele Programmteile miteinander verknüpft, sodass bei der Aufnahme einer neuen Anforderung große Programmteile angepasst werden müssen.

Eine weitere Idee eines Prototyps ist der Ansatz des experimentellen Prototypings, auch „throw-away“-Ansatz. Hierfür wird ein erster lauffähiger Programmteil entworfen, welcher zur ersten Analyse des Problems dienen soll. Das Programm selbst wird im Nachhinein verworfen, die gewonnen Erkenntnisse jedoch fließen im Aufbau des neuen Programms mit ein.

Ein weitverbreiteter Anwendungsfall stellt der inkrementelle Ansatz dar. Ähnlich zum evolutionären Ansatz wird hier zunächst ein Programm mit den Grundfunktionalitäten aufgebaut. Der Unterschied ist jedoch, dass es sich hier um einen stabilen Programmkern handelt, welcher später nicht mehr verändert werden soll. Fehlende Programmteile, die eingeplant sind, werden bis zur Implementierung durch Simulationen umgesetzt. Deshalb lässt sich das Programm in zwei Teile aufgliedern, den simulierten und den implementierten Teil. Dieser Ansatz stellt eine Verbesserung des evolutionären Ansatzes dar, da ein fixer lauffähiger Kern besteht. Allerdings ist dem nicht immer so, denn häufig muss das Gesamtsystem (auch der Kern) bei Hinzufügen eines neuen Programmteils stark verändert werden, was Kosten und Zeit in Anspruch nimmt.

Die letzten beiden Arten von prototypischen Softwareumsetzungen unterscheiden sich von den vorherigen. Beim horizontalen und vertikalen Prototyping wird das Softwareprojekt in Ebenen unterteilt. Wird nun eine einzelne Ebene sehr genau aufgebaut und vorgestellt, spricht man vom horizontalen oder Demonstrationsprototyping. Dies wird oft verwendet um zum Beispiel Homepages vorzustellen. So können das User Interface und die einzelnen Möglichkeiten bereits existieren, jedoch ohne Funktion der Schaltflächen auf der Seite.

Das vertikale Prototyping dient eher dem Zweck einen funktional vollumfänglichen Teil des Projekts durch alle Ebenen hindurch zu entwickeln. Bei offenen Funktionalitätsfragen kann so durch bereits umgesetzte Softwarefunktionen eine Entscheidung erleichtert werden.

3.5 V-Modell

Trotz der verschiedenen Anwendungsfälle des Rapid Prototypings in der Softwaretechnik haben alle eins gemeinsam. Sie lassen sich in das Vorgehen des in der Entwicklung beliebten V-Modells einbinden, welches nachfolgend in seinen Phasen beschrieben wird.

Am Anfang eines Entwicklungsprozesses wird ein Modell erstellt, in dem wichtige Zusammenhänge dargestellt werden. Heutzutage werden dafür oft Simulationen verwendet, welche ersten Tests unterzogen werden. Gibt es allerdings nicht die Möglichkeit Simulationsmodelle zu verwenden, können theoretische Modelle erstellt und getestet werden. Nach erfolgreichem Abschluss dieser Versuche erfolgt eine Übertragung auf leistungsfähige Simulationshardware. Das bedeutet, dass das reale System mit dem theoretischen Simulationsalgorithmus betrieben werden kann. Dieses Schema nennt sich auch Software-in-the-Loop (SiL) und wird bei verschiedenen Vorgehensmodellen in der Softwareentwicklung angewandt [18].

Ein häufig verwendetes Vorgehensmodell ist das V-Modell, welches als Erweiterung für das Wasserfallmodell zu sehen ist. Es bietet viel Flexibilität und kann bei Softwareprojekten jeder Größe angewendet werden. Den Namen hat das Modell von seinem V-förmigen Aufbau der Prozessschritte. Dieser ist in Abbildung 4 zu sehen.

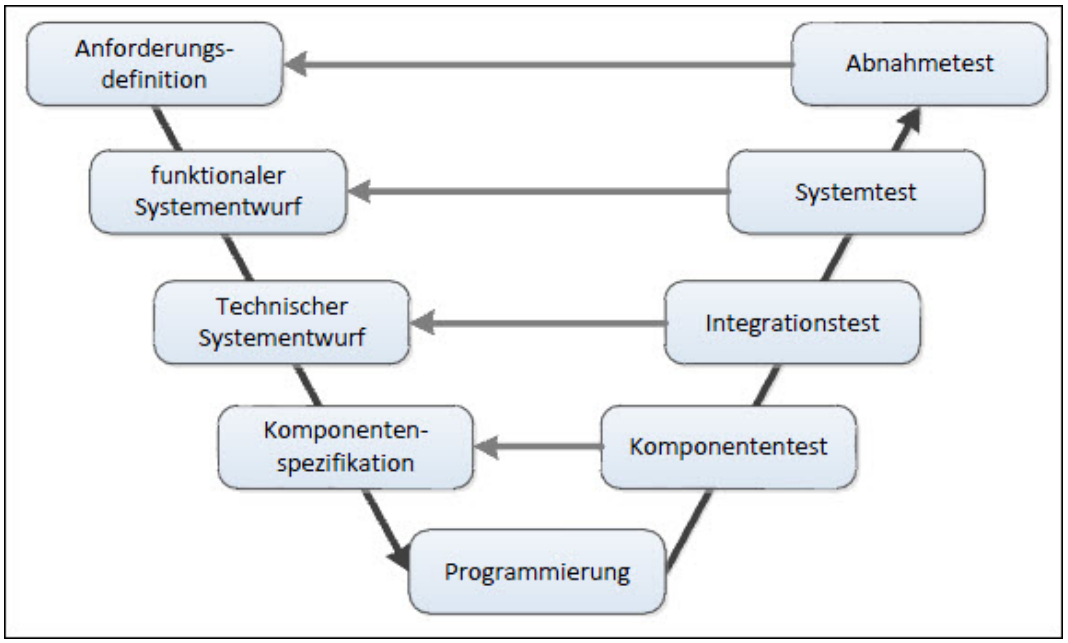

Abbildung 4: Schematische Darstellung des V-Modells [19]

Grundprinzip des V-Modells ist die Beziehung zwischen den Anforderungen und dem Testen [20]. Die Basis des Modells bilden die Anforderungen und deren Umsetzung, welche sich in der Akzeptanz der Nutzer widerspiegelt. Es untergliedert sich in einzelne Stationen, wodurch der Fokus des Entwicklungsprozesses auf spezielle Teile der einzelnen Stufen gelegt werden kann. Falls Anforderungslücken oder Fehler auftreten, bietet dieses Modell die Möglichkeit schnell den geeigneten Ansatzpunkt für Änderungen zu finden [20].

Zu Beginn eines Projekts müssen die Anforderungen definiert werden, um eine Basis für die Bedürfnisse der Anwender zu schaffen. Durchgeführt wird dies vom Bauteilverantwortlichen, dem Funktionsverantwortlichen und dem Applikations-ingenieur.

An nächster Stelle folgt der funktionale Systementwurf. Er legt fest, welche Funktionen die Software beinhalten und wie sie sich in der Interaktion mit dem Nutzer verhalten soll. Beim technischen Systementwurf findet eine Bewertung statt, welche sich in eine Kosten-Nutzen-Analyse sowie der Priorisierung der Funktionen untergliedert. Anschließend werden die einzelnen Komponenten spezifiziert. Ab hier zeigt sich der Vorteil des V-Modells, da einige Komponententests bereits parallel zur Komponentenspezifikation bzw. –implementierung erfolgen können [21].

Die unterste Stufe bildet die Programmierung und Modulimplementation. Dazugehörig sind Modultests, welche die kleinsten Einheiten der Software, sogenannte Programmmodule, überprüfen. Diese werden zu Komponenten zusammengefasst und wiederum auf Fehlerfreiheit getestet. Anschließend integriert man die einzelnen Komponenten und verknüpft sie miteinander, um im Integrationstest das Zusammenspiel zu überprüfen [21]. Der nächste Schritt ist der Systemtest, welcher die Software auf Funktionalität und logische Abfolge überprüft. Das bedeutet, dass die Software nicht nur funktioniert, also frei von Syntaxfehlern o.ä. ist, sondern auch keine Logikfehler mehr enthält. Zum Schluss werden die fachlichen Anforderungen mit den Ergebnissen aus den Abnahmetests verglichen, um weitere Anforderungen aufzustellen, bestehende abzuändern oder die Software freizugeben [21].

Zusammenfassend lässt sich sagen, dass das V-Modell viel Wert auf das Testen der Software legt. Die Stärke dieses Modells liegt zudem in der portionsweisen Entwicklung der Software. So können einzelne Komponenten schon frühzeitig getestet und korrigiert werden oder ganze Funktionen aufgeschoben und bei Bedarf nach Ende der erfolgreichen Abschlusstests mit einem erneuten Durchlaufen des Modells hinzugefügt werden.

4 Aktueller Stand des Frameworks

In diesem Kapitel soll mit Bezug auf die in 3.2 angesprochenen Punkte der aktuelle Stand vor Beginn dieser Arbeit dargestellt werden. Es wird beschrieben, welche Teilbereiche schon umgesetzt sind, welche noch fehlen und wo Probleme aufgetreten sind, sodass Überarbeitungen erforderlich werden.

Zu Beginn der Arbeit waren alle Daten des Projekts der Fahrwerksabteilung bereits geloggt und vorhanden. Allerdings wurden nur einige wenige Daten zu Testzwecken des Programms konvertiert. Da der Fokus zunächst auf der Erstellung eines funktionalen Frameworks bestand, wurden die Daten auch noch nicht auf die jeweiligen Fahrer bezogen. Dazu war es nötig, die Daten zu bereinigen und anzureichern, um die ersten Funktionsprototypen testen zu können.

Zu den vorhandenen Implementierungen zählten das Finden und Vereinen von Hotspots, sowie einzelne Visualisierungen. Auch wurde ein Algorithmus vorbereitet, der die Anzahl an Durchfahrten durch einen Hotspot zählen konnte.

Offene Punkte vor dem Beginn der Arbeit waren demnach das Konvertieren der Daten und das anschließende Zuordnen zu den Fahrern. Außerdem musste ein Lern-Algorithmus geschrieben und die Art der Hotspotfindung erweitert werden. Weitere Optimierungen mussten an der Durchfahrtserkennung vorgenommen werden, die neben der Anzahl der Durchfahrten vor allem auch die einzelnen Koordinaten und Zusatzinformationen extrahieren sollte. Die Auswahl und Berechnung eines Maßes für die Güte des zu schreibenden Lern-Algorithmus fehlte ebenso wie die Visualisierung dessen.

5 Durchführung

In diesem Kapitel wird der Umgang mit den Funktionen der Datenvorverarbeitung, sowie die technische Umsetzung des Frameworks vorgestellt. Es wird ein Lern-Algorithmus für die automatische Fahrwerksumschaltung gezeigt und die nötigen Schritte zur anschließenden Analyse beschrieben.

5.1 Daten konvertieren

Nachdem alle Messdaten von der Fahrwerksfachabteilung zur Verfügung gestellt wurden, mussten diese konvertiert werden. Bei den beiden Vorgängerarbeiten, die sich mit dem Framework beschäftigten, lag der Fokus zunächst auf einem Framework mit Basisfunktionen, die mit einigen Daten getestet und validiert wurden. Im Rahmen dieser Arbeit sollte das Framework mit allen vorhandenen Daten bespielt werden, um letzte kleine Bugs ausfindig zu machen und zu beheben.

Durch die erneute Konvertierung ergab sich die Möglichkeit die Signale zur Umsetzung der Funktion noch einmal zu überprüfen und falls nötig weitere Signale hinzuzufügen und ebenfalls zu konvertieren. Hierbei wurde entschieden, eine größere Menge an Signalen hinzuzufügen, um den langwierigen Schritt der Datenumwandlung nicht erneut durchführen zu müssen, wenn weitere geobasierte lernende Funktionen entwickelt werden. Dafür wurde über verschiedene weitere Funktionen nachgedacht, welche in Zukunft umgesetzt werden können. Eine Idee, welche bereits bearbeitet wird, ist die prädiktive Vorkonditionierung. Dort geht es darum häufige Abfahrtszeiten und –orte ausfindig zu machen und deren Verhalten zu analysieren, um zum Beispiel morgens vor Abfahrt zur Arbeitsstätte die Sitzheizung voreinstellen zu können. Für die gemeinsame Signalauswahl werden sog. CAN-Matrizen genutzt. Diese stellen in Form einer Excel-Tabelle alle Signale mit ihren jeweiligen Eigenschaften, wie in Tabelle 2 zu sehen, dar.

Tabelle 2: Ausschnitt CAN Matrix

	A	B	C	D	E	F	K	L	M	N	O	S	V	W	X
1	Botschaften						Signale								
2													Physikalische Werte		
3	Botschaft	Identifier [hex]	Identifier [dez]	Botschaftslänge	Zykluszeit normal [ms]	Zykluszeit schnell [ms]	Signal	StartByte	StartBit	Signal Länge	Signalsendeart	InitWert roh [dez]	Min Rohwert [dez]	Max Rohwert [dez]	phy Werte [dez]
5	ACC_05	0x10D	269	8	20		ACC_05_CHK	1	0	8	Cyclic		0	255	0 .. 255
6	ACC_05	0x10D	269	8	20		ACC_05_BZ	2	0	4	Cyclic		0	15	0 .. 15
7	ACC_05	0x10D	269	8	20		ACC_Freigabe_Momentenanf	2	4	1	Cyclic	0			
8	ACC_05	0x10D	269	8	20		ACC_Freigabe_Verzanf	2	5	1	Cyclic	0			
9	ACC_05	0x10D	269	8	20		ACC_Getriebestellung_P	2	6	1	Cyclic	0			
10	ACC_05	0x10D	269	8	20		ACC_limitierte_Anfahrdyn	2	7	1	Cyclic	0			
11	ACC_05	0x10D	269	8	20		ACC_Momentenanforderung	3	0	10	Cyclic	0	0	1021	0 .. 1021
12	ACC_05	0x10D	269	8	20		ACC_zul_Regelabw	4	2	6	Cyclic	0	0	63	0 .. 1.512
13	ACC_05	0x10D	269	8	20		ACC_Verz_anf	5	0	11	Cyclic	1444	0	2047	-7.22 .. 3.015

Zum einfachen Umgang mit dem Konverter wurde deshalb beschlossen eine CSV-Tabelle mit den notwendigen Daten einzulesen. Damit reduziert sich der Aufwand darauf die nötigen Signal- und Botschaftsnamen herauszusuchen und in einer CSV-Datei abzuspeichern. In dieser Arbeit wurden die in der nachfolgenden Tabelle dargestellten Signale ausgewählt.

Tabelle 3: CSV der PIDM Signale

Mandatory	Optional
CAN1__ESP_01__ESP_v_Signal	CAN1__Airbag_02__AB_Gurtschloss_FA
CAN1__ESP_01__ESP_QBit_v_Signal	CAN1__Airbag_02__AB_Gurtschloss_BF
CAN1__ESP_02__ESP_QBit_Gierrate	CAN1__Airbag_02__AB_Gurtschloss_Reihe2_FA
CAN1__ESP_02__ESP_QBit_Laengsbeschl	CAN1__Airbag_02__AB_Gurtschloss_Reihe2_MI
CAN1__ESP_02__ESP_QBit_Querb	CAN1__Airbag_02__AB_Gurtschloss_Reihe2_BF
CAN1__ESP_02__ESP_Querbeschleunigung	CAN1__Airbag_02__AB_Gurtschloss_Reihe3_FA
CAN1__ESP_02__ESP_Laengsbeschl	CAN1__Airbag_02__AB_Gurtschloss_Reihe3_MI
CAN1__ESP_02__ESP_Gierrate	CAN1__Airbag_02__AB_Gurtschloss_Reihe3_BF
CAN1__ESP_02__ESP_VZ_Gierrate	CAN1__Gateway_31_PAG__VS_VD_offen_ver
CAN3__PCM_04_PAG__GPS_N_Grad	CAN3__Klima_03__KL_AussenTemp_WK
CAN3__PCM_04_PAG__GPS_E_Grad	CAN3__Klima_03__KL_Sonnenintensitaet
CAN3__Charisma_01__CHA_Fahrer_Umschaltung	CAN3__Klima_08_PAG__KL_Sportmodus
CAN3__Charisma_01__CHA_Ziel_FahrPr_MO	CAN3__LWI_01__LWI_Lenkradwinkel
CAN3__Charisma_02__MO_Charisma_FahrPr	CAN3__LWI_01__LWI_VZ_Lenkradwinkel
CAN4__ESP_11__ESP_Charisma_FahrPr	CAN3__LWI_01__LWI_Lenkradw_Geschw
CAN4__ESP_11__ESP_Charisma_Umschaltung	CAN3__LWI_01__LWI_VZ_Lenkradw_Geschw
	CAN4__Motor_10__MO_Fahrpedalgradient
	CAN4__Motor_01__MO_Kickdown
	CAN4__ESP_12_PAG__ESP_FZR_uebersteuern
	CAN4__ESP_12_PAG__ESP_FZR_untersteuern
	CAN3__PCM_CDR_01_PAG__PCM_Std
	CAN3__PCM_CDR_01_PAG__PCM_Min
	CAN3__PCM_CDR_01_PAG__PCM_Sek

Die Signale können, wie in Tabelle 3 zu sehen, in Mandatory und Optional eingeteilt werden. Das bezieht sich darauf, dass je nach Ausstattungsvariante des Fahrzeugs andere CAN-Nachrichten existieren. So gibt es zum Beispiel den Porsche 911 mit oder ohne Handschaltgetriebe. Durch die Einteilung lassen sich Signale definieren, die zwingend vorhanden sein müssen, wie die GPS-Position, damit das Datenfile konvertiert wird (Mandatory) und andere Signale (Optional), die falls sie vorhanden sind konvertiert und andernfalls nicht berücksichtigt werden.

Bei den vorherigen Arbeiten wurde für die ersten Datenumwandlungen nach dem Prinzip des experimentellen Prototypings ein Konverter entwickelt, der die Möglichkeit bot einige Daten umzuwandeln, welche weiterverarbeitet werden konnten. Allerdings stieß dieser bei einer hohen Datenmenge schnell an seine Grenze. Mit der Datenmenge von mehr als 1TB ergaben sich sowohl Probleme bei der Zuverlässigkeit als auch der Geschwindigkeit. Für den Entwurf des neuen Codes wurde also der alte Prototyp genutzt und dessen Schwachstellen präventiv berücksichtigt.

