

# **Entwicklung eines Abstandsmessers zur Ermittlung von Überholabständen von Kraftfahrzeugen gegenüber Fahrradfahrern**

**Studienarbeit  
des Studiengangs Elektrotechnik  
an der Dualen Hochschule Baden Württemberg Stuttgart**

Jakob Seitzer

Juni 2020

|                      |                              |
|----------------------|------------------------------|
| Bearbeitungszeitraum | Oktober 2019 - Juni 2020     |
| Matrikelnummer, Kurs | 2271407, TEL17GR5            |
| Betreuer             | Prof. Dr.-Ing. Harald Mandel |

## **Ehrenwörtliche Erklärung**

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: „Entwicklung eines Abstandsmessers zur Ermittlung von Überholabständen von Kraftfahrzeugen gegenüber Fahrradfahrern“ 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)

## Zusammenfassung

Die Studienarbeit befasst sich mit der Entwicklung eines Messsystems zur Ermittlung der Überholabstände von Kraftfahrzeugen gegenüber Fahrradfahrern. Zu Beginn werden aktuell bestehende Lösungsansätze betrachtet. Von besonderem Interesse ist in diesem Zusammenhang der Radmesser, welcher durch den Berliner Tagesspiegel entwickelt wurde.

Verschiedene Varianten der Hardwarekomponenten werden diskutiert, wobei als Recheneinheit der Raspberry Pi Zero dient, der Abstand durch die Ultraschallsensoren HC-SR04 bestimmt wird, die Temperatur mit Hilfe des Sensors DS18B20 ermittelt wird und die Zeit und Position durch das GPS-Modul GY-GPS6MV2 erfasst wird. Mit den ermittelten Hardwarekomponenten wird ein Schaltplan entworfen, welcher dann in eine Leiterkarte übergeht.

Danach wird eine auf die Hardware zugeschnittene Software in der Programmiersprache Python 3 entworfen, mit welcher der Abstand zu Objekten auf der linken Seite des Fahrrads dokumentiert wird. Für die aufgezeichneten Daten wird eine Auswertungssoftware entwickelt. Diese extrahiert aus den gemessenen Abstandsdaten Überholvorgänge und bestimmt die Geschwindigkeit und Länge des überholenden Kraftfahrzeuges.

Anschließend werden die notwendigen Hardware- und Softwaretests erstellt, um die Funktionsfähigkeit des Messsystems zu belegen. Ein großer Teil der Tests entfällt auf die Ultraschallsensoren HC-SR04, welche maßgeblich für die Messgenauigkeit verantwortlich sind. Abschließend wird festgehalten, dass die entwickelte Lösung den bisher bestehenden Lösungsansätzen bezüglich Akkulaufzeit, Funktionalität und Kosten überlegen ist. Zudem wird ein mögliches weiteres Vorgehen skizziert.

## Vorwort

Die Studienarbeit entstand im Rahmen des *Urban Mobility Labs*, welches durch den „Fonds Erfolgreich Studieren in Baden-Württemberg (FEST-BW)“ des Ministeriums für Wissenschaft, Forschung und Kunst Baden-Württemberg gefördert wird.

Im Laufe der Erstellung der Arbeit wurde mir im Gespräch mit vielen Personen bewusst, für wie viele Fahrradfahrer das Thema Sicherheit im Straßenverkehr eine große Bedeutung hat.

Während ich in meiner Freizeit viel mit dem Fahrrad auf Fahrradwegen unterwegs bin, lässt es sich nie gänzlich vermeiden auch Mal eine Straße zu benutzen. Die Überholmanöver, welche ich dort bereits erlebt habe, suchen Ihresgleichen - und finden sie auch. So schilderte mir eine Freundin, dass sie morgens auf dem Weg zur Uni von einem Auto überholt wurde und der Fahrer während des Überholvorgangs aufgrund einer Engstelle nach rechts ausgewichen ist und sie so über den Gehsteig gestürzt ist. Ein ehemaliger Klassenkamerad berichtete mir von einem Überholvorgang, bei welchem der Außenspiegels eines Autos einklappte, nachdem er ihn mit dem Unterarm touchiert hatte.

Am 28.04.2020 traten Änderungen der Straßenverkehrsordnung in Kraft, welche einen Mindestabstand zum Überholen von Fahrradfahrern vorschreiben. Mein aktueller Eindruck ist jedoch, dass dies nichts am Fahrverhalten der Autofahrer geändert hat.

Letztendlich ist es auch mit die Aufgabe von Stadtplanern und Verantwortlichen der Stadtentwicklung allen Verkehrsteilnehmern ausreichend Platz zur Verfügung zu stellen und ausreichend Sicherheit zu garantieren. Sie müssen besonders gefährliche Stellen (für Fahrradfahrer) erkennen, um Umgestaltungsmöglichkeiten entwickeln, beschließen und umsetzen zu können.

# Inhaltsverzeichnis

|  |           |
|--|-----------|
| Ehrenwörtliche Erklärung . . . . .                             | I         |
| Zusammenfassung . . . . .                                      | II        |
| Vorwort . . . . .  | III       |
| Abkürzungsverzeichnis und Konventionen . . . . .               | VIII      |
| <b>1 Einführung und Motivation</b>                             | <b>1</b>  |
| 1.1 Motivation . . . . .                                       | 1         |
| 1.2 Seitliche Überholabstände . . . . .                        | 2         |
| 1.2.1 Rechtliche Grundlagen . . . . .                          | 2         |
| 1.2.2 Fahrverhalten von Fahrradfahrern . . . . .               | 3         |
| 1.2.3 Seitliche Überholabstände in der Praxis . . . . .        | 4         |
| 1.3 Abgrenzung der Aufgabenstellung . . . . .                  | 6         |
| <b>2 Stand der Technik</b>                                     | <b>7</b>  |
| 2.1 Tagesspiegel Berlin - Radmesser . . . . .                  | 7         |
| 2.2 Flipdot - wheelknife . . . . .                             | 8         |
| 2.3 Zweirat Stuttgart RadmesserS und OpenBikeSensor . . . . .  | 8         |
| 2.4 Codaxus LLC C3FT v3 . . . . .                              | 9         |
| 2.5 Analyse bestehender Lösungsansätze . . . . .               | 9         |
| <b>3 Zusammenstellung der Hardwarekomponenten</b>              | <b>11</b> |
| 3.1 Controller oder Computer . . . . .                         | 11        |
| 3.2 Ultraschallsensoren . . . . .                              | 12        |
| 3.3 Temperatursensor . . . . .                                 | 13        |
| 3.4 GPS-Modul . . . . .  | 14        |
| 3.5 Entwicklung der Platine . . . . .                          | 14        |
| 3.5.1 Berechnung des Winkels der Ultraschallsensoren . . . . . | 15        |
| 3.5.2 Schaltplanentwicklung . . . . .                          | 17        |
| 3.5.3 Layout der Platine . . . . .                             | 18        |
| 3.6 Gehäuse . . . . .  | 18        |
| <b>4 Entwicklung der Radmesser-Software</b>                    | <b>20</b> |
| 4.1 Softwarekonzept . . . . .                                  | 20        |