Zur Steigerung der Zuverlässigkeit und Reduzierung der Laufzeit wurden verschiedene Abbruchbedingungen definiert. Durch diese wird eine fehlerhafte oder falsche Datei aussortiert und ein Programmabsturz vermieden. Die Abbruchbedingungen wurden in einer, für die Laufzeit optimalen, Abfolge angeordnet, sodass falsche Dateien schnellstmöglich verworfen werden. Im Nachfolgenden wird der Konvertiervorgang anhand des Codes noch einmal genauer vorgestellt. Der Funktionsaufruf gibt ersten Aufschluss über die Handhabung:

res=c.convert(c,load_folder_path,save_folder_path,tool_folder_path,keyChanName);

Zum Konvertieren benötigt man einen Ursprungspfad mit Dateien, die umgewandelt werden sollen, einen Zielpfad, in welchen die konvertierten Daten gespeichert werden und einen Toolpfad, der alle Codeteile und Funktionen beinhaltet, die verwendet werden. Als Rückgabewert erhält man eine Zahl, die Auskunft über das Gelingen oder Scheitern der Konvertierung gibt. Der zusätzliche Übergabeparameter „c“ stellt ein Objekt dar, das je nach Anwendungsfall unterschiedliche Parameter enthält, welche die Konvertierung steuern. In Listing 1 ist der Konstruktor zu sehen, welcher Aufschluss gibt, um welche Parameter es sich hierbei handelt:

```
function c=Converter(channels, channels_mandatory,minValueCount,
keyChannelName)
    c.reader=PD5PIDMReader();
    c.downsampler=MeanMerger();
    c.upsampler=UpSampler();
    c.channels=channels;
    c.channels_mandatory = channels_mandatory;
    c.minValueCount=minValueCount;
    c.keyname=keyChannelName;
    c.start = 1;
end
```

Listing 1: Konstruktor des Konverters

Im Konstruktor, welcher vor Start der Umwandlung aufgerufen werden muss, werden die oben beschriebenen Signale in der Reihenfolge optionale, dann wichtige Signale übergeben. Des Weiteren wird ein Schwellwert an Datenpunkten übergeben, um zu entscheiden, ob es sich lohnt die Datei zu konvertieren. Als letzter Parameter wird ein Signal angegeben, welches als Schlüsselsignal dient. Das bedeutet, dass während des Konvertierungsvorgangs alle anderen Signale auf die Auflösung des Hauptsignals, wie in Kapitel 3.1 beschrieben, angepasst werden. In Listing 2 ist die Struktur mit den Abbruchbedingungen als Code dargestellt.

```
function res=convert(c,load_folder_path,save_folder_path,tool_folder_path)
% Abspeichern der Dateien in der Variable listing
cd(load_folder_path)
listing=dir;
res=0;

% Iterieren über alle Dateien in dem Ordner
for i=3:length(listing)
    names='';

    % Laden des i-ten Files
    p=listing(i).name;
    display(['Load File: ' p]);

    % Prüfen ob File schon konvertiert wurde
    cd(save_folder_path);
    if (exist([save_folder_path '\' p '.mat'], 'file') == 0)
        % Wenn mandatory Channels Ok, dann starten mit einlesen
        % des Key Channels
        if isempty(names)==0
            display('Mandatory Channels OK')
            % Prüfen ob file genug Daten enthält
            if dfl{keyn}.length>c.minValueCount
                display(dfl{keyn}.length);
```

```
                % Konvertiervorgang
                res=1;
            else
                display(dfl{keyn}.length);
                display('Key Channel does not have enough data points.');
                res=2;
            end
        else
            display('No Mandatory Channels found')
            res=3;
        end
    else
        display('File is already converted');
        res=4;
    end
end
end
```

Listing 2: Codeausschnitt des Konverters

Anhand der Formatierung lassen sich die vier Abbruchbedingungen schnell ablesen. Zuerst wird überprüft, ob der Ursprungsordner überhaupt Dateien zum Konvertieren enthält. Als nächstes wird verhindert, dass eine bereits umgewandelte Datei noch einmal konvertiert wird. Die beiden weiteren Bedingungen prüfen die Vollständigkeit der wichtigen Signale sowie das Vorhandensein genügender Datenpunkte. Jede Abbruchbedingung hat ihren eigenen Rückgabewert, sodass hierdurch schnell analysiert werden kann, wo das Problem liegt ohne den Code kennen zu müssen. Dafür benötigt der Anwender nur die nachfolgende Tabelle:

Tabelle 4: Abbruchbedingung und Rückgabewert des Konverters

Abbruchbedingung	Rückgabewert
Datei bereits konvertiert	4
Nicht alle wichtigen Signale enthalten	3
Zu wenig Datenpunkte	2
Datei wurde erfolgreich konvertiert	1
Keine Dateien in Ursprungspfad	0

Trifft keine der Abbruchbedingungen zu, wird der Konvertierungsvorgang gestartet und nach dessen Abschluss eine Eins zurückgegeben. Welche Schritte jede umgewandelte Datei durchlaufen hat, ist im nachfolgenden Listing zu sehen.

```
if dfl{keyn}.length>c.minValueCount
    display(dfl{keyn}.length);

    % Einlesen aller Channels
    dfl=c.reader.readIn([load_folder_path '\' p],c.channels);

    display('Converting File...');
    display('Up Sampling...');
    ndfl=c.upsampler.upsample(c.keyname,dfl);
    display('Down Sampling...');
    data=c.downsampler.merge(ndfl);
    save([save_folder_path '\' p '.mat'],'data');
    display('saved...');
    res=1;
else
    display(dfl{keyn}.length);
    display('Key Channel does not have enough data points.');
    display('Proceeding with next file.');
    res=2;
end
```

Listing 3: Codeausschnitt des Konvertierungsvorgangs

Zuerst werden die Daten eingelesen bevor sie durch „Upsampling“ und „Downsampling“ an die Auflösung des Schlüsselsignals angepasst werden. Im letzten Schritt wird die Datei unter dem alten Namen im neuen MAT-Format im Zielordner abgespeichert. Die Daten sind zwar umgewandelt, liegen aber dennoch in der falschen Struktur vor. Um die Daten sowohl für diese Arbeit als auch die prädiktive Vorkonditionierung vernünftig nutzen zu können, mussten sie, wie im nachfolgenden Kapitel beschrieben, nach Fahrern statt Fahrzeugen sortiert werden. Der gesamte Code des Konverters ist im Anhang A1 dargestellt.

5.2 Fahrermapping

Zum Sortieren der Daten nach Fahrern ist es nötig zu wissen, welcher Fahrer, wann und wie lange, welches Fahrzeug gefahren ist. Diese Informationen wurden mittels einer Excel-Tabelle, welche in dieser Form in mehreren Projekten angewandt wird, zur Verfügung gestellt. Ein Ausschnitt der Tabelle ist unten dargestellt.

Tabelle 5: Ausschnitt aus Datei der Fahrzeugzuordnung

KW	39	40	41	42	43	44	45
Fahrer 1	2	2	18				
Fahrer 2		1	2		9	20	20
Fahrer 3	3	3			11	1	1
Fahrer 4		1	3		1	23	
Fahrer 5	5	5	5	5	5	5	5
Fahrer 6		1	1	3		1	
Fahrer 7	1	1			3	1	1
Fahrer 8					22	3	
Fahrer 9		1		10	1	1	
Fahrer 10		4					
Fahrer 11	1		4				
Fahrer 12				1	4		
Fahrer 13							4
Fahrer 14		1	1	1	1	1	
Fahrer 15							
Fahrer 16			1	6			
Fahrer 17					6	1	
Fahrer 18						6	
Fahrer 19	1	7	1				
Fahrer 20		1	7				
Fahrer 21				7	7		
Fahrer 22						7	
Fahrer 23							7
Fahrer 24							
Fahrer 25							
Fahrer 26				8			
Fahrer 27					8		
Fahrer 28	9	9		21			

Aus der Tabelle ist zu entnehmen, dass zwischen der Kalenderwoche 39 (KW 39) im Jahr 2016 und der Kalenderwoche 10 im Jahr 2017 insgesamt 179 Fahrer die 27 Fahrzeuge bewegt haben. Diese Liste wurde als CSV abgespeichert und gemeinsam mit dem Zeitstempel im Namen der Dateien verwendet, um die Ordnerstruktur neu anzulegen. Zwei weitere CSV Tabellen sind nötig, um die Daten korrekt zuordnen zu können.

Zum einen benötigt man eine Gegenüberstellung der Fahrzeugnamen zu den Kürzeln aus obiger Liste, zum anderen sind die Dateien nach dem Datumsformat JJJJ-MM-TT benannt, wodurch eine Zuordnung auf Kalenderwochen nicht direkt erfolgen kann. Die Inhalte der beiden CSV Dateien sind auszugsweise in den Tabellen 6 und 7 dargestellt.

Tabelle 6: Zuordnung KW zu Datum

Datum	KW	KW_ISO
2016-01-01	1	53
2016-01-02	1	53
2016-01-03	2	53
2016-01-04	2	1
…	…	…
2017-12-29	52	52
2017-12-30	52	52
2017-12-31	53	52

Tabelle 7: Zuordnung Fahrzeugbezeichnung zu Kürzel

Fahrzeugbezeichnung	Kürzel
PO 991 F SES785H6	2
PO 991 E SES792G6	3
PO 991 G SE0600H6	4
PO 991 F SE0586G6	5
…	…
PO 991 E SES991B2	26
PO 991 G SEB001F5	27
PO 991 F SE0745	28

Werden für weitere Messdaten fahrerabhängige Zusammenhänge gesucht, lassen sich über die oben genannten drei CSV-Dateien die Daten definieren, um eine solche Sortierung vornehmen zu können. Zusätzlich müssen, wie im Schritt der Konvertierung, Ordnerpfade für den Ursprung der Dateien, den Zielort der Daten und der Skripte, die die Sortierung vornehmen, angegeben werden.

5.3 Überblick über das Framework

Nach der Datenvorverarbeitung beginnt die Nutzung des Frameworks zur Konzeptionierung und Bewertung von lernenden Fahrzeugfunktionen. Das Framework selbst ist grob in drei Skripte untergliedert: param, start, main.

Das Param-Skript erlaubt die Veränderung von globalen Variablen, die an vielen Stellen im Programm genutzt werden. Darüber kann zum einen definiert werden, was ein Hotspot ist. Zum anderen lässt sich damit entscheiden, ob die Visualisierung angezeigt werden soll oder nicht. Des Weiteren können damit Parameter variiert werden, um den Lern-Algorithmus zu optimieren.

Am Anfang des Start-Skriptes wird zuerst das Param-Skript aufgerufen und alle Variablen in den Workspace von MATLAB eingelesen. Ansonsten dient das Start-Skript dazu die main-Methode für jeden Fahrer aufzurufen und am Ende die Daten derer in einer Tabelle auszugeben und abzuspeichern.

Das Main-Skript ist der Kern des Frameworks. Hier finden sich die Schritte 4 bis 9 wieder, die in Kapitel 3.2 beschrieben wurden. Es besteht selbst aus vielen kleinen Unterfunktionen, die vom Anwender allerdings nur über die im Param-Skript vorhanden Variablen verändert bzw. eingestellt werden sollen. In den nachfolgenden Kapiteln wird auf einige Unterfunktionen des main-Skriptes genauer eingegangen.

5.4 Lern-Algorithmus für Fahrwerksumschaltung

Der erste Schritt im Main-Skript ist das Ausführen des Lern-Algorithmus, um die gesuchten Positionen zu erlernen. Hierfür wurde ein einfacher Algorithmus entwickelt, der das Fahrwerk dem Kundenwunsch entsprechend automatisiert umstellen soll. Da im späteren Testfahrzeug kein vollwertiger PC, sondern nur ein kleines Steuergerät zur Verfügung steht, musste beim Entwurf darauf geachtet werden, dass der Algorithmus nicht sehr rechenintensiv ist.

Um den Code so schlank und einfach wie möglich zu halten, wurden sich einige Grundregeln überlegt nach denen der Algorithmus handeln soll.

1. Prädiktion für jede neue GPS Position
2. Prädiktionswert halten bis Ort mit häufiger Umschaltung
3. Lernen bei jeder neuen Umschaltung
4. Falls Position schon gelernt Durchfahrtszähler inkrementieren
5. Aggregation bei aufeinanderfolgenden Orten gleicher Prädiktion

Da Prädiktion, Lernen und Aggregation zu verschiedenen Zeitpunkten ausgeführt werden müssen, wurde der Algorithmus in diese drei Funktionen gegliedert: Predict, Learn, Aggregate.

Die Prädiktion wird für jede neue GPS-Position ausgeführt und soll den Fahrerwunsch vorhersagen. Dies kann nur für gelernte Positionen geschehen, deshalb wird immer wieder geprüft, ob man sich innerhalb eines Geofensters von einer gespeicherten Position befindet. Trifft diese Bedingung zu, wird geprüft, wie oft schon an dem Ort umgeschaltet wurde. Bei genügender Anzahl wechselt der Prädiktionswert in den angelernten Zustand, andernfalls wird der letzte Prädiktionswert gehalten. Wie viele Umschaltungen nötig sind, kann über den Parameter *min_Occurence_*Count eingestellt werden. Nachfolgend ist der Quellcode der Prädiktion dargestellt.

```
function Predicted_Mode =
PIDM_GeoEngine_Predict(Longitude,Latitude,Heading)
% function to predict mode based on learned and aggregated positions.
% to be executed cyclically for each GPS update
global PIDM_GeoEngine_Longitudes PIDM_GeoEngine_Latitudes
global PIDM_GeoEngine_Headings PIDM_GeoEngine_Modes
global PIDM_GeoEngine_Occurence_Count min_Occurence_Count
global PIDM_GeoEngine_Predicted_Reference_Index
global PIDM_GeoEngine_Predicted_Execute_Index

% Define Geofence
% 100 Meter umgerechnet in Grad bei Breitengrad 44°
delta_Longitude_fence = cosd(49) * 100 * 360 / 40000000;
% 100 Meter umgerechnet in Grad
delta_Latitude_fence = 100 * 360 / 40000000;
% Richtung in Grad
delta_Heading_fence = 45;

if (length(PIDM_GeoEngine_Longitudes) == 0)
    Predicted_Mode = 0; % Initial
end

for i = 1:length(PIDM_GeoEngine_Longitudes)
    % check if actual Position is near learned positions
    if ((abs(Longitude - PIDM_GeoEngine_Longitudes(i)) <
delta_Longitude_fence) ...
            && (abs(Latitude - PIDM_GeoEngine_Latitudes(i)) <
delta_Latitude_fence) ...
            && (abs(Heading - PIDM_GeoEngine_Headings(i)) <
delta_Heading_fence))

        PIDM_GeoEngine_Predicted_Reference_Index = i;
        % check if switchovers are often enough
        if(PIDM_GeoEngine_Occurence_Count(i) > min_Occurence_Count)
            PIDM_GeoEngine_Predicted_Execute_Index = i;
            Predicted_Mode = PIDM_GeoEngine_Modes(i); % set predicted Mode
        else
            if PIDM_GeoEngine_Predicted_Execute_Index == 0
                Predicted_Mode = 0;
            else
                % hold predicted Mode
                Predicted_Mode =
PIDM_GeoEngine_Modes(PIDM_GeoEngine_Predicted_Execute_Index);
            end
        end
    else
        if PIDM_GeoEngine_Predicted_Execute_Index == 0
            Predicted_Mode = 0;
        else
            % hold predicted Mode
            Predicted_Mode =
PIDM_GeoEngine_Modes(PIDM_GeoEngine_Predicted_Execute_Index);
        end
    end
end
```

Listing 4: Quellcode der Prädiktion

Um das Fahrerverhalten prädizieren zu können, muss es zuerst angelernt werden. Im Falle der Fahrwerksumschaltung wurde definiert, dass bei Betätigen der Fahrwerkstaste die Position gespeichert werden soll. Wird an einer gewissen Stelle der umgeschaltete Wert bereits prädiziert bedeutet das, dass der Ort bereits gelernt wurde. In diesem Fall wird die Anzahl an Umschaltungen in diesem Ort erhöht. Die Umsetzung der Lernfunktion ist in Listing 5 zu sehen.

```
function PIDM_GeoEngine_Learn(Longitude,Latitude,Heading,Mode_current)
% function to learn new positions, at which mode is changed
% to be executed each time the driver changes the mode

global PIDM_GeoEngine_Longitudes PIDM_GeoEngine_Latitudes
PIDM_GeoEngine_Headings
global PIDM_GeoEngine_Modes PIDM_GeoEngine_Occurence_Count
global PIDM_GeoEngine_Predicted_Reference_Index

% get actual predicted Mode
if (PIDM_GeoEngine_Predicted_Reference_Index == 0)
    Mode_predicted = 0;
else
    Mode_predicted =
PIDM_GeoEngine_Modes(PIDM_GeoEngine_Predicted_Reference_Index);
end
%check if switchover at actual position
if(Mode_current ~= Mode_predicted ||
PIDM_GeoEngine_Predicted_Reference_Index == 0)
    % Current Mode not yet learned at current position
    PIDM_GeoEngine_Latitudes = [PIDM_GeoEngine_Latitudes Latitude];
    PIDM_GeoEngine_Longitudes = [PIDM_GeoEngine_Longitudes Longitude];
    PIDM_GeoEngine_Headings = [PIDM_GeoEngine_Headings Heading];
    PIDM_GeoEngine_Modes = [PIDM_GeoEngine_Modes Mode_current];
    PIDM_GeoEngine_Occurence_Count = [PIDM_GeoEngine_Occurence_Count 1];
else
    % position is already learned --> increase Occurence Counter

PIDM_GeoEngine_Occurence_Count(PIDM_GeoEngine_Predicted_Reference_Index) =
...

PIDM_GeoEngine_Occurence_Count(PIDM_GeoEngine_Predicted_Reference_Index) +
1;
end
```

Listing 5: Quellcode des Lernens

Gibt es zwei gelernte benachbarte Orte, die beide zur selben Umschaltung führen, müssen diese zusammengefasst werden. Die Aggregationsfunktion fasst jene aufeinanderfolgenden Orte zusammen, an denen der gleiche Wert prädiziert wird. Der zuerst durchfahrene Punkt wird gespeichert und die Anzahl an Umschaltungen beider Orte wird aufsummiert. Die Implementierung ist nachfolgend zu sehen.

```
function
PIDM_GeoEngine_Aggregate(PIDM_GeoEngine_Predicted_Reference_Index_last)
% function to aggregate redundant positions.
% to be executed each time when "PIDM_GeoEngine_Predict" calculates
% the same mode as in the execution before with different
% PIDM_GeoEngine_Predicted_Reference_Index

global PIDM_GeoEngine_Longitudes PIDM_GeoEngine_Latitudes
PIDM_GeoEngine_Headings
global PIDM_GeoEngine_Modes PIDM_GeoEngine_Occurence_Count
global PIDM_GeoEngine_Predicted_Reference_Index
PIDM_GeoEngine_Predicted_Execute_Index

% sum up Occurence Counter
PIDM_GeoEngine_Occurence_Count(PIDM_GeoEngine_Predicted_Reference_Index_las
t) = ...

PIDM_GeoEngine_Occurence_Count(PIDM_GeoEngine_Predicted_Reference_Index_las
t) + ...

PIDM_GeoEngine_Occurence_Count(PIDM_GeoEngine_Predicted_Reference_Index);

% delete additional position
PIDM_GeoEngine_Latitudes(PIDM_GeoEngine_Predicted_Reference_Index) = [];
PIDM_GeoEngine_Longitudes(PIDM_GeoEngine_Predicted_Reference_Index) = [];
PIDM_GeoEngine_Headings(PIDM_GeoEngine_Predicted_Reference_Index) = [];
PIDM_GeoEngine_Modes(PIDM_GeoEngine_Predicted_Reference_Index) = [];
PIDM_GeoEngine_Occurence_Count(PIDM_GeoEngine_Predicted_Reference_Index) =
[];

% set Index on saved position
if (PIDM_GeoEngine_Predicted_Reference_Index <
PIDM_GeoEngine_Predicted_Reference_Index_last)
    % In this case the index has been decremented by one

if(PIDM_GeoEngine_Predicted_Reference_Index==PIDM_GeoEngine_Predicted_Execu
te_Index)
        PIDM_GeoEngine_Predicted_Execute_Index =
PIDM_GeoEngine_Predicted_Reference_Index_last - 1;

elseif(PIDM_GeoEngine_Predicted_Reference_Index<PIDM_GeoEngine_Predicted_Ex
ecute_Index)
        PIDM_GeoEngine_Predicted_Execute_Index =
PIDM_GeoEngine_Predicted_Execute_Index -1;
    end
    PIDM_GeoEngine_Predicted_Reference_Index =
PIDM_GeoEngine_Predicted_Reference_Index_last - 1;
else
```

```
if(PIDM_GeoEngine_Predicted_Reference_Index==PIDM_GeoEngine_Predicted_Execu
te_Index)
        PIDM_GeoEngine_Predicted_Execute_Index =
PIDM_GeoEngine_Predicted_Reference_Index_last;

elseif(PIDM_GeoEngine_Predicted_Reference_Index<PIDM_GeoEngine_Predicted_Ex
ecute_Index)
        PIDM_GeoEngine_Predicted_Execute_Index =
PIDM_GeoEngine_Predicted_Execute_Index -1;
    end
    PIDM_GeoEngine_Predicted_Reference_Index =
PIDM_GeoEngine_Predicted_Reference_Index_last;
end
```

Listing 6: Quellcode der Aggregation

Ob der Lern-Algorithmus das Fahrerverhalten nachahmen bzw. vorausschauen kann, muss nachträglich überprüft werden. Hierfür müssen vor allem alle Orte betrachtet werden, bei denen die Lernkomponente aufgerufen wird, also alle Umschaltpunkte. Wie diese gefunden und untersucht werden, wird in den nachfolgenden Kapiteln vorgestellt.