|          |  |           |
|----------|--|-----------|
| 4.1.1    | Parallele Sensorauswertung . . . . .                             | 21        |
| 4.1.2    | Erfassen von Zeit und Position . . . . .                         | 21        |
| 4.1.3    | Temperatursensor . . . . .                                       | 22        |
| 4.1.4    | Anzeigen des Betriebszustandes . . . . .                         | 22        |
| 4.2      | Dateiformat . . . . .  | 23        |
| 4.3      | Ausfallsicherheit . . . . .                                      | 25        |
| 4.4      | Realisierung der Software . . . . .                              | 25        |
| <b>5</b> | <b>Entwicklung der Auswertung-Software</b>                       | <b>27</b> |
| 5.1      | Eingangs-Datenbearbeitung . . . . .                              | 27        |
| 5.2      | Erkennen eines Überholvorgangs . . . . .                         | 29        |
| 5.2.1    | Bestimmung der Geschwindigkeit des Kraftfahrzeug (KFZ) . . . . . | 31        |
| 5.2.2    | Unterscheidung der Fahrzeugarten . . . . .                       | 31        |
| 5.3      | Berechnung der zurückgelegten Strecke . . . . .                  | 32        |
| 5.4      | Auswertung und Visualisierung der Daten . . . . .                | 33        |
| 5.4.1    | Dateiformate zur Darstellung von GPS-Koordinaten . . . . .       | 33        |
| 5.4.2    | Analyse der besonders betroffenen Gebiete . . . . .              | 34        |
| 5.4.3    | Auswertung nach weiteren Parametern . . . . .                    | 35        |
| 5.4.4    | Weitere Auswertungsmöglichkeiten der GPS-Daten . . . . .         | 36        |
| <b>6</b> | <b>Tests</b>   | <b>37</b> |
| 6.1      | Radmesserhardware . . . . .                                      | 37        |
| 6.1.1    | Messwerte bei variabler Luftfeuchte . . . . .                    | 37        |
| 6.1.2    | Messwerte bei variabler Temperatur . . . . .                     | 40        |
| 6.1.3    | Messwerte bei externer Schalleinstreuung . . . . .               | 41        |
| 6.1.4    | Messwerte bei bewegten Luftmassen . . . . .                      | 43        |
| 6.1.5    | Messbereich der Ultraschallsensoren . . . . .                    | 44        |
| 6.1.6    | Weitere Tests der Ultraschallsensoren . . . . .                  | 47        |
| 6.2      | Radmessersoftware . . . . .                                      | 48        |
| 6.2.1    | Langzeittest . . . . .   | 48        |
| 6.2.2    | Fehlende GPS-Daten . . . . .                                     | 48        |
| 6.2.3    | Abgedeckter Ultraschallsensor . . . . .                          | 49        |
| 6.2.4    | Unabhängige Funktion der Ultraschallsensoren . . . . .           | 49        |
| 6.3      | Auswertungssoftware . . . . .                                    | 50        |
| 6.3.1    | Erkennen von Überholvorgängen . . . . .                          | 50        |
| 6.3.2    | Erkennen verschiedener Fahrten . . . . .                         | 51        |
| 6.4      | Ergebnis der Tests . . . . .                                     | 51        |
| <b>7</b> | <b>Fazit</b>   | <b>53</b> |

|                    |           |
|--------------------|-----------|
| <b>8 Literatur</b> | <b>55</b> |
|--------------------|-----------|

|   |          |
|---|----------|
| <b>9 Anhang</b>   | <b>i</b> |
| 9.1 Abmessungen von KFZ . . . . .                         | i        |
| 9.2 Definition eines Überholvorgangs . . . . .            | i        |
| 9.2.1 Stadtverkehr . . . . .                              | i        |
| 9.2.2 Außerhalb geschlossener Ortschaften . . . . .       | iii      |
| 9.2.3 Zusammenfassung der Überholzeiten . . . . .         | v        |
| 9.3 Erläuterung der in Kaufnahme von Bußgeldern . . . . . | v        |
| 9.3.1 Geschwindigkeitsüberschreitungen . . . . .          | v        |
| 9.4 Erläuterung der Straßenbreite . . . . .               | vii      |
| 9.5 Quellcode Radmesser Python . . . . .                  | ix       |
| 9.6 Inbetriebnahme Radmesser . . . . .                    | xvi      |
| 9.6.1 Materialliste . . . . .                             | xvi      |
| 9.6.2 Bestückung der Platine . . . . .                    | xix      |
| 9.6.3 SD-Karte vorbereiten . . . . .                      | xix      |
| 9.6.4 Funktionsnachweis Radmesser . . . . .               | xix      |
| 9.6.5 Handhabung im täglichen Einsatz . . . . .           | xxii     |
| 9.7 Quellcode Auswertungssoftware D . . . . .             | xxiii    |
| 9.8 Adressabfrage in R . . . . .                          | xlvi     |
| 9.9 Zusätzliche Diagramme . . . . .                       | xliv     |