5.5 Hotspots erkennen und vereinen

In dieser Arbeit wurden Orte gesucht bei denen eine Fahrwerksumschaltung stattgefunden hat. Diese wurden dann als sog. Hotspots bezeichnet. Da das Framework allerdings für etwaige geobasierte lernende Funktionen allgemein nutzbar sein soll, benötigt man die Möglichkeit Hotspots auch auf andere Weise zu definieren.

Für das Projekt der prädiktiven Vorkonditionierung wurden Orte gesucht, bei denen das Fenster mind. 30 Sekunden am Stück geöffnet oder geschlossen wurde. Dafür musste die Hotspot-Erkennung um einige Funktionen erweitert werden, die, wie oben erwähnt, durch Änderungen im Param-Skript individuell aufgerufen werden können. Einerseits können Orte gespeichert werden, bei denen ein Schwellwert über- oder unterschritten wird. Dabei können verschiedene Signale sowohl per logischem „UND“ als auch per logischem „ODER“ miteinander verbunden werden.

Das kann zum Beispiel genutzt werden, um Koordinaten von Stellen zu bekommen mit hohen Quer- oder Längsbeschleunigungen, sog. Fahrdynamik-Hotspots. Für die Fahrwerksumschaltung musste das Drücken der Umschalttaste erkannt werden. Das ist im CAN-Signal durch ein Wechseln des Zustandes von Null zu Eins oder umgekehrt feststellbar. Deshalb wurde eine Funktion entworfen, die sich genau dann Punkte merkt, wenn sich das ausgewählte Signal verändert (On-Change). Als letzte Möglichkeit lassen sich Hotspots in festgelegten Abständen erzeugen, um so zufällige Punkte zum Prüfen des Algorithmus zu erhalten.

Zusammengefasst können folgende Fälle zur Hotspot Findung berücksichtigt werden:

- Vergleich eines Signals auf einen Schwellwert (>, <, =)
- Verknüpfung durch „UND“ und „ODER“ mehrerer Signal-Schwellwertpaare
- Erkennung von monoton steigenden oder monoton fallenden Sequenzen ausgewählter Signale
- Erkennung von wechselnden Werten eines Signals (On-Change)
- Erzeugung von Hotspots alle *x* Meter

Durch die vielfältigen Möglichkeiten lassen sich die Skripte auch in weiteren Projekten anwenden, sodass mithilfe des Frameworks neue Konzepte schnell bewertet werden können.

Da die Fahrwerkstaste nicht immer exakt an der gleichen Stelle betätigt wird und aufgrund der Ungenauigkeit des GPS ist es nötig, naheliegende Hotspots zu verbinden. Um Hotspots zusammenzuführen, muss ein Einzugsgebiet definiert werden, ab dem zwei Hotspots zusammengehören. Zusätzlich ist in den meisten Fällen die Richtung der Durchfahrt durch den Hotspot von großer Bedeutung. Eine Fahrwerksumschaltung wird beispielsweise oft vor einer Serpentinenstrecke getätigt, allerdings nie danach. Das führt dazu, dass ein Hotspot nicht mehr nur als Punkt definiert wird, sondern als Punkt mit einem Einzugsgebiet und einer Richtung. Die Darstellung eines Hotspots ist in Abbildung 5 zu sehen. Der grüne Punkt steht für den Hotspot in dem alle cyan-farbenen Punkte vereint wurden.

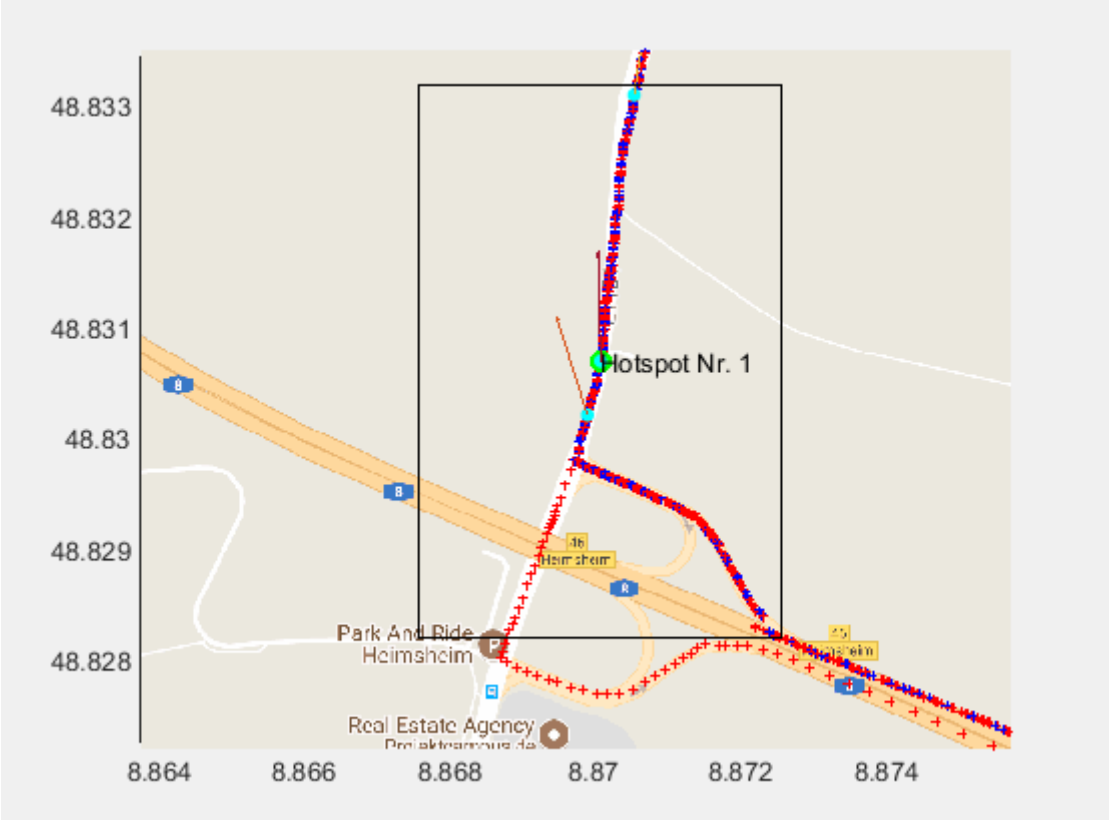

Abbildung 5: Darstellung eines Hotspots

5.6 Durchfahrtserkennung

Zur Bewertung lernender Algorithmen sowie zur Konzeptbestätigung von lernenden Fahrzeugfunktionen ist es nötig nicht nur Hotspots, sondern ganze Durchfahrten zu analysieren. Zur Konzeptbestätigung ist es zusätzlich zur Anzahl der Hotspots hilfreich zu wissen, wie häufig die Hotspots durchfahren wurden und in welchem Zustand sich das Fahrzeug befand. Man prüft also zunächst nicht die Güte des Algorithmus, sondern stellt erst sicher, dass es auch Nutzer der geplanten Funktion gibt. Deshalb wurde eine Funktion entwickelt, die sowohl die Anzahl der Durchfahrten mit erfüllter Bedingung (Betätigen der Fahrwerkstaste) als auch die Gesamtanzahl an Durchfahrten eines Hotspots zählt.

Eine erste einfache Umsetzung versuchte die Durchfahrten zu zählen, indem alle Punkte gesucht wurden, die innerhalb des Einzugsbereiches des Hotspots lagen und die Richtung mit einer Abweichung bis zu ±90° erfüllten. Anschließend wurde eine Zeitspanne angenommen, die zur Durchfahrt benötigt wird. Die ersten *x* Punkte, die innerhalb der Zeitspanne liegen, wurden gelöscht und der Zähler für eine Durchfahrt um Eins erhöht. Dies wurde solange wiederholt bis keine Punkte mehr übrig waren. Die Anzahl der Durchfahrten mit erfüllter Bedingung wurde durch die Menge an zusammengeführten Hotspots gezählt. In dem nachfolgenden Listing ist der Code der Durchfahrtserkennung dargestellt.

```
function cohs=detectCrossOver(all_data,merged_hs)
    global d_GPS d_dir th_time
    for ihs=1:length(merged_hs)
        hs=merged_hs(ihs);


        oc_all=0;
        for d=1:length(all_data)
            c=0;
            dps=all_data(d);
            tmp_dir=mod(dps.dir-hs.dir,360);
            tmp_dir2=mod(hs.dir-dps.dir,360);
            %all idx of datapoints of one drivethrough in one hs
            dpix=(dps.lat>hs.latcenter-d_GPS &...
                dps.long>hs.longcenter-d_GPS  & ...
                dps.lat<hs.latcenter+d_GPS  & ...
                dps.long<hs.longcenter+d_GPS & ...
                min(tmp_dir,tmp_dir2)<d_dir);
            dtime=dps.time(dpix);
            while(isempty(dtime)==0)
                cot=dtime(1);
```

```
                idx=(abs(dtime-cot)<th_time);
                dtime(idx)=[];
                c=c+1;
            end
            oc_all=oc_all+c;
        end

        disp([num2str((ihs-1)*100/length(merged_hs)) ' %'])

        merged_hs(ihs).conddt=length(merged_hs(ihs).datapoints);
        merged_hs(ihs).totaldt=oc_all;
    end
end
```

Listing 7: Code des Durchfahrtszählers

Beim Überprüfen der Werte stellten sich schnell erste Probleme heraus. Zum einen war die Definition der Zeit zu klein, sodass deutlich mehr Durchfahrten gezählt wurden als es letztendlich gab, zum anderen wurden ebenfalls entgegengesetzte Fahrten mitgezählt. Deshalb wurde zunächst die Zeit für eine Durchfahrt von 50 auf 150 Sekunden erhöht und die zulässige Abweichung der Richtung auf ±45° gesenkt, sodass Kurven noch zu einem gewissen Grad berücksichtigt werden, Fahrten in falscher Richtung allerdings nicht mehr gezählt werden.

Dies führte allerdings auch nicht zu zufriedenstellenden Ergebnissen, da je nach Geschwindigkeit und Strecke, die innerhalb des Geofences zurückgelegt wurde, die Fahrer eine Zeitspanne zwischen 40 und 400 Sekunden zum Durchqueren benötigten. Außerdem wurden nun viele Punkte durch die Richtungsvorgabe aussortiert. Deshalb ist ein neuer Ansatz entworfen worden, welcher nachfolgend vorgestellt werden soll. Bei der neuen Umsetzung wurde darauf geachtet, dass sie zeitunabhängig ist und die Richtung erst sehr spät einbezieht.

Zuerst wurde überlegt, wie eine Durchfahrt überhaupt definiert werden kann. Diese Definition lautet wie folgt: Das Fahrzeug muss sich dem Punkt erst nähern, also den Abstand immer weiter verkleinern und danach wieder davon entfernen, folglich den Abstand vergrößern. Gesucht sind zunächst also alle Punkte, die ein lokales Minimum zum Hotspot haben.

Anschließend muss, wie in der vorherigen Umsetzung geprüft werden, ob die Punkte sich innerhalb des Geofences befinden. Diesmal wurde die Richtung bei der Selektion nicht berücksichtigt. Der Codeabschnitt ist in Listing 8 zu sehen.

```
% Abstand zwischen 3 kontinuierlichen Punkten zum
% Hotspot berechnen
for k=2:length(dps.long)-1

    [~,dist]= geodist(dps.lat(k-1), dps.long(k-1), hs.latcentercenter,
hs.longcentercenter);
    [~,dist2]= geodist(dps.lat(k), dps.long(k), hs.latcentercenter,
hs.longcentercenter);
    [~,dist3]= geodist(dps.lat(k+1), dps.long(k+1), hs.latcentercenter,
hs.longcentercenter);

    % Wenn mittlerer Punkt kleiner als die Äußeren -->
    % lokales Minimum
    if(dist >= dist2 && dist3 >= dist2)
        minima = [minima k];
    end
end

% Suche alle Minima die innerhalb des Geofences liegen
dmin=(dps.lat(minima)>=hs.latcentercenter-d_GPS  & ...
    dps.long(minima)>=hs.longcentercenter-d_GPS   & ...
    dps.lat(minima)<=hs.latcentercenter+d_GPS   & ...
    dps.long(minima)<=hs.longcentercenter+d_GPS);

% Lösche alle Minima, die außerhalb des Geofences
% liegen
minima=minima(dmin>0);
```

Listing 8: Codeausschnitt der Minima-Suche

Um zeitunabhängig zu sein, wurde diesmal zur Erkennung einer Durchfahrt keine feste Zeit angenommen, sondern es wurden Eintritts- sowie Austrittsort aus dem Geofenster gesucht. Konnten keine gefunden werden, so musste es sich um den Start oder das Ende einer Fahrt handeln. Durch kurvige Streckenverläufe und die Ungenauigkeit von GPS kommt es zu einem weiteren Problem, welches beachtet werden muss. Innerhalb einer Durchfahrt können mehrere Minima liegen.

Damit diese nicht doppelt gezählt werden, müssen sie vorzeitig aussortiert werden und nur das näher liegende Minima gespeichert werden. In den nachfolgenden Listings sind der Ausschnitt zur Bestimmung der Start- und Endpunkte sowie das Aussortieren der falschen Minima zu sehen.

```
% Finde alle Start- und Endpunkte von weiteren
% Durchfahrten
k=1;
m=length(minima); % Schleifenvariable

while(m>0)
    % Finde den letzten Punkt vor dem ersten Minima,
    % der außerhalb des Geofences liegt --> Startpunkt
    % der Fahrt
    startpunkt{k}=find(dps.lat(1:(minima(k))) < hs.latcentercenter-d_GPS |
...
        dps.lat(1:(minima(k))) > hs.latcentercenter+d_GPS | ...
        dps.long(1:(minima(k))) < hs.longcentercenter-d_GPS | ...
        dps.long(1:(minima(k))) > hs.longcentercenter+d_GPS, 1, 'last');

    if(isempty(startpunkt{k}))
        startpunkt{k}=1;
    end

    % Finde den ersten Punkt nach dem Minimum,
    % der außerhalb des Geofences liegt -->
    % Endpunkt der Fahrt
    endpunkt{k}=find(dps.lat(minima(k):end) < hs.latcentercenter-d_GPS |
...
        dps.lat(minima(k):end) > hs.latcentercenter+d_GPS | ...
        dps.long(minima(k):end) < hs.longcentercenter-d_GPS | ...
        dps.long(minima(k):end) > hs.longcentercenter+d_GPS, 1, 'first');

    endpunkt{k} = endpunkt{k}+minima(k)-1;

    if(isempty(endpunkt{k}))
        endpunkt{k}=length(dps.lat);
    end
end
```

Listing 9: Codeausschnitt der Start- und Endpunkterfassung

```
% Wenn mehrere Minima innerhalb einer Durchfahrt
% gefunden werden, wird nur Minima, welches
% näher an HS ist gespeichert

% alle Minima einer Durchfahrt finden
temp_idx=(minima(k:end)-endpunkt{k}<0);
for i=1:k-1
    temp_idx=[0 temp_idx];
end
temp1=minima(temp_idx>0);
% Minima mit kleinstem Abstand zu HS finden
[~,dist]= geodist(dps.lat(temp1), dps.long(temp1), hs.latcenter,
hs.longcenter);
temp2=(find(dist==min(dist),1,'last'));
temp2=minima(temp2+k-1);
% Weiterentfernte Minima löschen
temp=(temp1~=temp2);
for i=1:k-1
    temp=[0 temp];
end
minima(temp>0)=[];
% Schleifenvariable um Anzahl der weiterentfernten Minima reduzieren
m=m-(length(temp1)-1);

% Dekrementieren der Schleifenvariable
m=m-1;
% Inkrementieren der ZÃ¤hlvariable der Start- und
% Endpunkte
k=k+1;
```

Listing 10: Codeausschnitt zum Aussortieren der falschen Minima

Sind die richtigen Minima mit ihren Start- und Endpunkten gefunden, lassen sich damit Arrays der Durchfahrten bilden. Innerhalb der Gesamtdurchfahrt wird nun der Punkt gesucht, welcher dem Hotspot am nächsten ist. Den Durchfahrtspunkt zu kennen ist wichtig, da der Fall auftreten kann, dass ein Fahrzeug zwar in den Einzugsbereich fährt und diesen wieder verlässt, allerdings ohne jemals in der Nähe des Hotspots gewesen zu sein. Mithilfe des zum Hotspot nahestehenden Punktes können diejenigen Fahrten aussortiert werden. Außerdem werden nun auch all diejenigen Fahrten aussortiert, die eine falsche Richtung aufweisen.

Da die Richtungserkennung sehr viele Fahrten aussortieren kann, wurden bei der neuen Umsetzung nicht nur die Einzelrichtungen mit der Hotspotrichtung verglichen, sondern zusätzlich die Durchschnittsrichtung der gesamten Durchfahrt. Dieses Mal wurde für die Festlegung der Richtungsabweichung empirisch vorgegangen, sodass anhand der vorliegenden Messdaten zwei verschiedene zulässige Abweichungen gefunden wurden.

Die Richtung des Durchfahrtszeitpunktes darf sich ±45° von der Hotspot-Richtung unterscheiden, wohingegen die durchschnittliche Durchfahrtsrichtung den 1,5-fachen Wert unterschreiten muss. Der Code dafür ist in Listing 11 dargestellt.

```
% Array der Durchfahrten bilden
for k=1:length(startpunkt)
    durchfahrt{k,d}=((startpunkt{k}+1):(endpunkt{k}-1));
end

% Ermittlung des Durchfahrtzeitpunktes
k=length(durchfahrt(:,d));
temp1=[];
while(k>0)
    dist=[];
    if(isempty(durchfahrt{k,d})==0)
        tmp=durchfahrt{k,d};
        % Suchen des Punktes mit dem kleinsten Abstand zum Hotspot
        [~,dist]= geodist(dps.lat(tmp), dps.long(tmp), hs.latcenter,
hs.longcenter);
        idx_min_dist=(find(dist==min(dist),1,'last'));
        temp1=[durchfahrt{k,d}(idx_min_dist) temp1];

        % Wenn kleinster Abstand>30m oder (Durchschnitt der Richtungen
einer
        % Durchfahrt und Richtung des Durchfahrtszeitpunktes) nicht
innerhalb
        % der erlaubten Abweichung zur Hotspot Richtung, dann Durchfahrt
lÃ¶schen
        min_dist=dist(idx_min_dist);
        if(min_dist>30 ...
                || (mean(min(mod(dps.dir(tmp)-
hs.dircenter,360),mod(hs.dircenter-dps.dir(tmp),360)))>(1.5*d_dir)...
                && min(mod(dps.dir(temp1(i))-
hs.dircenter,360),mod(hs.dircenter-dps.dir(temp1(i)),360))>(d_dir)))

            startpunkt(k)=[];
            temp1(i)=[];
            durchfahrt{k,d}=[];
            endpunkt(k)=[];
        else
            i=i+1;
        end
    end
    k=k-1;
end
% Durchfahrtszeitpunkte speichern
durchfahrtszeitpunkt{d}=temp1;
```

Listing 11: Codeausschnitt zum Finden der Durchfahrtszeitpunkte

Für die bessere Einstellbarkeit der Größe des Einzugsbereichs ist es hilfreich, wenn zusätzlich die Umschaltzeitpunkte bekannt sind. Damit kann empirisch herausgefunden werden, in welchem Bereich sich die Fahrer gleich verhalten und die Taste frühestens bzw. spätestens drücken. Um die Orte, an denen Umschaltungen getätigt wurden, herauszufinden, müssen wie beim Finden der Hotspots die Signaländerungen innerhalb jeder Durchfahrt gefunden werden. Die Anzahl an Durchfahrten mit Umschaltungen lassen sich prinzipiell genau wie die Gesamtzahl an Durchfahrten über die Länge der Durchfahrtszeitpunkte gewinnen. Allerdings soll der Zähler für die Umschaltungen nur einmal pro Durchfahrt inkrementiert werden, deshalb muss die Variable bei mehreren Umschaltungen während einer Durchfahrt wieder um die zusätzlichen Umschaltungen reduziert werden. Dargestellt als Code ist die Erkennung der Umschaltpunkte in Listing 12.

```
% Herausfinden der Durchfahrten mit Umschaltpunkt
temp1=[];
for k=1:length(durchfahrt(:,d))
    cnt=0;
    temp=durchfahrt{k,d};
    if(isempty(temp)==0)
        for i=2:length(temp)
            % Umschaltszeitpunkte finden
            if(dps.Fahrprogramm(temp(i-1))~=dps.Fahrprogramm(temp(i)))
                temp1 = [temp1 (i-1)];
                cnt=cnt+1;
            end
        end
    end
    % Wenn mehrere Umschaltungen pro Durchfahrt:
    % Reduzieren von Zählvariable für Anzahl
    % der Durchfahrten mit Umschaltung
    if(cnt>1)
        oc_cond = oc_cond -(cnt-1);
    end
end
% Umschaltszeitpunkte speichern
temp_idx_cond{d} = temp1;

% Berechnung der Anzahl der Durchfahrten ohne und
% mit Umschaltungen
oc_all = oc_all+length(temp_durchfahrt);
oc_cond = oc_cond+length(temp_idx_cond{d});
```

Listing 12: Codeausschnitt zum Finden der Umschaltpunkte

Abschließend werden alle Daten noch im jeweiligen Hotspot-Objekt gespeichert. Ein zusätzlicher Vorteil der neuen Umsetzung ist, dass Hotspots gezielt innerhalb ihrer Durchfahrt genauer analysiert werden können, da diese nur gespeichert werden muss und dann in allen anderen Funktionen nutzbar ist. Der gesamte Programmcode der Funktion ist im Anhang A2 zu sehen.

5.7 Auswahl und Berechnung des Gütekriteriums

Die verschiedenen gängigen Größen zur Gütebewertung eines lernenden Algorithmus wurden eingangs in Kapitel 3.3 vorgestellt. In diesem Kapitel soll kurz erläutert werden, welche Größen in der Arbeit betrachtet werden und anschließend die Umsetzung im Code dafür gezeigt werden.

Bei der Vorstellung der einzelnen Bewertungsgrößen wurde bereits darauf eingegangen, dass sich bestimmte Paare in den vergangenen Jahren etabliert haben. Eines davon ist zum Beispiel Precision und Recall. Es bietet die Möglichkeit sowohl die Trefferquote als auch die Genauigkeit zu bewerten, sodass eine ausreichend gute Aussage über die Qualität des Algorithmus gemacht werden kann. Außerdem wurde zusätzlich der F-Score ausgewählt, um eine schnelle Bewertung anhand nur einer Messgröße vornehmen zu können.

Ein weiterer Grund für die Verwendung von Precision und Recall ist, dass sie leicht zu analysieren und umzusetzen sind. Außerdem findet sich viel Literatur, sodass die Ergebnisse einfacher und besser interpretiert und diskutiert werden können. In Kapitel 8 wird zudem aufgezeigt, wie die Größen noch zur Optimierung des Lern-Algorithmus herangezogen bzw. erweitert werden können.

Die Bewertung der Güte des Algorithmus beschränkt sich auf den Bereich der Hotspots mit ihrem Einzugsgebiet, da der Algorithmus nur lernt, wenn eine Umschaltung getätigt wird. Das bedeutet, dass sich der Prädiktionswert zwischen den Hotspots nicht verändert. Für die Berechnung der Güte werden also nur die im vorherigen Kapitel beschriebenen Durchfahrten analysiert. Precision und Recall werden jeweils pro Durchfahrt und für den gesamten Hotspot berechnet. In den nachfolgenden Listings ist die Berechnung der True Positives, True Negatives, False Positives und False Negatives sowie der Precision- und Recall-Werte zu sehen.

```
% Werte mit 0 füllen
tp{i,d}=zeros(size(hs.real{i,d}));
fp{i,d}=zeros(size(hs.real{i,d}));
tn{i,d}=zeros(size(hs.real{i,d}));
fn{i,d}=zeros(size(hs.real{i,d}));

% Zuordnen der Werte in Tabelle
tp{i,d}(hs.predict{i,d} & hs.real{i,d})=1;
fn{i,d}(~hs.predict{i,d} & hs.real{i,d})=1;
fp{i,d}(hs.predict{i,d} & ~hs.real{i,d})=1;
tn{i,d}(~hs.predict{i,d} & ~hs.real{i,d})=1;

% Durchschnitt über Durchfahrt bilden
tp_mean{i,d} = mean(tp{i,d});
fp_mean{i,d} = mean(fp{i,d});
tn_mean{i,d} = mean(tn{i,d});
fn_mean{i,d} = mean(fn{i,d});

% Precision und Recall für Durchfahrt berechnen
PAR_DT{i,d} = PrecisionAndRecall(tp_mean{i,d}, fp_mean{i,d}, tn_mean{i,d},
fn_mean{i,d});
PAR_DT{i,d} = PAR_DT{i,d}.calculatePrecision(PAR_DT{i,d});
PAR_DT{i,d} = PAR_DT{i,d}.calculateRecall(PAR_DT{i,d});
PAR_DT{i,d} = PAR_DT{i,d}.calculateFScore(PAR_DT{i,d}, 1);

% Durchschnittswerte der Durchfahrten speichern
tp_overall = [tp_overall PAR_DT{i,d}.tp];
fp_overall = [fp_overall PAR_DT{i,d}.fp];
tn_overall = [tn_overall PAR_DT{i,d}.tn];
fn_overall = [fn_overall PAR_DT{i,d}.fn];
```

Listing 13: Codeausschnitt der Berechnung von Precision und Recall für eine Durchfahrt

```
% Werte für Hotspot bilden
tp_overall = mean(tp_overall);
fp_overall = mean(fp_overall);
tn_overall = mean(tn_overall);
fn_overall = mean(fn_overall);

% Precision und Recall für HotSpot berechnen
merged_hs(ihs).PrecisionAndRecall=PrecisionAndRecall(tp_overall,
fp_overall, tn_overall, fn_overall);
merged_hs(ihs).PrecisionAndRecall =
merged_hs(ihs).PrecisionAndRecall.calculatePrecision(merged_hs(ihs).Precisi
onAndRecall);
merged_hs(ihs).PrecisionAndRecall =
merged_hs(ihs).PrecisionAndRecall.calculateRecall(merged_hs(ihs).PrecisionA
ndRecall);
merged_hs(ihs).PrecisionAndRecall =
merged_hs(ihs).PrecisionAndRecall.calculateFScore(merged_hs(ihs).PrecisionA
ndRecall, 1);
```

Listing 14: Codeausschnitt der Berechnung von Precision und Recall für einen Hotspot

In Listing 13 ist zu erkennen, wie die Metriken definiert sind. True Positive sind alle Werte, bei denen sowohl Prädiktion als auch tatsächlicher Wert auf Eins sind. Sind wiederum beide Werte gleich, aber nicht Eins, so wird ein True Negative gezählt. Unterscheiden sich die Werte für Prädiktion und tatsächlichem Wert, werden entweder False Positive, falls die Prädiktion Eins ist oder False Negative, falls die Prädiktion Null ist, auf Eins gesetzt.

Mit den gemittelten Werten lassen sich der Precision sowie der Recall Wert der Durchfahrt errechnen. Die Berechnung erfolgt nach den Formeln, die im Theoriekapitel 3.3 aufgezeigt wurden. Die Umsetzung für die Berechnung der Precision ist in dem nachfolgenden Listing dargestellt.

```
function c = calculatePrecision(c)
display(' ');
display(['calculate Precision...']);
try
    tp=cell2mat(c.tp);
    fp=cell2mat(c.fp);
catch
    tp=c.tp;
    fp=c.fp;
end
try
    c.precision = tp./(tp+fp);
    display(['Precision = ' num2str(c.precision)]);
    display('Calculation finished.');
    display(' ');
catch
    display('Calculating failed.');
    display(' ');
end
%c.precision(isnan(c.precision))=0;
end
```

Listing 15: Code für die Berechnung der Precision

Recall und F-Score werden analog nach den Formeln aus 3.3 berechnet. Die gesamte Funktionalität ist in Anhang A3 zu sehen. Um die Metriken verstehen und interpretieren zu können, müssen diese für den Anwender visualisiert werden. Wie dies umgesetzt wurde, wird im nächsten Kapitel vorgestellt.

5.8 Visualisierung

Eine Visualisierung dient im Allgemeinen dazu, Zusammenhänge einfach und übersichtlich darstellen zu können, sodass außenstehenden Personen schnell verständlich gemacht werden kann, worum es geht, was die Ergebnisse sind bzw. was sie bedeuten. Deshalb werden nachfolgend verschiedene Darstellungen gezeigt, welche helfen sollen neue Konzepte zu bestätigen als auch den Algorithmus zu bewerten.

Für die Konzeptbestätigung ist es wichtig Fahrer zu finden, die sich einer Idee entsprechend verhalten. In dieser Arbeit wurden deshalb Fahrer gesucht, die wiederholt die Fahrwerkstaste an der gleichen Stelle betätigt haben. Anders gesagt sind Fahrer gesucht, die viele Hotspots haben. Ein weiteres Kriterium, welches geeignete Fahrer von den anderen unterscheidet, ist die durchschnittliche Anzahl an Durchfahrten mit und ohne Umschaltung. Diese Daten werden tabellarisch dargestellt und erlauben so die Anzahl aller Fahrer zu finden, die das gewünschte Verhalten zeigen. Außerdem lassen sich so gezielt weitere Analysen der ausgewählten Fahrer vornehmen. Die Übersicht ist in nachfolgendem Bild zu sehen.

	Driver	HS	DTs	Cond_DTs	driver_precision	driver_recall	driver_FScore
139	62	0	NaN	NaN	NaN	NaN	NaN
140	63	1	6	3	0.5434	0.7045	0.6135
141	64	0	NaN	NaN	NaN	NaN	NaN
142	65	0	NaN	NaN	NaN	NaN	NaN
143	66	0	NaN	NaN	NaN	NaN	NaN
144	67	0	NaN	NaN	NaN	NaN	NaN
145	68	0	NaN	NaN	NaN	NaN	NaN
146	69	0	NaN	NaN	NaN	NaN	NaN
147	7	0	NaN	NaN	NaN	NaN	NaN
148	70	1	10	4	0.5048	0.9963	0.6701
149	71	0	NaN	NaN	NaN	NaN	NaN
150	72	0	NaN	NaN	NaN	NaN	NaN
151	73	4	9.5000	4	0.9033	0.9291	0.9144
152	74	1	3	3	0.8919	0.9976	0.9418
153	75	1	8	3	0.2963	1	0.4571
154	76	2	7	3	0.5754	0.8042	0.6168
155	77	4	6.7500	3	0.5569	0.7245	0.5922
156	78	0	NaN	NaN	NaN	NaN	NaN
157	79	0	NaN	NaN	NaN	NaN	NaN
158	8	0	NaN	NaN	NaN	NaN	NaN
159	80	0	NaN	NaN	NaN	NaN	NaN
160	81	0	NaN	NaN	NaN	NaN	NaN

Abbildung 6: Tabellarische Darstellung der Güte mehrerer Fahrer

Aus der Tabelle kann herausgelesen werden, wie viele Fahrer eine hohe Wiederholgenauigkeit aufweisen. Dadurch können Entscheidungen über die Nutzung der geplanten Funktion getroffen werden. Die Tabelle hilft aber nicht nur bei der Konzeptbestätigung. Sie kann auch zur Analyse des Lern-Algorithmus herangezogen werden. Denn wie in Abbildung 6 zu sehen ist, werden auch Precision, Recall und F-Score der Fahrer dargestellt. Somit können sehr schnell Fahrer gefunden werden, bei denen das Anlernen gut funktioniert bzw. umgekehrt Fahrer gefunden werden, die schlechte Precision und Recall Werte aufweisen. Auch hier kann man anschließend vertieft auf diese Fahrer eingehen und analysieren, was die Gründe für die hohen bzw. niedrigen Werte der Metriken sind. Dafür müssen weitere Visualisierungen geschaffen werden, die es erlauben einen Fahrer genauer zu analysieren.

Ein erster Überblick schafft hier eine Karte, die anzeigt, wo der Fahrer überall gefahren ist und an welchen Stellen die Hotspots liegen. So lassen sich zum Beispiel Verknüpfungen mit der Straßenart (Autobahn, Bundes-, Landstraße) bilden. Außerdem kann eine erste Verteilung der Hotspots erkannt werden. Die Fahrten werden hierfür jeweils als GPS-Punkte auf die Karte aufgetragen und die Information des aktuellen Fahrprogramms wird durch die Farben blau (Normal) und rot (Sport) hinterlegt. In Abbildung 7 ist für den Fahrer 77 eine Karte erstellt worden.

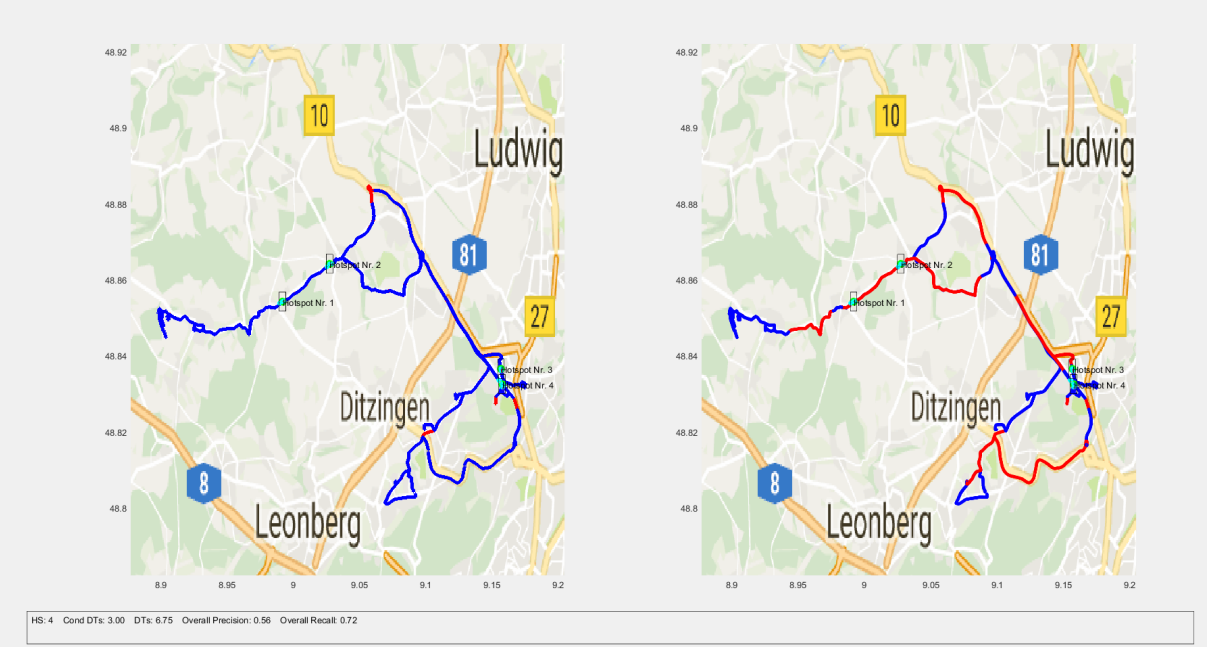

Abbildung 7: Karte aller Fahrten des Fahrers 77

Im linken Teil sind die tatsächlich gefahrenen Fahrprogramme angezeigt, während im rechten Teil die prädizierten Fahrprogramme zu sehen sind. Außerdem kann man erkennen, dass die ersten beiden Hotspots auf einer Landstraße liegen und die weiteren beiden innerhalb einer Stadt. Insgesamt sind die Hotspots zwar verteilt, liegen aber dennoch jeweils auf einer Route. Anhand der Precision- und Recall-Werte kann man zusammen mit den blauen und roten Punktekurven einschätzen, wie gut der Algorithmus funktioniert. Um noch genauere Aussagen über die Güte des Algorithmus für den Fahrer zu treffen, müssen die einzelnen Hotspots aufbereitet werden. Hierfür wurde die in Abbildung 8 dargestellte Visualisierung gewählt.