# Abbildungsverzeichnis

|         |   |      |
|---------|---|------|
| Abb. 1  | Seitlicher Abstand beim Überholen . . . . .   | 2    |
| Abb. 2  | Vergleich Radfahrstreifen und Schutzstreifen . . . . .  | 4    |
| Abb. 3  | Spannungsteiler für Ultraschallsensoren . . . . .   | 13   |
| Abb. 4  | Prinzipskizze der Ultraschallsensoren . . . . .   | 15   |
| Abb. 5  | Beispielmesswerte nach Prinzipskizze 4 . . . . .  | 16   |
| Abb. 6  | Prototyp des Radmessers . . . . .   | 19   |
| Abb. 7  | Graphische Darstellung des Glättungsalgorithmus . . . . .   | 29   |
| Abb. 8  | Testaufbau Radmesserhardware . . . . .  | 37   |
| Abb. 9  | Temperaturabhängigkeit der Schallgeschwindigkeit . . . . .  | 41   |
| Abb. 10 | Genauigkeit der Schallgeschwindigkeitsmessung . . . . .   | 41   |
| Abb. 11 | Einfluss von externer Schalleinstreuung auf die Präzision von Messungen mit Ultraschallsensoren . . . . . | 42   |
| Abb. 12 | Einfluss von Wind auf die Präzision von Messungen mit Ultraschallsensoren . . . . .                       | 43   |
| Abb. 13 | Messbereich der Ultraschallsensoren . . . . .   | 45   |
| Abb. 14 | Messgenauigkeit bei schrägen Flächen . . . . .  | 45   |
| Abb. 15 | Messbereich der Ultraschallsensoren . . . . .   | 46   |
| Abb. 16 | Ultraschallsensoren bei schrägen Flächen . . . . .  | 47   |
| Abb. 17 | Überlrvorgänge auf einer Karte dargestellt . . . . .  | 51   |
| Abb. 18 | Benötigtes Material . . . . .   | xvi  |
| Abb. 19 | Leiterplatten Layout . . . . .  | xvii |
| Abb. 20 | Messbereich der Ultraschallsensoren . . . . .   | xliv |
| Abb. 21 | Messbereich der Ultraschallsensoren . . . . .   | xliv |
| Abb. 22 | Ultraschallsensoren bei schrägen Flächen . . . . .  | xliv |

# Abkürzungsverzeichnis und Konventionen

|                       |   |
|-----------------------|---|
| <b>ADAC</b>           | Allgemeiner Deutsche Automobil Club           |
| <b>ADFC</b>           | Allgemeiner Deutscher Fahrrad Club            |
| <b>AD-Wandler</b>     | Analog-Digital-Wandler                        |
| <b>API</b>            | Application Programming Interface             |
| <b>BTK</b>            | Bundeseinheitlicher Tatbestandskatalog        |
| <b>EKL</b>            | Entwurfsklassen (bei Landstraßen)             |
| <b>GeoJSON</b>        | Geo-JavaScript Object Notation                |
| <b>GML</b>            | Geography Markup Language                     |
| <b>GND</b>            | Ground  |
| <b>GPCLK</b>          | General Purpose Clock                         |
| <b>GPIO</b>           | General Purpose Input Output                  |
| <b>GPRMC</b>          | RECOMMENDED MINIMUM SPECIFIC GPS/TRANSIT DATA |
| <b>GPX</b>            | GPS Exchange Format                           |
| <b>JSON</b>           | JavaScript-Object-Notation                    |
| <b>KFZ</b>            | Kraftfahrzeug                                 |
| <b>KML</b>            | Keyhole Markup Language                       |
| <b>LKW</b>            | Lastkraftwagen                                |
| <b>NMEA</b>           | National Marine Electronics Association       |
| <b>ÖPNV</b>           | Öffentlicher Personen Nahverkehr              |
| <b>PKW</b>            | Personenkraftwagen                            |
| <b>RAL</b>            | Richtlinien für die Anlage von Landstraßen    |
| <b>RASt</b>           | Richtlinien für die Anlage von Stadtstraßen   |
| <b>Schwerfahzeuge</b> | Busse und Lastkraftwagen (LKW)                |
| <b>SMD</b>            | Surface mounted devices                       |
| <b>StVO</b>           | Straßenverkehrsordnung                        |
| <b>TBNR</b>           | Tatbestands Nummer                            |
| <b>UART</b>           | Universal Asynchronous Receiver Transmitter   |
| <b>UTC</b>            | Coordinated Universal Time                    |

|                         |                            |                         |
|-------------------------|----------------------------|-------------------------|
| <b>XML</b>              | Extensible Markup Language |                         |
| $\alpha, \beta, \gamma$ | [°]                        | Winkel                  |
| $c$                     | [m s <sup>-1</sup> ]       | Schallgeschwindigkeit   |
| $\kappa$                | []                         | Adiabatenexponent       |
| $l$                     | [m]                        | Länge                   |
| $m$                     | [kg]                       | Masse                   |
| $N$                     | []                         | Anzahl                  |
| $R$                     | [Nm/kmol · K]              | Gaskonstante            |
| $R$                     | [\Omega]                   | elektrischer Widerstand |
| $r$                     | []                         | Volumenanteil           |
| $\rho$                  | [kg m <sup>-3</sup> ]      | Dichte                  |
| $t$                     | [s]                        | Zeit in Sekunden        |
| $T$                     | [K]                        | Temperatur              |
| $U$                     | [V]                        | Spannung                |
| $v$                     | [m s <sup>-1</sup> , km/h] | Geschwindigkeit         |
| $V$                     | [m <sup>3</sup> ]          | Volumen                 |
| $w$                     | []                         | Faktor                  |
| $y$                     | []                         | Massenanteil            |