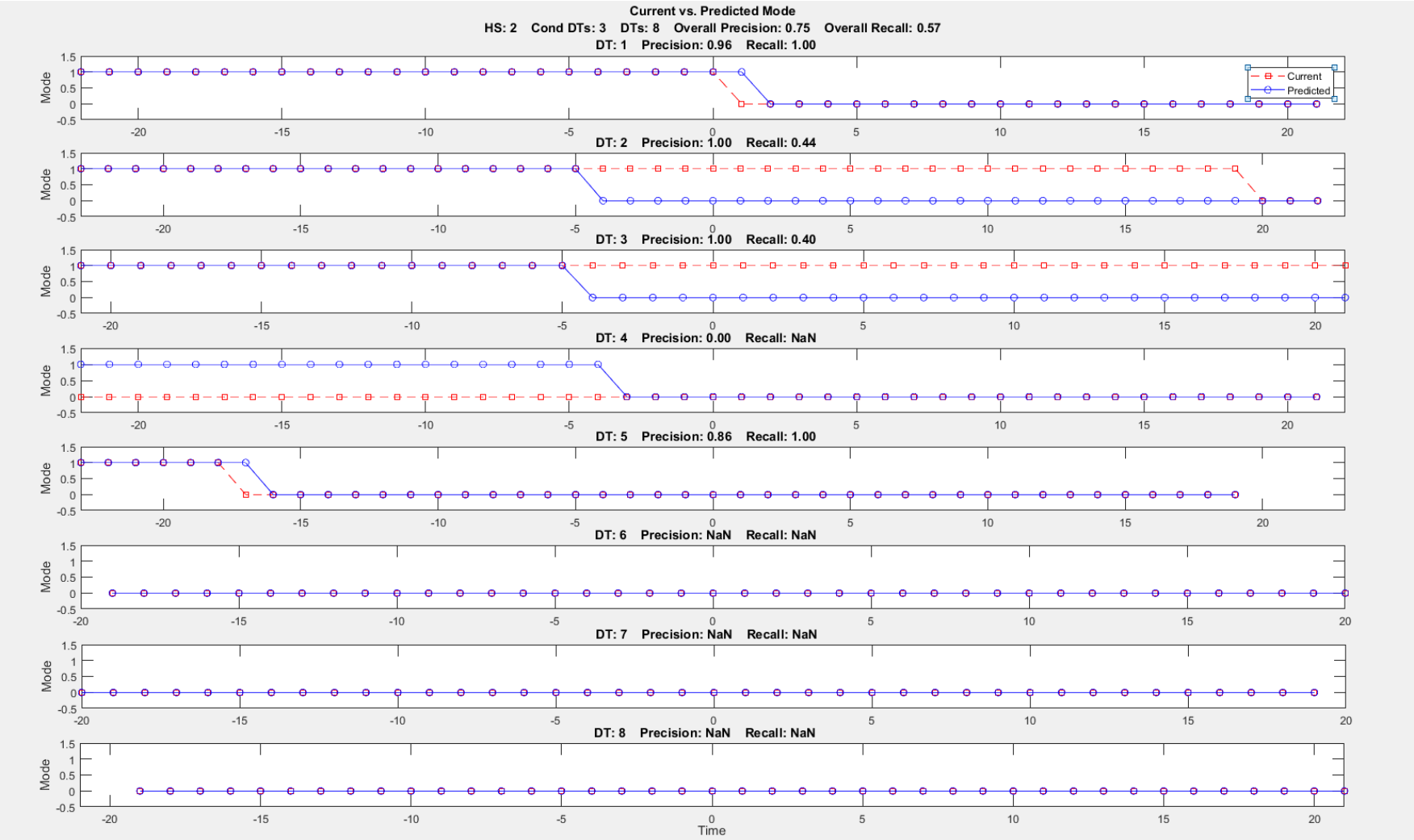

Abbildung 8: Current vs. Predicted Mode (Fahrer 77 HS2)

Dieses Bild hilft dabei die Werte für Precision und Recall nachzuvollziehen. Es enthält die Werte für den gesamten Hotspot sowie für jede Durchfahrt. Außerdem sind die Werte des Fahrprogramms (Rot) und des prädizierten Fahrprogramms (Blau) für jede Durchfahrt dargestellt. Mit der Definition der True Positives u.a. kann durch diese Plots der Wert für Precision und Recall nachvollzogen werden. Zudem kann man das Lernen des Algorithmus erkennen.

Bei der ersten Durchfahrt wird eine Rückschaltung erlernt, die man in den folgenden Durchfahrten wiederfindet. Die roten und blauen Linien unterscheiden sich hier, da bei den Messdaten kein Lern-Algorithmus im Fahrzeug vorhanden war und deshalb keine automatische Umschaltung stattgefunden hat. In Durchfahrt 2 und Durchfahrt 5 ist zudem das wiederholte Rückschalten innerhalb des gleichen Geofensters zu erkennen. Die Werte für Precision und Recall sind in den letzten drei Durchfahrten nicht errechenbar, da es nur True Negatives gibt und diese in der Berechnung von Precision und Recall keine Berücksichtigung finden.

Die Interpretation von Precision und Recall ist anhand der zweiten Durchfahrt gut zu sehen. Anfangs sind sowohl Prädiktion als auch realer Wert auf Eins, weshalb hier ein Abschnitt an True Positives vorliegt. Später prädiziert der Algorithmus eine Rückschaltung, wodurch sich die Werte unterscheiden und eine Phase von False Negatives erkennbar ist. Am Ende schaltet der Fahrer erneut zurück, weshalb dort ein Abschnitt von True Negatives dargestellt wird. Mittelt man nun die Werte der True Positives und False Negatives, so kann man die Werte für Precision und Recall berechnen. Anhand der Zeitabschnitte der einzelnen Phasen lassen sich diese auch abschätzen. So ist die Dauer der True Positive Phase etwas kürzer als der Bereich mit False Negative-Werten, weshalb man auf einen Recall knapp unter 0.5 schätzen kann. Die Precision ist Eins, da es keine False Positive Phase in dieser Durchfahrt gab.

Nun haben die einzelnen Durchfahrten bessere und schlechtere Metriken. Um diese noch besser verstehen zu können, hilft es die einzelnen Durchfahrten auf einer Karte darzustellen. Hierbei kann man erkennen, ob von der Route eventuell abgewichen wurde, also eine Abzweigung gewählt und deshalb keine Umschaltung getätigt wurde. Eine solche Visualisierung ist nachfolgend zu sehen.

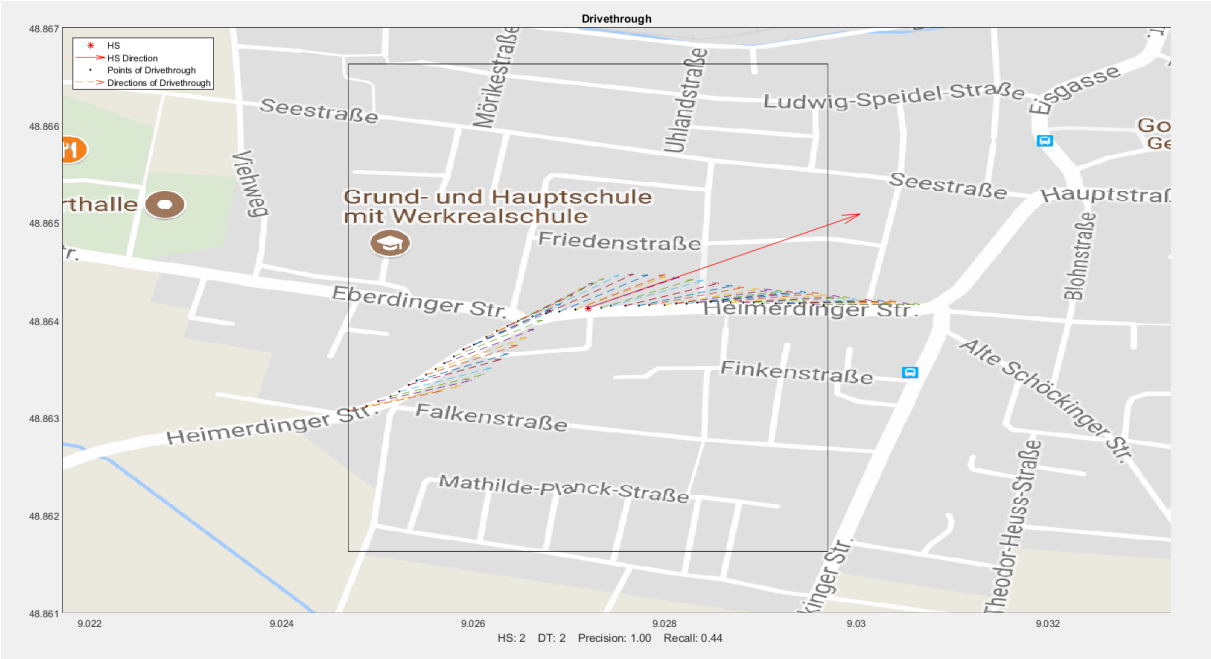

Abbildung 9: Karte einer Durchfahrt des Fahrers 77 (HS2)

In der Abbildung sind alle GPS-Punkte der Durchfahrt mit ihrer Richtung dargestellt. Der Punkt in Sternform mit längerem Pfeil steht für den Hotspot, der bei einer anderen Durchfahrt gefunden wurde. Anhand dieser Darstellungen wurden auch die empirischen Werte für den maximalen Abstand und die maximale Abweichung der Richtung gefunden. Würde der Fahrer in die Eberdinger Str. abbiegen, würde er den Hotspot nie durchfahren und möchte deshalb auch keine Umschaltung prädiziert bekommen. Genauso könnte man schlechte Werte in den Metriken erklären, wenn der Fahrer nach dem Hotspot in eine Seitengasse abbiegt. Deshalb ist es wichtig mit beiden Darstellungen zu arbeiten, wenn man einen Algorithmus analysiert, da der Fehler sonst möglicherweise an der falschen Stelle gesucht wird.

Zusammengefasst kann man sagen, dass mithilfe der vorgestellten Visualisierungen eine oberflächliche und falls gewünscht eine tiefer greifende Analyse durchgeführt werden kann. Die Darstellungen helfen in ihrer Form nicht nur der Bewertung, sondern dienen im ersten Schritt dazu, die Konzepte für kommende Funktionen auf Umsetzbarkeit und Sinn zu überprüfen. Im nächsten Kapitel sollen die Ergebnisse dieser Arbeit anhand einiger Bilder der vorgestellten Visualisierungen erklärt werden.

6 Ergebnisse

In diesem Kapitel sollen alle wichtigen Ergebnisse der Arbeit vorgestellt werden. Dabei werden die Resultate des Lernalgorithmus, aber auch die Konzeptbestätigung der automatisierten Fahrwerksumschaltung und die Erkenntnisse, die für eine fahrzeugseitige Umsetzung gewonnen werden konnten, erklärt.

6.1 Konzeptbestätigung

Zuerst wurde überprüft, ob sich ein Algorithmus zur automatisierten Fahrwerksumschaltung überhaupt lohnt. Hierfür wurden Nutzer als Personen, welche mehr als zwei Umschaltungen am gleichen Ort vornehmen, definiert. Anschließend wurde untersucht, wie viele der 179 Testpersonen dieses Kriterium erfüllen und somit als potenzielle Kunden wahrgenommen werden können. Aus der Tabelle, die in Abbildung 6 dargestellt ist, kann herausgelesen werden, dass 36 Fahrer eine hohe Wiederholgenauigkeit aufweisen. Dies entspricht 20% der Testpersonen. Hochgerechnet auf alle seit 2004 weltweit verkauften Porsche Fahrzeuge und mit der Annahme von mindestens zwei Fahrern pro Fahrzeug, zählt man insgesamt über drei Millionen Porsche Fahrer [22]. Mit dem errechneten Prozentsatz bedeutet das, dass es auf der Welt über 600.000 potenzielle Kunden für die Funktion gibt. Damit kann man sagen, dass es sinnvoll ist, eine automatisierte Fahrwerksumschaltung in zukünftigen Fahrzeugen anzubieten.

6.2 Fahrzeugseitige Umsetzung

Für die Konzeptbestätigung war es essentiell eine verlässliche Durchfahrtserkennung zu haben, um die richtige Anzahl an Umschaltungen zu erkennen und somit Fahrer als Nutzer zu bewerten. Hierbei wurde vor allem die richtige Richtungserkennung als großes Problem identifiziert, welches durch eine Kombination verschiedener Bedingungen gelöst wurde. Zum einen wurde nicht nur die Richtung eines Punktes mit der Hotspot Richtung verglichen, sondern auch der Durchschnitt aller

Durchfahrtspunkte. Zum anderen wurde ein Minimalabstand zum Hotspot definiert, um Fahrten auszusortieren, bei denen der Fahrer vor der ausschlaggebenden Stelle bereits abgebogen ist. Diese beiden Probleme konnten vorzeitig mit dem Framework behandelt werden, um im Testfahrzeug bessere Ergebnisse zu erzielen. Allerdings ist die Lösung mit der Durchschnittsrichtung der Durchfahrt im Zielfahrzeug nur schwer umsetzbar. Einerseits benötigt dies viel Speicherplatz, andererseits sind die prädiktiven Streckendaten, um die in der Zukunft liegenden Punkte zu berücksichtigen, auf dem geplanten Steuergerät nicht vorhanden. Deshalb soll in den kommenden Wochen eine einfachere Lösung für dieses Problem gefunden werden.

6.3 Bewertung des Lern-Algorithmus

Die Hauptaufgabe des Frameworks dient letztendlich dazu einen gut funktionierenden Lern-Algorithmus für geobasierte Funktionen anhand von Messdaten zu entwickeln. In dieser Arbeit wurde zunächst ein sehr einfacher Algorithmus entworfen, um Rechenleistung auf der Zielhardware zu sparen. Die Bewertung des Algorithmus wurde mithilfe der Visualisierungen aus Kapitel 5.8 vorgenommen. Es ist zu beachten, dass nur diejenigen Fahrer, die eine hohe Wiederholgenauigkeit aufweisen, analysiert wurden.

Der Algorithmus funktioniert nach den in Kapitel 5.4 beschriebenen Regeln und kann über den dort beschriebenen Parameter verändert werden. Mithilfe des Parameters kann entschieden werden, wie schnell die Positionen gelernt werden. In der ersten Analyse wurde die Geopositionen direkt nach der ersten Umschaltung erlernt und in allen weiteren Durchfahrten idealerweise eine Umschaltung prädiziert. Nachfolgend sind die Ergebnisse dieser Parametrierung zu sehen. In Abbildung 10 sind die Precision und Recall Werte einiger Fahrer und die Gesamtgütebewertung des Algorithmus dargestellt.

	Driver	HS	DTs	Cond_DTs	driver_precision	driver_recall	driver_FScore
159	80	0	NaN	NaN	NaN	NaN	NaN
160	81	0	NaN	NaN	NaN	NaN	NaN
161	82	0	NaN	NaN	NaN	NaN	NaN
162	83	0	NaN	NaN	NaN	NaN	NaN
163	84	0	NaN	NaN	NaN	NaN	NaN
164	85	0	NaN	NaN	NaN	NaN	NaN
165	86	0	NaN	NaN	NaN	NaN	NaN
166	87	0	NaN	NaN	NaN	NaN	NaN
167	88	0	NaN	NaN	NaN	NaN	NaN
168	89	0	NaN	NaN	NaN	NaN	NaN
169	9	1	8	3	0.6037	0.8082	0.6911
170	90	1	5	3	1	0.9930	0.9965
171	91	0	NaN	NaN	NaN	NaN	NaN
172	92	0	NaN	NaN	NaN	NaN	NaN
173	93	1	4	3	0.9834	0.5642	0.7171
174	94	0	NaN	NaN	NaN	NaN	NaN
175	95	0	NaN	NaN	NaN	NaN	NaN
176	96	0	NaN	NaN	NaN	NaN	NaN
177	97	0	NaN	NaN	NaN	NaN	NaN
178	98	0	NaN	NaN	NaN	NaN	NaN
179	99	0	NaN	NaN	NaN	NaN	NaN
180	Overall	82	8.2106	3.6616	0.7712	0.8298	0.7677

Abbildung 10: Tabellarische Darstellung der Gesamtgüte

Hieraus ist zu erkennen, dass insgesamt 82 Orte mit durchschnittlich 8,2 Durchfahrten gefunden wurden. Bei 45% davon wurde ebenfalls mind. eine Umschaltung vorgenommen. Die Precision- und Recall-Werte liegen im oberen Bereich mit 0,77 und 0,83, jedoch schwanken die Werte der Fahrer deutlich. Es ist sowohl ein Fahrer mit schlechten Precision-Werten als auch ein Fahrer mit schlechten Recall-Werten erkennbar. Für den Fahrer 34, dessen Daten in Abbildung 11 dargestellt sind, wird nachfolgend analysiert, weshalb die Werte nicht ideal sind.

	Driver	HS	DTs	Cond_DTs	driver_precision	driver_recall	driver_FScore
105	31	0	NaN	NaN	NaN	NaN	NaN
106	32	0	NaN	NaN	NaN	NaN	NaN
107	33	0	NaN	NaN	NaN	NaN	NaN
108	34	12	9.3333	4.0833	0.7129	0.7069	0.6983
109	35	0	NaN	NaN	NaN	NaN	NaN
110	36	0	NaN	NaN	NaN	NaN	NaN

Abbildung 11: Güte des Fahrers 34

Die Funktionsweise des Lern-Algorithmus ist am einfachsten aus den Grafiken erkennbar, die Prädiktion und realen Wert gegenüberstellen. Anhand des ersten Hotspots, welcher eine Precision von 0,96 und einen Recall von 0,69 hat, soll diese aufgezeigt werden.

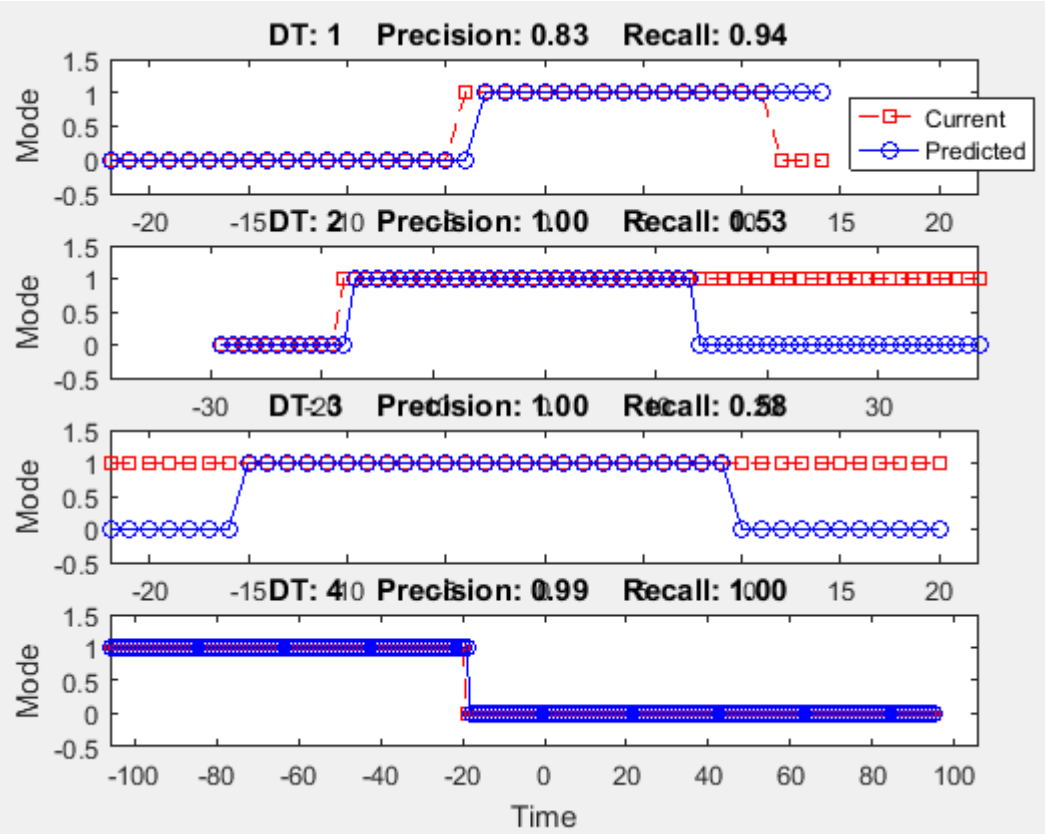

Abbildung 12: Current vs. Predicted Mode (Fahrer 34 HS1)

Im obersten Bild ist zu erkennen, wie eine Umschaltung stattfindet und die Position sofort gelernt und prädiziert wird. Die zweite Umschaltung in dieser Durchfahrt wird ebenfalls gelernt, aber nicht prädiziert. Dieses Fehlverhalten kann mit dem Austritt aus dem Geofenster erklärt werden. Die Durchfahrt ist in der nachfolgenden Abbildung dargestellt.

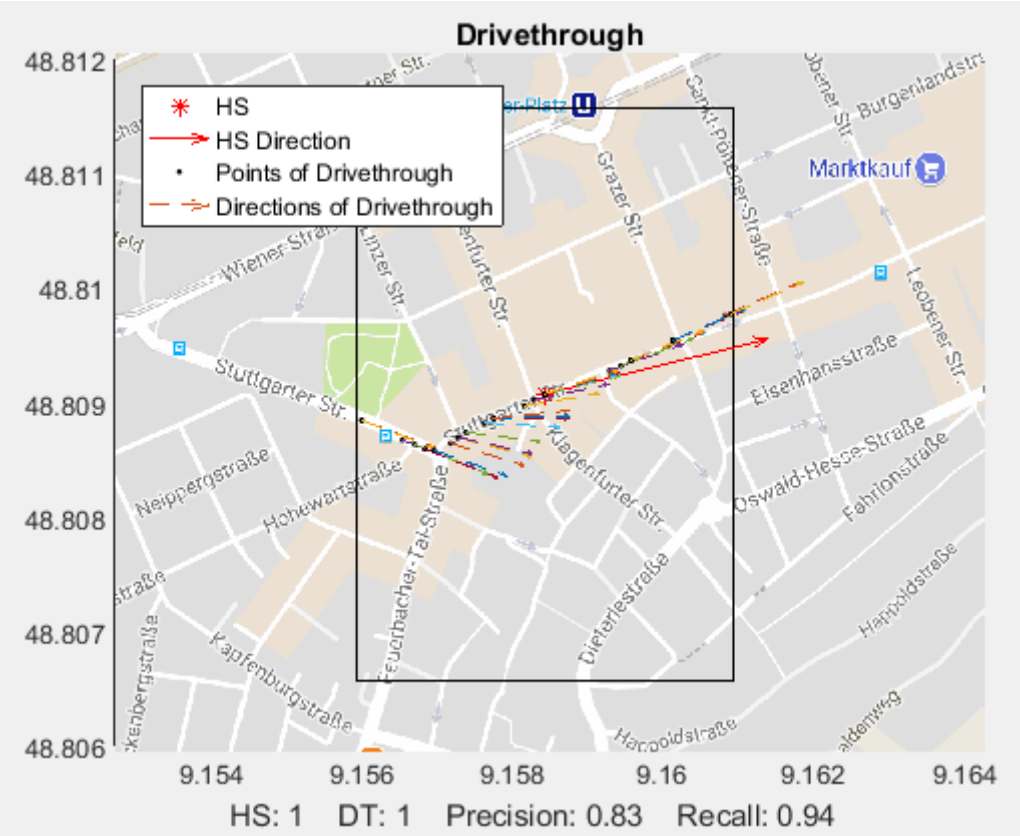

Abbildung 13: Durchfahrt mit zwei Umschaltungen (Fahrer 34)

Anhand der zweiten Durchfahrt kann bestätigt werden, dass das Verhalten richtig gelernt wurde, da dieses Mal eine Rückschaltung prädiziert wurde. Der Zeitpunkt der Umschaltung ist nach vorne gerückt, da die Prädiktion bereits umschaltet, sobald sich ein gelernter Umschaltpunkt in einem Luftlinienabstand von 100 Metern befindet. Im Bild der dritten Durchfahrt ist zum Eintrittszeitpunkt eine Fehlprädiktion vorhanden. Dies hängt damit zusammen, dass die Prädiktion eine Rückschaltung zu einem früheren Zeitpunkt der Strecke gelernt hat, der Fahrer bei dieser Fahrt allerdings nicht zurückgeschaltet hat. Das bedeutet, dass schlechte Werte der Metriken eines Hotspots nicht unbedingt am Hotspot selbst liegen.

Ein weiterer Schwachpunkt des Prädiktionsalgorithmus ist mit einem anderen Hotspot dieses Fahrers erklärbar. Nachfolgend sind die realen und prädizierten Werte für den zehnten Hotspot zu sehen.

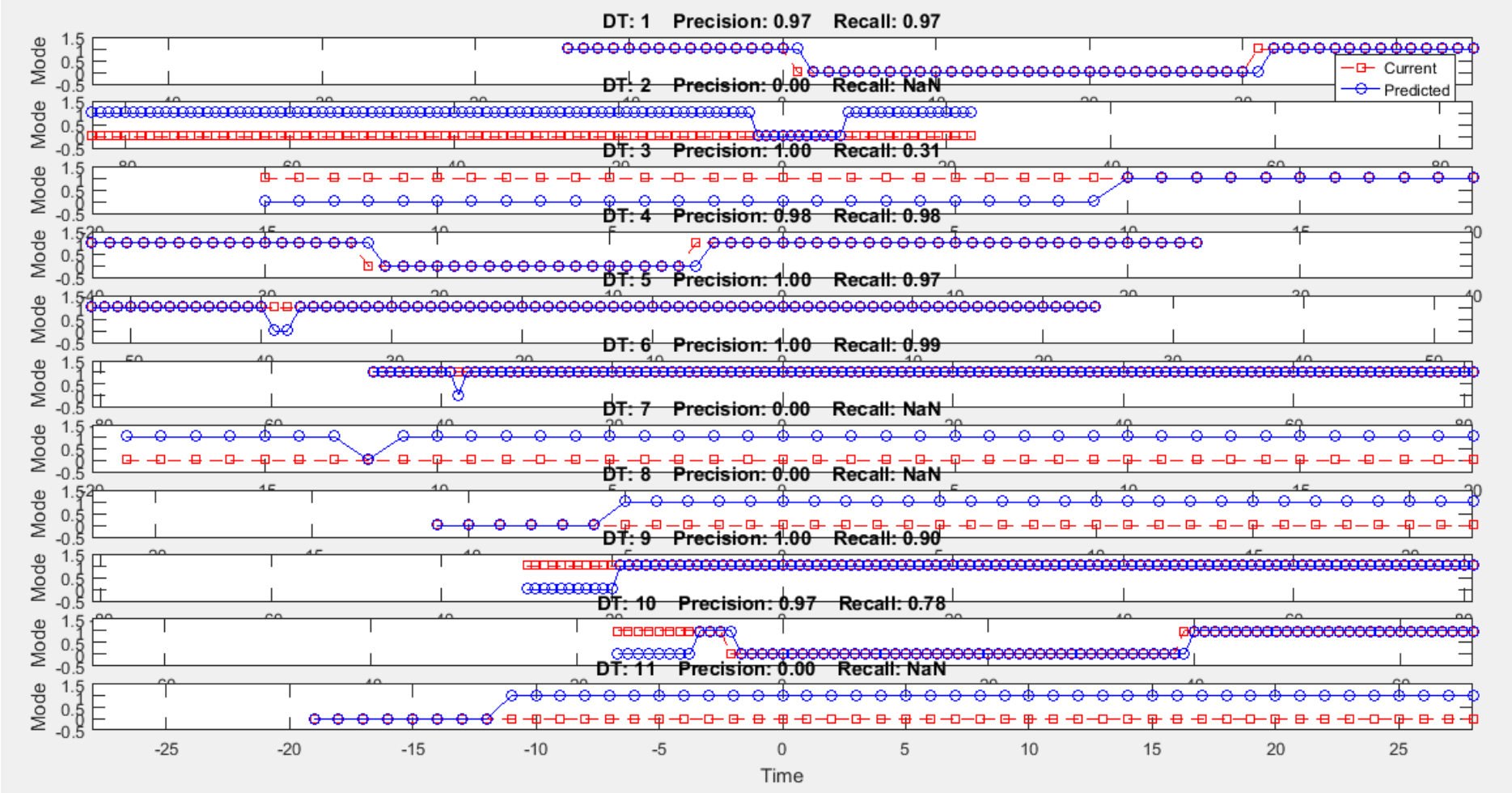

Abbildung 14: Current vs. Predicted Mode (Fahrer 34 HS10)

In Abbildung 14 kann ab Durchfahrt 4 die Problematik von mehreren örtlich naheliegenden Umschaltungen gezeigt werden. In der angesprochenen Durchfahrt wird eine Rückschaltung mit folgendem Wechsel in den Sport Modus vorgenommen. Aufgrund der Funktionsweise der Prädiktion wird in Durchfahrt 5 nur sehr kurz der Normalmodus prädiziert. Das hängt damit zusammen, dass die beiden Umschaltorte nur knapp mehr als 100 Meter entfernt voneinander liegen und der zweite Ort, im gespeicherten Array an Umschaltpunkten, einen höheren Index hat. Negativ zu bemängeln ist hier zusätzlich, dass die Metriken gute Werte zurückgeben, obwohl kaum einem Fahrer eine zweisekündige Rückschaltung gefallen würde.

Ein weiteres Ergebnis bei der Analyse der Bewertung kann mithilfe des Fahrers 70 erklärt werden. Bei diesem Fahrer lässt sich zum einen zeigen, dass die Aggregation gemeinsam mit der Prädiktion Probleme bereiten kann. Zum anderen kann ein grundsätzliches Problem der Bewertung gezeigt werden. Seine prädizierten Werte sind in Abbildung 15 zu sehen.

Die Aggregation vereint zwei gleichwertige Umschaltungen in einem Geofenster, addiert die Anzahl an Durchfahrten und löscht einen der beiden Punkte. Das kann dazu führen, dass sich der Umschaltpunkt immer weiter der Grenze des Geofensters nähert. Durch den zusätzlichen Einzugsbereich der Prädiktion kann dies bewirken, dass die Prädiktion bereits außerhalb des Geofensters des Hotspots schon die Umschaltung vornehmen würde. Das heißt der Umschaltpunkt wäre um bis zu 350 Meter nach vorne verschoben.

Weiterhin ist zu erkennen, dass dieser Fahrer nicht bei jeder Durchfahrt in den Sportmodus wechselt. Dies führt dementsprechend bei den Durchfahrten ohne Wechsel des Fahrprogramms dazu, dass die Metriken sehr schlechte Werte aufweisen. Möchte man den Fahrerwunsch darstellen, so sind die Metriken dafür geeignet, soll allerdings dargestellt werden, ob der Algorithmus sich nach seinen Regeln verhält, müssen diese neu definiert werden. Zudem zeigt sich, dass eine Analyse des Fahrerwunsches mit diesen Messdaten schwer ist, da nicht abschätzbar ist, ob der Fahrer ein Problem mit einer automatisierten Umschaltung in den Durchfahrten 8 bis 10 hätte.

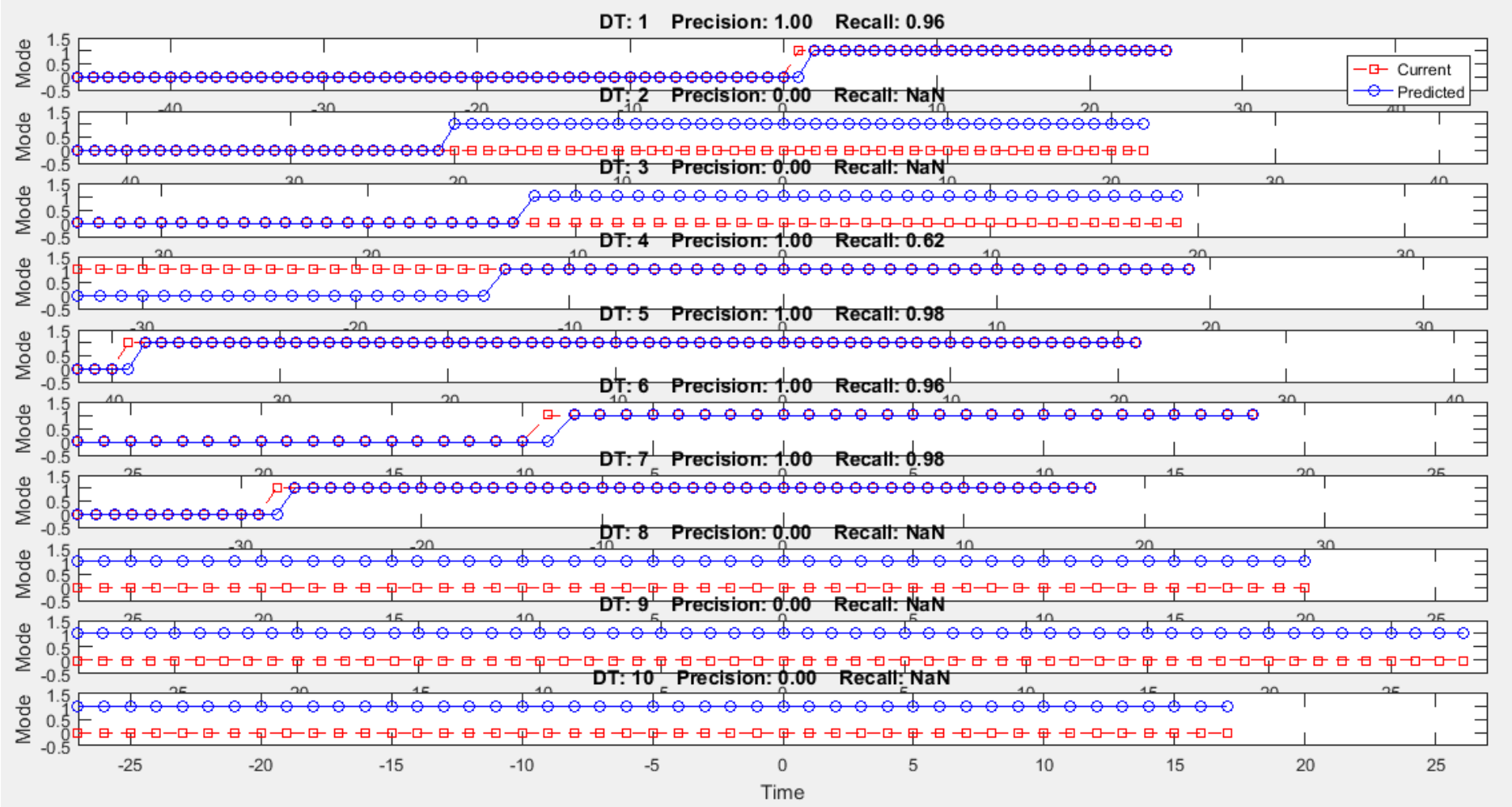

Abbildung 15: Current vs. Predicted Mode (Fahrer 70 HS1)

Ein Verhalten, welches einige Fahrer zeigten, war versehentliches Betätigen der Taste, sodass innerhalb kurzer Zeit oft zwei oder drei Umschaltungen getätigt wurden. Um fehlerhaftes Prädizieren in diesen Fällen zu vermeiden, wurde die Grenze für die nötige Anzahl an Umschaltungen an einem Hotspot auf drei erhöht. Eine Folge davon können schlechte Metriken bei den ersten Durchfahrten sein, was ein weiterer Grund für eine neue Definition der Metriken ist. Für die neue Parametrierung ist die Gesamtgüte in nachfolgender Abbildung zu sehen.

	Driver	HS	DTs	Cond_DTs	driver_precision	driver_recall	driver_FScore
159	80	0	NaN	NaN	NaN	NaN	NaN
160	81	0	NaN	NaN	NaN	NaN	NaN
161	82	0	NaN	NaN	NaN	NaN	NaN
162	83	0	NaN	NaN	NaN	NaN	NaN
163	84	0	NaN	NaN	NaN	NaN	NaN
164	85	0	NaN	NaN	NaN	NaN	NaN
165	86	0	NaN	NaN	NaN	NaN	NaN
166	87	0	NaN	NaN	NaN	NaN	NaN
167	88	0	NaN	NaN	NaN	NaN	NaN
168	89	0	NaN	NaN	NaN	NaN	NaN
169	9	1	8	3	0.1339	0.1884	0.1566
170	90	1	5	3	NaN	0	NaN
171	91	0	NaN	NaN	NaN	NaN	NaN
172	92	0	NaN	NaN	NaN	NaN	NaN
173	93	1	4	3	0.7308	0.2435	0.3653
174	94	0	NaN	NaN	NaN	NaN	NaN
175	95	0	NaN	NaN	NaN	NaN	NaN
176	96	0	NaN	NaN	NaN	NaN	NaN
177	97	0	NaN	NaN	NaN	NaN	NaN
178	98	0	NaN	NaN	NaN	NaN	NaN
179	99	0	NaN	NaN	NaN	NaN	NaN
180	Overall	82	8.2106	3.6616	0.5596	0.3581	0.4827

Abbildung 16: Gesamtgüte bei neuer Parametrierung

Aus der Abbildung ist eine deutlich schlechtere Gesamtgüte des Algorithmus abzulesen. Durch genauere Analyse der Fahrer kann dieses Ergebnis erklärt werden. Der Lern-Algorithmus verhält sich nicht wie erwartet, sondern prädiziert die Umschaltungen willkürlich. Bei manchen Fahrern werden gar keine Umschaltungen gelernt, obwohl häufiger umgeschaltet wurde, wie in Abbildung 17 zu sehen ist. Andere Fahrer hatten bereits prädizierte Umschaltvorgänge, obwohl noch nicht genügend Durchfahrten vorhanden waren. Dies ist in Abbildung 18 dargestellt.