# 1 Einführung und Motivation

## 1.1 Motivation

Während des Erstellungszeitraumes dieser Studienarbeit fand und findet in vielen Lebensbereichen ein radikaler Wandel durch die SARS-CoV-2-Pandemie statt, welcher auch stark das Mobilitätsverhalten der Menschen beeinflusst hat und es immer noch tut. Es ist zu beobachten, dass bei fast allen Personen weniger Mobilität erforderlich ist [1] und öffentliche Verkehrsmittel aufgrund des höheren Infektionsrisikos stark an Popularität einbüßen [2]. Dafür gewinnt das Auto am meisten an Beliebtheit dazu [2] während das Fahrrad jedoch den stärksten Nutzerzuwachs erfährt [1]. Um diesen geänderten Mobilitätserfordernissen gerecht zu werden, errichten viele Städte eine sogenannte Pop-up-Infrastruktur an Fahrradstreifen [3]. Eine Bewertung der Fahrradinfrastruktur aus gelben Linien, Piktogrammen auf den Straßen und Warnbaken steht noch aus. Es ist weiterhin fraglich, wie mit der aktuell entstehenden Pop-up-Infrastruktur nach der SARS-CoV-2-Pandemie verfahren wird [5].

Durch das Vermeiden öffentlicher Verkehrsmittel steigt das Aufkommen an Individualverkehr stark an, dies führt besonders in großen Städten zu langen Staus und schlechter Luft. Um einen Verkehrskollaps abzuwenden, wird das Fahrrad als Verkehrsmittel in den Vordergrund gerückt [4].

Eine langfristige Änderung des Mobilitätsverhaltens ist erforderlich, um die Ziele des Pariser Klimaschutzabkommens umzusetzen, welche in Deutschland im *Klimaschutzplan 2050* fixiert sind [6]. Die Attraktivität des Radverkehrs ist besonders in Städten in den letzten Jahren gestiegen. Um diesen Trend fortzuführen, muss das Fahrradfahren durch eine verbesserte Infrastruktur sicher gestaltet werden [7]. Besonders in Städten wie Stuttgart, welche unter einer starken Feinstaubbelastung leiden ist eine Änderung des Mobilitätsverhaltens alternativlos. Dies beinhaltet sowohl die vermehrte Nutzung des Öffentlichen Personen Nahverkehr (ÖPNV), das Verwenden von Elektroantrieben in KFZ sowie den Umstieg auf das Fahrrad. Um den zuletzt genannten Aspekt zu fördern, muss die Sicherheit für Fahrradfahrer im Straßenverkehr gesteigert werden. In diesem Zusammenhang

müssen zuerst gefährliche Stellen für Fahrradfahrer erkannt werden - erst dann können Gegenmaßnahmen ergriffen werden. In diesem Zusammenhang sollen die seitlichen Überholabstände von KFZ gegenüber Fahrradfahrern ermittelt werden, und anhand der Koordinaten Gefahrenstellen durch die Verkehrsinfrastruktur erkannt werden.

## 1.2 Seitliche Überholabstände

Als seitlicher Überholabstand wird der Abstand zwischen den äußersten Fahrzeuggrenzen definiert, in unserem Fall heißt dies, dass der seitliche Überholabstand zwischen Fahrradlenker und Außenspiegel zu messen ist. In Abb. 1 ist ein seitlicher Abstand von 50 cm zu sehen.



Abbildung 1: Seitlicher Abstand beim Überholen, zu messen zwischen Fahrradlenker und Außenspiegel

### 1.2.1 Rechtliche Grundlagen

**Stand bis 27.04.2020** Der seitliche Abstand, welcher eingehalten werden muss, wenn ein Fahrradfahrer durch ein KFZ überholt wird, ist in der Straßenverkehrsordnung (StVO) nicht eindeutig festgelegt. Die StVO legt lediglich fest, dass andere Verkehrsteilnehmer nicht gefährdet werden dürfen und der Seitenabstand beim Überholen ausreichend sein muss. [8]

Aufgrund von Gerichtsurteilen wird unter normalen Umständen ein Seitenabstand von 1,5 m als ausreichend angesehen. Liegen besondere Umstände vor oder ist davon auszugehen, dass der Fahrradfahrer zu Ausweichmanövern gezwungen ist, sollte ein Seitenabstand

von 2 m eingehalten werden [8], [9]. Diese Ansicht vertritt auch der Allgemeine Deutsche Automobil Club (ADAC), welcher typischer Weise eher als Lobby der Autofahrer auftritt [10]. Im Folgenden soll erläutert werden, warum ein ausreichender seitlicher Abstand zu Fahrradfahrern notwendig ist und wie dieser unter äußeren Einflüssen variieren kann.