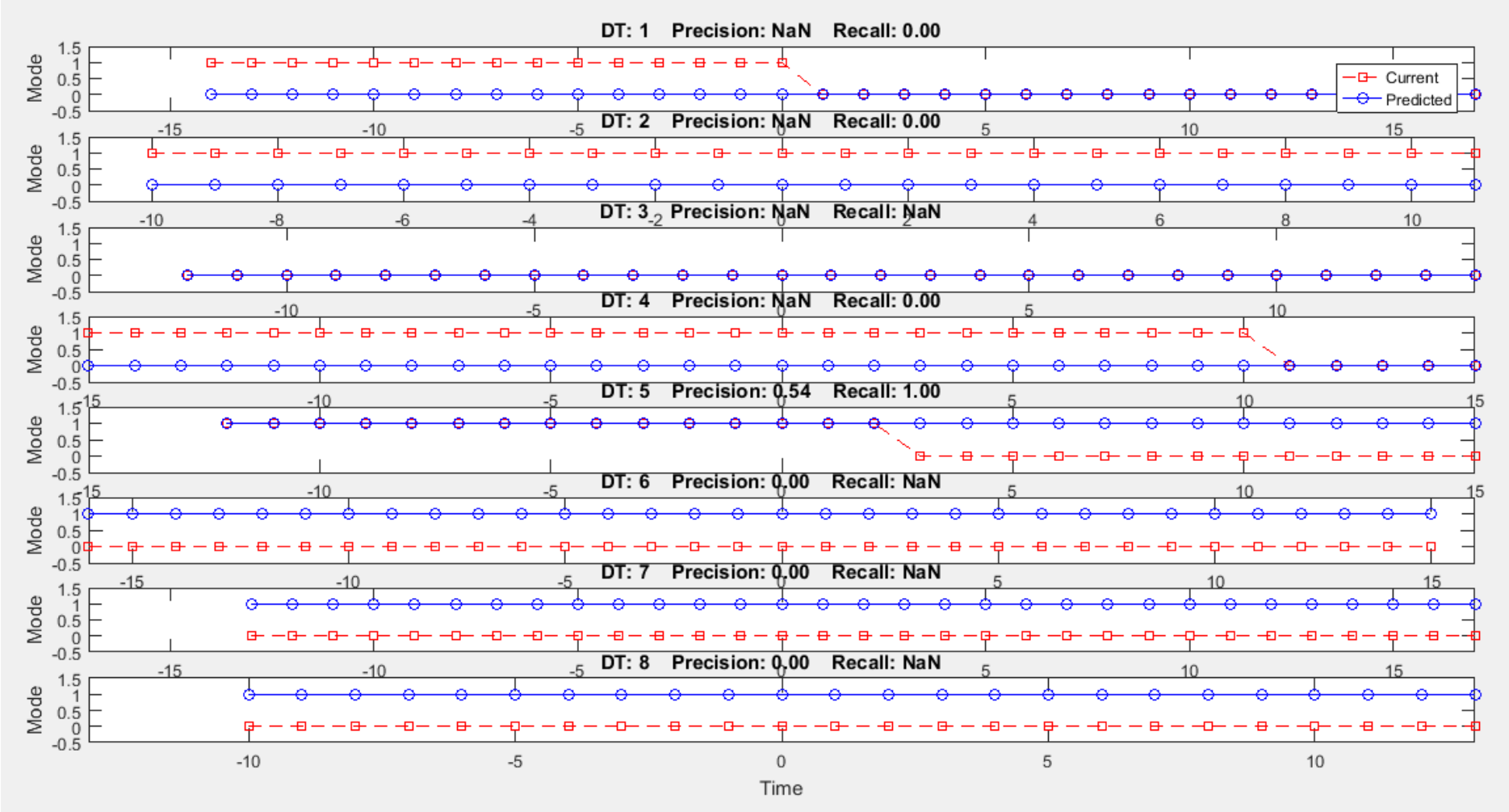

Abbildung 17: Current vs. Predicted Mode (Fahrer 9 HS1)

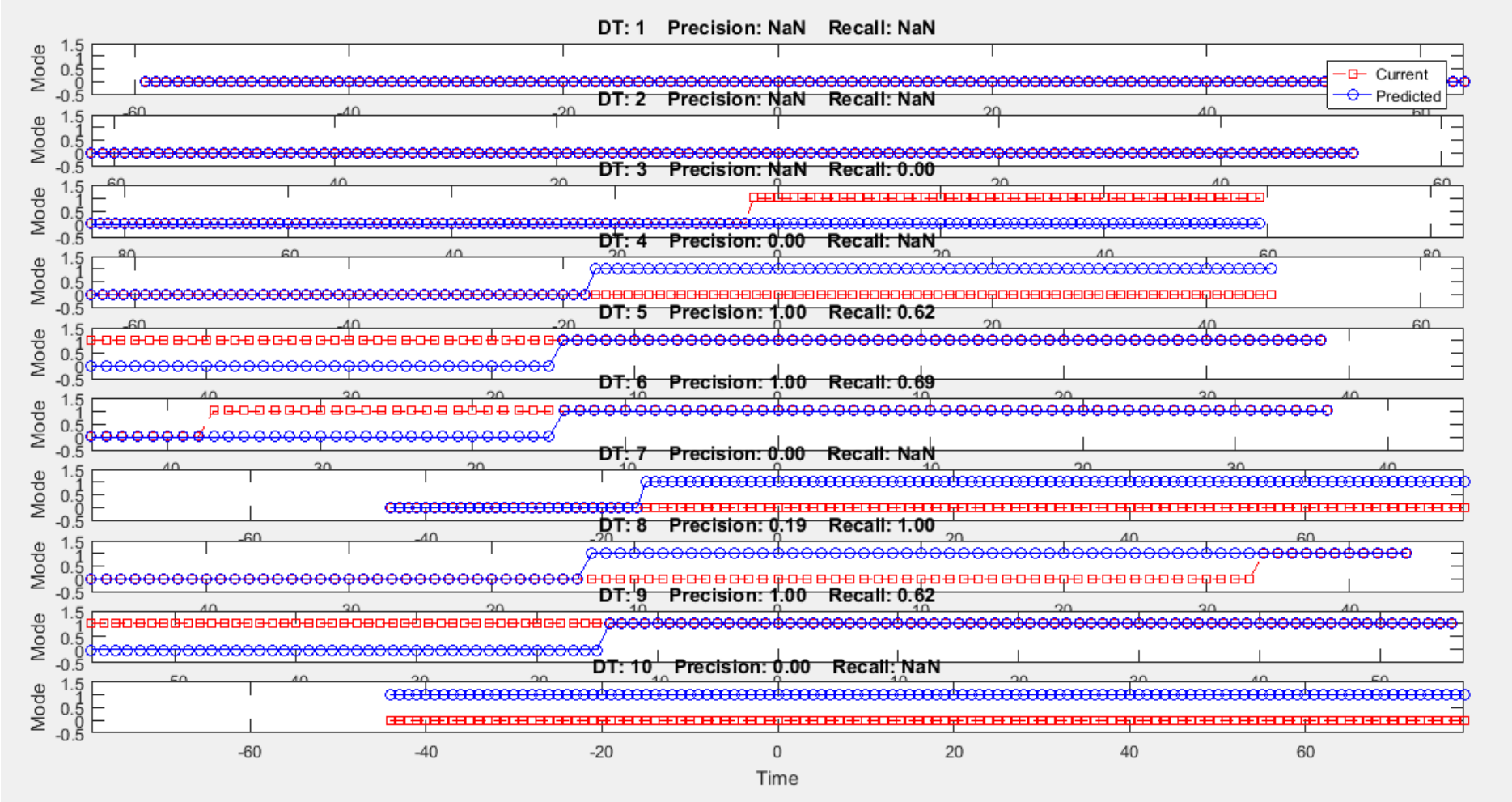

Abbildung 18: Current vs. Predicted Mode (Fahrer 34 HS2)

Die oben dargestellte Problematik ist durch den Lern- und Aggregationsprozess erklärbar. Während bei der Prädiktion die gelernten Punkte innerhalb eines Geofensters gesucht werden, findet sowohl beim Lernen als auch beim Aggregieren keine Ortsbeschränkung Berücksichtigung. Das führt dazu, dass Umschaltpunkte, die mehrere Kilometer entfernt sind, zusammengefasst werden. Dadurch ist es möglich, dass die Anzahl an Umschaltpunkten für einen Hotspot erhöht wird, sodass eine Prädiktion bereits in der zweiten Durchfahrt vorkommt. Andererseits kann dies für einen anderen Hotspot auch bedeuten, dass die Umschaltungen nicht richtig zugeordnet werden und somit niemals die nötige Anzahl erreicht wird. Dieses Problem tritt bei der Prädiktionsschwelle von Null nur sehr selten auf. Je höher die Schwelle allerdings gesetzt wird, desto häufiger besteht das Problem. Da eine weitere Analyse des vorliegenden Algorithmus bis zur Einführung einer Ortsbeschränkung für Lernen und Aggregation nicht sinnvoll erscheint, wurde die Analyse hier beendet.

Insgesamt kann als Ergebnis festgehalten werden, dass der Lern-Algorithmus in seiner aktuellen Umsetzung nicht verwendet werden kann. Das Verlernen von Geopositionen und eine Berücksichtigung eines Einzugsbereichs für Lernen und Aggregieren muss unbedingt hinzugefügt werden, bevor der Algorithmus in einem Testfahrzeug nutzbar wird. Die Metriken sollten ebenfalls neu definiert werden, um anhand derer die Probleme des Algorithmus suchen zu können. Aktuell erweisen sich die Bilder der Gegenüberstellung von prädiziertem und aktuellem Wert des Fahrprogramms als hilfreichstes Tool, um die Probleme des Algorithmus zu finden. Dennoch kann durch die Metriken ein grober Anhaltspunkt über die Güte des Algorithmus getroffen werden.

7 Zusammenfassung

In dieser Arbeit wurde unter anderem der Aufbau eines Frameworks zur Konzeptionierung und Bewertung von lernenden geobasierten Fahrzeugfunktionen beschrieben. Besonders die Erkennung der Durchfahrtsrichtung durch die gelernten Positionen stellte eine große Herausforderung dar. Durch die Datenanalyse konnten wichtige Erkenntnisse gewonnen werden, um eine verbesserte Erkennung der Richtung im Testfahrzeug umzusetzen. Die Idee des Erlernens von Umschaltpunkten des Fahrwerks wurde vorgestellt und anhand der Messdaten gezeigt, dass eine solche Funktion von ca. 20% der Testpersonen genutzt werden würde.

Dafür wurde anschließend ein einfacher Lern-Algorithmus entworfen und durch Parametervariation anhand der eingeführten Metriken analysiert und bewertet. Der Algorithmus verhält sich bei sofortiger Prädiktion den aufgestellten Regeln entsprechend, allerdings konnte der Fahrerwunsch auch oft nicht richtig prädiziert werden. Die Güte des Algorithmus war vor allem bei Fahrern ohne hohe Wiederholgenauigkeit schlecht. Aber auch bei einigen „guten“ Fahrern wurden niedrige Werte ermittelt. Dies liegt an der Definition der Metriken und könnte mit realen Messdaten behoben werden. Bei neuer Parametrierung, mit Prädiktion erst ab dem dritten Umschalten, konnten nur unlogische, nicht reproduzierbare Ergebnisse erzielt werden. Grund dafür ist vor allem das fehlende Geofenster für die Lern- und die Aggregationsfunktion.

Insgesamt ist festzustellen, dass der Algorithmus noch nicht zufriedenstellend arbeitet und die Definition der Metriken überprüft werden muss, um den Lernprozess verbessern zu können. Dennoch konnten viele wichtige Kenntnisse bereits aus den Messdaten gewonnen werden, die bei der fahrzeugseitigen Implementierung von Nutzen sein werden.

8 Ausblick

In dieser Arbeit wurde die Basis für einen funktionsfähigen Lern-Algorithmus geschaffen, welcher allerdings noch optimiert werden muss, um die Umschaltungen, dem Fahrerwunsch entsprechend, zu erfüllen. Hierfür muss sowohl für die Lern- als auch die Aggregationsfunktion eine Ortsbeschränkung eingeführt werden. Außerdem muss eine Möglichkeit gefunden werden, verschiedene Orte wieder abzutrainieren. Da die Metriken vor allem bei Durchfahrten ohne Umschaltung schlechte Werte liefern, könnten Messdaten aus einem Testfahrzeug mit der Prototypensoftware helfen. Dadurch könnte das Kundenverhalten bei automatisierter Umschaltung in diesen Durchfahrten analysiert werden.

Sollten anschließend immer noch nicht die gewünschten Ergebnisse erzielt werden, kann zum einen ein Lern-Algorithmus nach dem Prinzip des Machine Learnings entwickelt oder zum anderen ein Optimierungsalgorithmus für die Metriken hinzugezogen werden. Ein Verfahren zur Optimierung ist eine Precision-Recall-Kurve aufzustellen und diese in eine Receiver-Operating-Characteristics-Kurve (ROC-Kurve) zu transferieren. Über Parametervariation wird der maximale Flächeninhalt unter der ROC-Kurve gesucht und ergibt so den bestmöglichen Lern-Algorithmus.

Ein letzter offener Punkt ist die weitere Anpassung des Frameworks auf andere Use Cases. Hierfür müssen noch einige Verallgemeinerungen vorgenommen werden, um nur noch mit dem Start- und Parameter-Skript arbeiten zu müssen. Denkbar wäre hier auch eine grafische Oberfläche, über die die Einstellungen an das Framework übergeben werden können.

9 Literaturverzeichnis

[1] L. T. L. W. A. V. Feng Xia, „Internet of Things,“ *INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS,* 2012.

[2] B. Zerche, „Entwicklung eines Frameworks zur Konzeptbestätigung ortsbasierter automotive Dienste und deren Abbildbarkeit auf Fog Architekturen,“ Weissach, 2017.

[3] IPETRONIK, „M-LOG V3,“ 06 2016. [Online]. Available: https://www.ipetronik.com/sites/default/files/field_product_file/ipe_datasheet_m_log_v3_prelim.pdf. [Zugriff am 16 08 2017].

[4] CarMedialab, „Flea 3,“ 26 02 2016. [Online]. Available: http://carmedialab.de/wp-content/uploads/2016/02/Flea3_26.02.2016.pdf. [Zugriff am 16 08 2017].

[5] o.V., „Uni Bundeswehr München - Map Matching,“ 2009. [Online]. Available: https://www.unibw.de/inf4/professuren/geoinformatik/lehre/skripten/skripte/skripten_ht_09/map-matching-2009.pdf. [Zugriff am 16 08 2017].

[6] P. D.-I. h. V. Schwieger, „Uni Stuttgart - Kartengestütze Ortung:Ablaufschema,“ 02 09 2016. [Online]. Available: https://www.uni-stuttgart.de/ingeo/forschung/projekte/mapmatching_ablaufschema.html. [Zugriff am 16 08 2017].

[7] P. D.-I. h. V. Schwieger, „Uni Stuttgart - Kartengestütze Ortung,“ 02 09 2016. [Online]. Available: https://www.uni-stuttgart.de/ingeo/forschung/projekte/mapmatching.html. [Zugriff am 16 08 2017].

[8] M. Meyer, Signalverarbeitung: Analoge und digitale Signale, Systeme und Filter, Braunschweig/Wiesbaden: vieweg, 2000, p. 275.

[9] W. Stangl, „Gütekriterien empirischer Forschung,“ 2017. [Online]. Available: http://arbeitsblaetter.stangl-taller.at/FORSCHUNGSMETHODEN/Guetekriterien.shtml. [Zugriff am 16 08 2017].

[10] G. Gigerenzer, Das Einmaleins der Skepsis: Über den richtigen Umgang mit Zahlen und Risiken, Berlin: Berlin Verlag, 2014.

[11] R. Kuhlen, T. Seeger und D. Strauch, Grundlagen der praktischen Information und Dokumentation, 5. Hrsg., Bd. 1, München: K. G. Saur Verlag GmbH, 2004, p. 228 ff..

[12] C. V. v. Rijsbergen, Information Retrieval, London: Butterworth, 1979, p. 112 ff..

[13] Stanford University, *Text Classification and Naive Bayes,* Stanford, 2011, p. 45 ff..

[14] D. Pham und S. Dimov, Rapid Manufacturing, Cardiff: Springer Verlag, The Technologies & Applications of Rapid Prototyping & Rapid Tooling, p. 1ff..

[15] A. Acar, „Gründerszene,“ o. J.. [Online]. Available: https://www.gruenderszene.de/lexikon/begriffe/rapid-prototyping. [Zugriff am 23 08 2017].

[16] ETAS GmbH, „Software Engineering - Rapid Prototyping,“ 2017. [Online]. Available: https://www.etas.com/de/products/solutions_software_engineering-rapid_prototyping.php. [Zugriff am 23 08 2017].

[17] M. Kropatschek, A. Obert, R. Sattler und S. Stefan, „TU Wien - Softwareentwicklungsmodelle - Prototyping,“ 2002. [Online]. Available: http://cartoon.iguw.tuwien.ac.at/fit/fit01/prototyping/welcome.html. [Zugriff am 23 08 2017].

[18] R. Aachen, „Rapid Control Prototyping,“ o. J., [Online]. Available: http://www.irt.rwth-aachen.de/29/methoden/simulation-dynamischer-systeme-

rapid-control-prototyping/rapid-control-prototyping. [Zugriff am 15 08 2017].

[19] EVAS Softwarelösungen, „v-modell," o.J., [Online]. Available: https://www.evas.de/leistung/von-der-idee-zur-software/v-modell/. [Zugriff am 15 08 2017].

[20] E. Hull, K. Jackson und J. Dick, Requirements Engineering, London: Springer Verlag, 2005, p. 7 ff..

[21] H. Brandt-Pook und R. Kollmeier, Softwareentwicklung kompakt und verständlich, Wiesbaden: Vieweg + Teubner, 2008, p. 28 ff..

[22] Statista, „Fahrzeugabsatz von Porsche in den Jahren 2004/2005 bis 2016," 2017. [Online]. Available: https://de.statista.com/statistik/daten/studie/38219/umfrage/absatzentwicklung-von-porsche/. [Zugriff am 12 08 2017].