**Stand ab 28.04.2020** Am 28.04.2020 trat eine überarbeitete Version der StVO in Kraft. Die für die Studienarbeit relevante Änderung ist in StVO § 5 Abs. 4 S. 2 zu finden. Hier wird der Satz: „Beim Überholen mit Kraftfahrzeugen von zu Fuß Gehenden, Rad Fahren- den und Elektrokleinstfahrzeug Führenden beträgt der ausreichende Seitenabstand innerorts mindestens 1,5 m und außerorts mindestens 2 m.“ eingefügt, welcher die bisherigen Unklarheiten bezüglich eines ausreichenden Überholabstands eindeutig beseitigt.

### 1.2.2 Fahrverhalten von Fahrradfahrern

Ein Fahrrad im Stand aufrecht zu halten ist nicht möglich. Die zwei Standpunkte des Fahrrads (Vorder- und Hinterrad) reichen nicht aus, um das Fahrrad stabil stehen zu lassen. Während des Fahrens ( $v \neq 0$ ) ändern sich jedoch die physikalischen Rahmenbedingungen. Als Beitrag zum Gleichgewicht kommt neben dem Lenkereinschlag und dem Kippwinkel aus der Senkrechten noch die an die Rollrichtung der Räder gebundene Bewegung hinzu. Ein Kippen aus der Senkrechten heraus kann durch eine Gewichtsverlagerung des Fahrradfahrers erfolgen oder durch Lenkereinschlag in Kipprichtung. Da der Einfluss des Lenkereinschlages quadratisch mit der Geschwindigkeit der Vorwärtsbewegung des Fahrrades ansteigt, [11], [12] ist es einfacher ein schnell fahrendes Fahrrad aufrecht zu halten, als ein langsam fahrendes.

Eine genaue physikalische Beschreibung des Gleichgewichts beim Fahrradfahren würde den Rahmen dieser Studienarbeit übersteigen. Es wird auf Suhr und Schlichting (2007) [12] und Saße (2007) [11] verwiesen.

Für unsere Zwecke reicht es aus fest zu halten, dass bei geringen Geschwindigkeiten, verkehrsbedingt (Anfahren oder Bergfahrt) oder durch den Fahrradfahrer bedingt (Kind o.ä.) die notwendigen Lenkereinschläge und damit verbunden die seitlichen Abweichungen von der mittleren Fahrlinie größer sind als bei höheren Geschwindigkeiten. Zusätzlich kann durch seitlichen Windböen der Kippwinkel aus der Senkrechten plötzlich geändert werden und eine Abweichung von der mittleren Fahrlinie verursachen. Luftbewegungen werden auch durch vorbeifahrende KFZ erzeugt, wobei die Intensität der Luftbewegung mit geringerem Abstand und höherer Geschwindigkeit zu nimmt [8].

### 1.2.3 Seitliche Überholabstände in der Praxis

Studien über das Thema Überholabstände von KFZ gegenüber Fahrradfahrern gibt es bereits. Diese erfassen die Verkehrssituationen jedoch nur stichpunktartig und nicht flächendeckend. Im Folgenden werden die drei zentralen Studien beschrieben, welche im Dezember 2019 zum Thema Überholabstände von KFZ gegenüber Fahrradfahrern existieren.

#### Gesamtverband der Deutschen Versicherungswirtschaft e.V.

Durch den Gesamtverband der Deutschen Versicherungswirtschaft e. V. wurde eine Forschungsstudie über die Sicherheit und Nutzbarkeit markierter Radverkehrsführung in Auftrag gegeben [13]. Zusammengefasst und bewertet wurde die Studie in der Broschüre Unfallforschung Nr. 89 [14]. Hierfür wurden Fahrradfahrer befragt, Unfälle analysiert und an bestimmten Stellen gezielt das Verhalten von Fahrradfahrern und KFZ beobachtet. Zentrales Objekt der Verkehrsbeobachtung waren Radfahrstreifen und Schutzstreifen (Vgl. Abb. 2), nicht jedoch Straßenabschnitte ohne markierte Radverkehrsführung. Insgesamt wurden rund 8700 Überholvorgänge ausgewertet, davon 1600 auf Radfahrstreifen und 6100 auf Schutzstreifen. Trotz Markierung wurden 50 % aller Überholvorgänge mit einem Abstand von weniger als 1,5 m durchgeführt. Bei Schwerfahrzeugen (Busse und LKW) wurde bei 68 % der Überholvorgänge der seitliche Abstand von 1,5 m unterschritten. Ein signifikanter Unterschied beim Einhalten der seitlichen Abstände konnte zwischen Radfahrstreifen und Schutzstreifen laut der Studie nicht extrahiert werden.



(a) Schutzstreifen, von *verkehrskommentar.de* [15]



(b) Radfahrstreifen, von *muenster.de* [16]

Abbildung 2: Vergleich Radfahrstreifen, mit durchgezogener Linie und Schutzstreifen, mit gestrichelter Linie

## **Ingenieur -und Verkehrspychologie - Technische Universität Braunschweig**

Von Huemer [17] wurden im Fahrsimulator (Software: SILAB 5.0) Überholabstände anhand verschiedener Verkehrsinfrastrukturen und -Situationen mit ca. 40 freiwilligen Probanden getestet und knapp 1000 Überholvorgänge ausgewertet. Wie bereits bei Richter et al. wurden wieder Radfahrstreifen und Schutzstreifen untersucht. Diese wurden aber auch noch Straßen ohne markierte Radverkehrsführung gegenüber gestellt.

Das Ergebnis der Studie kann wie folgt zusammen gefasst werden:

- Der durchschnittliche Überholabstand ist inakzeptabel klein (in 59,5 % der Fälle wurde mit weniger als 1,5 m Abstand überholt)
- Ist keine Radverkehrsführung markiert, ist der Überholabstand größer als mit Radfahrstreifen oder Schutzstreifen
- Ein Unterschied zwischen Radfahrstreifen und Schutzstreifen ist kaum zu erkennen
- Gegenverkehr erhöht die Wahrscheinlichkeit zu eng von einem KFZ überholt zu werden
- Eine mittlere Leitlinie zur Trennung der Fahrspuren verringert den durchschnittlichen Überholabstand
- Auf einer breiteren Fahrspur wird ein größerer Abstand zu Fahrradfahrern eingehalten

## **Radmesser Tagesspiegel Berlin**

Tagesspiegel Online hat im Jahr 2018 in Berlin eine große Befragung von Fahrradfahrern durchgeführt und mit 100 Probanden einen Monat lang die Überholabstände im Stadtgebiet Berlin von deren Fahrrad aus gemessen. Auf 13 300 km mit dem Fahrrad gefahrener Strecke wurden 16 700 Überholvorgänge erfasst. 56 % der Überholvorgänge wurde mit einem seitlichen Abstand von weniger als 1,5 m durchgeführt. Bei markierter Radverkehrsführung sank der Anteil, der Überholvorgänge mit einem seitlichen Abstand von weniger als 1,5 m auf 49 % und liegt damit 7 % unterhalb des Durchschnittswertes. Den größten Effekt hatten Busspuren, welche durch Fahrradfahrer mit benutzt werden dürfen. Hier wurde nur in 44 % der Fälle ein seitlicher Abstand von 1,5 m unterschritten. [9]

Für das Projekt erhielten die Journalisten des Tagesspiegel den Data Journalism Award 2019 [18].

## **Zusammenfassung und Diskussion der Studienergebnisse**

Die beschriebenen Studien ermöglichen keine flächendeckenden und allgemeinen Aussagen über seitliche Abstände beim Überholen, da sie nur punktuell durchgeführt wurden

(Gesamtverband der Deutschen Versicherungswirtschaft e.V), nur im Fahrsimulator (Ingenieur -und Verkehrspsychologie - Technische Universität Braunschweig) oder nur in einer Stadt (Radmesser Tagesspiegel Berlin). Wie bereits bei der Studie durch Tagesspiegel Online aufgezeigt, kommt es bereits innerhalb einer Stadt zu lokal sehr starken Abweichungen. Es ist davon auszugehen, dass die Messergebnisse in unterschiedlichen Städten auch unterschiedliche Ergebnisse liefern werden. Die Studie von Richter et al. zeigt nur lokale Messungen an Straßenabschnitten mit markierter Radverkehrsführung und ermöglicht deshalb keinen kompletten Abgleich mit der Tagesspiegel Online-Studie. Wissenschaftliche Relevanz ist den Studien jedoch zuzuweisen, da sie auf einer breiten Datenbasis aufbauen (über 20 000 Überholvorgänge).