[23] M. Kropatschek, A. Obert, R. Sattler und S. Stefan, „TU Wien - Softwareentwicklungsmodelle - V Modell," 2002. [Online]. Available: http://cartoon.iguw.tuwien.ac.at/fit/fit01/spiral/konzepte.html#vmodell. [Zugriff am 23 08 2017].

Anhang

A1 Converter

```
classdef Converter1
    %CONVERTER
    %1. Reading all files from given folder path in PD5 and MDF Files
    %2. Checking if given data channels are present
    %3. Checking if file with the keyname contains given minimum number of
    %data points (keyname has to be set seperately (obj.keyname=name)after
    %construction otherwise channel with lowest number of datapoints is
    %taken
    %4. Extracting given data channels and meta data inf channel
    %5. Down Sampeling data channels and merging to DataFrame with
    %consistent dimensions of channels
    %6. Saving DataFrame as MatLab .mat File to a given folder using
    %original file name
    %-------------------------------------
    %Arguments: channels_mandatory- Needed DataChannels(FullName or RegEx)
    %           in a cell list {}
    %           channels- Optional DataChannels(FullName or RegEx) in a
    %           cell list {}
    %           minValueCount- Minimum number of datapoints in the channels
    %
    %Function Call:
    %load_folder_path- Full path name of the folder, which contains data to
    %                  be read in
    %save_folder_path- Full path name of the folder for saving the
    %                  converted files
    %tool_folder_path- Full path name of the folder, which contains the
    %                  code for all required components and classes
    %                  including this
    %Bsp:
    %   load_folder_path='C:\Users\prakti\Desktop\Benjamin\Daten\
    %                     messdaten_schwarzer911Turbo';
    %   save_folder_path='C:\Users\prakti\Desktop\Benjamin\Daten\
    %                     messdaten_schwarzer911Turbo';
    %   tool_folder_path='C:\Users\prakti\Documents\MATLAB\PoCTool';
    %   channels={'CAN3__PCM_04_PAG__GPS_N_Grad'};
    %   channels_mandatory = {'CAN3__PCM_04_PAG__GPS_E_Grad'};
    %   minValueCount=500;
    %   c=Converter(channels, channels_mandatory, minValueCount);
    %   c.convert(c,load_folder_path,save_folder_path,tool_folder_path);

    properties
        reader;
        downsampler;
        upsampler;
        channels;
        channels_mandatory;
        minValueCount=500;
        keyname='';
        start=1;
    end
```

```
    methods

        function c=Converter(channels,channels_mandatory,minValueCount,
keyChannelName)
            c.reader=PD5PIDMReader();
            c.downsampler=MeanMerger();
            c.upsampler=UpSampler();
            c.channels=channels;
            c.channels_mandatory = channels_mandatory;
            c.minValueCount=minValueCount;
            c.keyname=keyChannelName;
            c.start = 1;
        end

        function obj=set.keyname(obj,name)
            obj.keyname=name;
        end
    end

    methods(Static)

        function
res=convert(c,load_folder_path,save_folder_path,tool_folder_path)

            % Abspeichern der Dateien in der Variable listing
            cd(load_folder_path)
            listing=dir;
            res=0;

            % Iterieren über alle Dateien in dem Ordner
            for i=3:length(listing)

                % Laden des i-ten Files
                p=listing(i).name;
                display(['Load File: ' p]);

                % Prüfen ob File schon konvertiert wurde
                cd(save_folder_path);
                if (exist([save_folder_path '\' p '.mat'], 'file') == 0)

                    cd(tool_folder_path);

                    % Prüfen ob alle mandatory Channels in File enthalten
                    try
                        cm=([load_folder_path '\' p],c.channels_mandatory);
                        names = pd5_getChannelNames(cm);
                    catch
                        names='';
                    end

                    % Wenn mandatory Channels Ok, dann starten mit einlesen
                    % des Key Channels
                    if isempty(names)==0
                        display('Mandatory Channels OK')

                        ch_key=([load_folder_path '\' p],c.keyname);
                        dfl=c.reader.readIn(ch_key);
```

```
keyn=0;
for f=1:length(dfl)
    if ismember(c.keyname,fieldnames(dfl{f}.data))
        keyn=f;
        break;
    end
end

% Prüfen ob file genug Daten enthält
if dfl{keyn}.length>c.minValueCount
    display(dfl{keyn}.length);

    % Einlesen aller Channel Namen des Files
    ch=([load_folder_path '\' p], '*');
    all_channels = pd5_getChannelNames(ch);
    % Rauswerfen doppelter Channel Namen
    all_channels = unique(all_channels);

    % Channels bei Erstaufruf in Hilfsvariable
    % speichern
    % Grund: wird später verändert und soll bei
    % neuem File aber wieder Anfangswerte besitzen
    found = 0;
    if(c.start==1)
        channels_Selected = c.channels;
        c.start = 0;
    end

    c.channels = channels_Selected;

    % Prüfen, ob alle optional Channels in File
    % enthalten
    % Falls nicht enthalten, wird optional Channel
    % entfernt
    for i=1:length(c.channels)
        found = 0;
        for k=1:length(all_channels)
            %Search for Optional Channels
         if(isequal(c.channels(i),all_channels(k)))
            found = 1;
            break;
         end
        end
        % Statt Optional Channel NaN eintragen,
        % damit Channel im Anschluss gelöscht
        % werden kann
        if(found ~= 1)
            c.channels{i} = nan;
        end
    end

    % Löschen aller Cells in denen NaN steht
    c.channels(~cellfun(@ischar, c.channels))=[];

    % Zusammenfügen der Optional und Mandatory
    % Channels und Transponieren dieser
    c.channels = [c.channels,c.channels_mandatory];
    c.channels=c.channels';
```

```
                        % Einlesen aller gewünschten Channels
                        channels=([load_folder_path '\' p],c.channels);
                        dfl=c.reader.readIn(channels);

                        display('Converting File...');
                        display('Up Sampling...');
                        ndfl=c.upsampler.upsample(c.keyname,dfl);
                        display('Down Sampling...');
                        data=c.downsampler.merge(ndfl);
                        save([save_folder_path '\' p '.mat'],'data');
                        display('saved...');
                        res=1;
                        % konvertierte mdf Datei löschen zur
                        % Speicheroptimierung
                        % display('Deleting...');
                        % delete([load_folder_path '\' p]);
                    else
                        display(dfl{keyn}.length);
                        display('Key Channel has not enough data.');
                        display('Proceeding with next file.');
                        res=2;
                        % zu kleine Dateien löschen
                        % display('Deleting...');
                        % delete([load_folder_path '\' p]);
                    end
                else
                    display('Mandatory Channels not found in file.');
                    display('Proceeding with next file.');
                    res=3;
                    % Dateien ohne Mandatory Channels löschen
                    % display('Deleting...');
                    % delete([load_folder_path '\' p]);
                end
            else
                display('File is already converted.');
                display('Proceeding with next file.');
                res=4;
                % bereits konvertierte Dateien löschen
                % display('Deleting...');
                % delete([load_folder_path '\' p]);
            end
        end
    end
end
```

Listing 16: Code des Konverters

A2 OverCrossDetector

```
classdef CrossOverDetector
    %OVERCROSSDETECTOR Summary of this class goes here
    %   Detailed explanation goes here

    properties
    end

    methods

        function cod=CrossOverDetector()
        end

    end
    methods(Static)

            function cohs=detectCrossOver(all_data,merged_hs, idxLS)
            global d_GPS d_dir visualize_deleted_drivethroughs

            ihs=length(merged_hs);
            numberOfHS=length(merged_hs);

            while(ihs>0)

                hs=merged_hs(ihs);

                %all idx of datapoints of one drive in one hs
                oc_all=0;
                oc_cond=0;
                durchfahrt=[];
                predict=[];
                actual=[];
                durchfahrtszeitpunkt=[];
                temp_idx_cond=[];

                for d=1:length(all_data)

                    dps=all_data(d);

                    %% Get all indizes of drivethroughs
                    minima=[];
                    startpunkt=[];
                    endpunkt=[];

                    temp_durchfahrt=[];

                    % Abstand zwischen 3 kontinuierlichen Punkten zum
                    % Hotspot berechnen
                    for k=2:length(dps.long)-1

                      [~,dist]= geodist(dps.lat(k-1), dps.long(k-1),
                                hs.latcentercenter, hs.longcentercenter);
                      [~,dist2]= geodist(dps.lat(k), dps.long(k),
                                 hs.latcentercenter, hs.longcentercenter);
                      [~,dist3]= geodist(dps.lat(k+1), dps.long(k+1),
                                 hs.latcentercenter, hs.longcentercenter);
```

```
    % Wenn mittlerer Punkt kleiner als die Äußeren
    % --> lokales Minimum
    if(dist >= dist2 && dist3 >= dist2)
        minima = [minima k];
    end
end

% Suche alle Minima die innerhalb des Geofences liegen
dmin=(dps.lat(minima)>=hs.latcentercenter-d_GPS & ...
    dps.long(minima)>=hs.longcentercenter-d_GPS  & ...
    dps.lat(minima)<=hs.latcentercenter+d_GPS  & ...
    dps.long(minima)<=hs.longcentercenter+d_GPS);

% Lösche alle Minima, die außerhalb des Geofences
% liegen
minima=minima(dmin>0);

if(isempty(minima)==0)
    % Finde alle Start- und Endpunkte von weiteren
    % Durchfahrten
    k=1;
    m=length(minima); % Schleifenvariable

    while(m>0)
        % Finde den letzten Punkt vor dem ersten
        % Minima, der außerhalb des Geofences liegt
        % --> Startpunkt der Fahrt
        startpunkt{k}=find(dps.lat(1:(minima(k))) <
                 hs.latcentercenter-d_GPS | ...
            dps.lat(1:(minima(k))) >
                 hs.latcentercenter+d_GPS | ...
            dps.long(1:(minima(k))) <
                 hs.longcentercenter-d_GPS | ...
            dps.long(1:(minima(k))) >
                 hs.longcentercenter+d_GPS, 1, 'last');

        if(isempty(startpunkt{k}))
            startpunkt{k}=1;
        end

        % Finde den ersten Punkt nach dem Minimum,
        % der außerhalb des Geofences liegt -->
        % Endpunkt der Fahrt
        endpunkt{k}=find(dps.lat(minima(k):end) <
                hs.latcentercenter-d_GPS | ...
            dps.lat(minima(k):end) >
                hs.latcentercenter+d_GPS | ...
            dps.long(minima(k):end) <
                hs.longcentercenter-d_GPS | ...
            dps.long(minima(k):end) >
                hs.longcentercenter+d_GPS, 1, 'first');

        endpunkt{k} = endpunkt{k}+minima(k)-1;

        if(isempty(endpunkt{k}))
            endpunkt{k}=length(dps.lat);
        end
```

```
        % Wenn mehrere Minima innerhalb einer
        % Durchfahrt
        % gefunden werden, wird nur Minima, welches
        % näher an HS ist gespeichert

        % alle Minima einer Durchfahrt finden
        temp_idx=(minima(k:end)-endpunkt{k}<0);
        for i=1:k-1
            temp_idx=[0 temp_idx];
        end
        temp1=minima(temp_idx>0);
        % Minima mit kleinstem Abstand zu HS finden
        [~,dist]= geodist(dps.lat(temp1),
         dps.long(temp1), hs.latcenter, hs.longcenter);
        temp2=(find(dist==min(dist),1,'last'));
        temp2=minima(temp2+k-1);
        % Weiterentfernte Minima löschen
        temp=(temp1~=temp2);
        for i=1:k-1
            temp=[0 temp];
        end
        minima(temp>0)=[];

        % Schleifenvariable um Anzahl der
        % weiterentfernten Minima reduzieren
        m=m-(length(temp1)-1);

        % Dekrementieren der Schleifenvariable
        m=m-1;
        % Inkrementieren der ZÃ¤hlvariable der Start-
        % und Endpunkte
        k=k+1;

    end

% Array der Durchfahrten bilden
for k=1:length(startpunkt)
 durchfahrt{k,d}=((startpunkt{k}+1):(endpunkt{k}-1));
end

    % Ermittlung des Durchfahrtzeitpunktes
    k=length(durchfahrt(:,d));
    temp1=[];
    while(k>0)
        dist=[];
        if(isempty(durchfahrt{k,d})==0)
           tmp=durchfahrt{k,d};
           % Suchen des Punktes mit dem kleinsten
           % Abstand zum Hotspot
           [~,dist]= geodist(dps.lat(tmp),
                        dps.long(tmp), hs.latcenter,
                        hs.longcenter);
           idx_min_dist=
                      (find(dist==min(dist),1,'last'));
           temp1=[durchfahrt{k,d}(idx_min_dist) temp1];
```

```
            % Wenn kleinster Abstand>30m oder
            % (Durchschnitt der Richtungen einer
            % Durchfahrt und Richtung des
            % Durchfahrtszeitpunktes) nicht innerhalb
            % der erlaubten Abweichung zur Hotspot
            % Richtung, dann Durchfahrt löschen

            min_dist=dist(idx_min_dist);
            dir=mod(dps.dir(tmp)-hs.dircenter,360);
            dir1=mod(hs.dircenter-dps.dir(tmp),360);
            dir2=mod(dps.dir(temp1(i))-hs.dircenter,360);
            dir3=mod(hs.dircenter-dps.dir(temp1(i)),360);
            if(min_dist>30 ...
               || (mean(min(dir,dir1))>(1.5*d_dir)...
               && min(dir2,dir3)>(d_dir)))

                  startpunkt(k)=[];
                  temp1(i)=[];
                  durchfahrt{k,d}=[];
                  endpunkt(k)=[];

               else

                  i=i+1;

               end
            end
            k=k-1;
        end
        % Durchfahrtszeitpunkte speichern
        durchfahrtszeitpunkt{d}=temp1;

        % Herausfinden der Durchfahrten mit Umschaltpunkt
        temp1=[];
        for k=1:length(durchfahrt(:,d))
            cnt=0;
            temp=durchfahrt{k,d};
            if(isempty(temp)==0)
                for i=2:length(temp)
                    % Umschaltszeitpunkte finden
                    F=dps.Fahrprogramm(temp(i-1));
                    F1=dps.Fahrprogramm(temp(i));
                    if(F~=F1)
                        temp1 = [temp1 (i-1)];
                        cnt=cnt+1;
                    end
                end
            end

            % Wenn mehrere Umschaltungen pro Durchfahrt:
            % Reduzieren von Zählvariable für Anzahl
            % der Durchfahrten mit Umschaltung
            if(cnt>1)
                oc_cond = oc_cond -(cnt-1);
            end
        end
        % Umschaltszeitpunkte speichern
        temp_idx_cond{d} = temp1;
```

```
                        % Berechnung der Anzahl der Durchfahrten ohne und
                        % mit Umschaltungen
                        oc_all = oc_all+length(temp_durchfahrt);
                        oc_cond = oc_cond+length(temp_idx_cond{d});

                        % Abspeichern des Predict und Actual Werte pro
                        % Datei und Durchfahrt
                        predict{1,d}=[];
                        actual{1,d}=[];
                        for i=1:length(temp_durchfahrt)
                            dt=temp_durchfahrt{i};
                            predict{i,d}=dps.Predicted_Mode(dt);
                            actual{i,d}=dps.Fahrprogramm(dt);
                        end
                    else
                        % Festlegen der Werte falls es kein Minimum gibt
                        % in Datei
                        durchfahrt{1,d}=[];
                        durchfahrtszeitpunkt{d}=[];
                        temp_idx_cond{d} = [];

                        predict{1,d}=[];
                        actual{1,d}=[];

                        oc_all = oc_all+0;
                        oc_cond = oc_cond+0;
                    end
                end

                % Abspeichern der Werte in den Hotspots
                % Zuweisen der Anzahl der Durchfahrten
                merged_hs(ihs).conddt=oc_cond;
                merged_hs(ihs).totaldt=oc_all;

                % Zuweisen der Indizes der Durchfahrten
                merged_hs(ihs).idxCond = temp_idx_cond;
                merged_hs(ihs).idxDT = durchfahrtszeitpunkt;
                merged_hs(ihs).idx = durchfahrt;

                % Zuweisen von predict und actual Werten an Hotspot
                merged_hs(ihs).predict = predict;
                merged_hs(ihs).real = actual;

                % Display Fortschritt der Funktion
                disp([' ' num2str((1-(ihs/numberOfHS))*100, '%.f') '%']);
                ihs=ihs-1;

            end

            disp(' 100%');

            % Rückgabe Parameter übergeben
            cohs=merged_hs;
        end
    end
end
```

Listing 17: Code der Durchfahrtserkennung

A3 Gütemaß

```
classdef PrecisionAndRecall
    %Precision And Recall

    properties
        tp;
        fp;
        tn;
        fn;
        precision;
        recall;
        fscore;
    end

    methods

        function c = PrecisionAndRecall(tp, fp, tn, fn)
            c.tp=tp;
            c.fp=fp;
            c.tn=tn;
            c.fn=fn;
        end

    end

    methods(Static)

        function c = calculatePrecision(c)
            display(' ');
            display(['calculate Precision...']);
            try
                tp=cell2mat(c.tp);
                fp=cell2mat(c.fp);
            catch
                tp=c.tp;
                fp=c.fp;
            end
            try
                c.precision = tp./(tp+fp);
                display(['Precision = ' num2str(c.precision)]);
                display('Calculation finished.');
                display(' ');
            catch
                display('Calculating failed.');
                display(' ');
            end
            %c.precision(isnan(c.precision))=0;
        end
```

```
        function c = calculateRecall(c)
            display(['calculate Recall...']);
            try
                tp=cell2mat(c.tp);
                fn=cell2mat(c.fn);
            catch
                tp=c.tp;
                fn=c.fn;
            end
            try
                c.recall = tp./(tp+fn);
                display(['Recall = ' num2str(c.recall)]);
                display('Calculation finished.');
                display(' ');
            catch
                display('Calculation failed.');
                display(' ');
            end
            %c.recall(isnan(c.recall)) = 0;
        end

        function c = calculateFScore(c,beta)
            display(['calculate FScore']);
            if nargin < 1
                display('Error, not possible to calculate F Score.');
                display(' ');
                error(['Too little input arguments.']);
                display(' ');
                display(' ');
            else
                if nargin < 2
                    try
                        beta = 1;
                    catch
                        display('Calculation failed.');
                        display(' ');
                        display(' ');
                    end
                end
                    try
                        z = (1+beta.^2)*(c.precision.*c.recall);
                        n = ((beta.^2.*c.precision)+c.recall);
                        c.fscore = z./n;
                        display(['F Score = ' num2str(c.fscore)]);
                        display('Calculation finished');
                        display(' ');
                        display(' ');
                    catch
                        display('Calculation failed.');
                        display(' ');
                        display(' ');
                    end
                    %c.fscore(isnan(c.fscore)) = 0;
            end
        end
    end
end
```

Listing 18: Code der Gütemaß Berechnung