### **1.3 Abgrenzung der Aufgabenstellung**

Für allgemeine Aussagen über Überholabstände bezüglich Radverkehrsinfrastruktur soll ein flächendeckend anwendbares System geschaffen werden, welches die Überholabstände eigenständig aufzeichnet und erkennt. Mit den gewonnenen Erkenntnissen sollen die oben beschriebenen Studienergebnisse überprüft werden.

Mit Hilfe einer Auswertung nach einzelnen Regionen können sowohl Städte und Stadtviertel miteinander verglichen werden, als auch einzelne Gefahrenstellen aufgezeigt werden.

## 2 Stand der Technik

Für das großflächige Erfassen von Überholvorgängen ist es erforderlich ein mobiles Messsystem zu haben, welches vom Fahrrad aus die Überholabstände bestimmt. Im Folgenden werden die Vorgehensweisen und bestehenden Messmethoden zur Bestimmung des Überholabstandes von KFZ gegenüber Fahrradfahrern analysiert und diskutiert. Auf Grundlage der bisher bestehenden Technologien soll eine möglichst günstige und genaue Messmethode entwickelt werden.

### 2.1 Tagesspiegel Berlin - Radmesser

Wie bereits in 1.2.3 gezeigt, wurde durch den Tagesspiegel Berlin mit dem Radmesser-Projekt die bisher größte Studie zum Thema Überholabstände von KFZ gegenüber Fahrradfahrern durchgeführt. Hierfür wurde eigens ein „Radmesser“ entwickelt, welcher am Fahrrad montiert wird, dort die Messungen durchführt und die Ergebnisse per Bluetooth an ein am Lenker montiertes Smartphone übermittelt. Das Smartphone nimmt zusätzlich ein Bild der Verkehrssituation auf. Frei verfügbar sind lediglich die Software für den am Fahrrad montierten Radmesser und die gemessenen Daten, nicht jedoch die zugehörige Smartphone-Applikation. Ebenfalls ist die Auswertungssoftware nicht frei zugänglich.

**Hardwareanalyse** Als Prozessor ein Arduino Nano mit *ATmega168* verwendet, an welchen drei Ultraschallsensoren HC-SR04 angeschlossen sind. Die zwei Sensoren auf der linken Seite sind in einem Winkel von jeweils  $10^\circ$  nach vorne und hinten geneigt. Auf der rechten Seite ermittelt ein Ultraschallsensor den Abstand zu parkenden KFZ. Die Daten werden über das Bluetooth-Modul *AT-09/MLT-BT05* an das am Lenker montierte Smartphone übertragen. Da die Datenübertragungsrate beschränkt ist, werden die Daten der Sensoren komprimiert. Eine 2000 mAh Powerbank sorgt für die Spannungsversorgung und soll den Arduino für 5 Stunden versorgen. Eine LED gibt Auskunft über den aktuellen Betriebszustand. Das Sensorsystem ist in einer vorgefertigten Box verpackt, welche mit Löchern für die Ultraschallsensoren durchbohrt wird und mit Klettbändern am Fahrradrahmen montiert wird. [19]

**Softwareanalyse** Die Software für Arduino-Controller wird in C++ verfasst und mit dem gcc-Compiler kompiliert. Dabei werden zu Beginn des Programmcodes Variablen und Unterprogramme definiert. Das Hauptprogramm wird in eine Dauerschleife gepackt. Die while-Schleife beginnt mit der Prüfung auf externe Signale, beispielsweise die Unterbrechung des Messzyklus. Liegt kein Signal zur Unterbrechung der Messungen vor, wird ein neuer Messzyklus gestartet. Ein Messzyklus besteht aus einem Zeitcode, dann wird vier Mal mit den Sensoren auf der linken Seite gemessen, wobei zuerst der hintere Sensor misst und danach der vordere. Dann wird einmal auf der rechten Seite gemessen, bevor wieder acht Mal mit den Sensoren der linken Seite gemessen wird. Abschließend werden die Daten komprimiert und per Bluetooth an ein Smartphone gesendet. Der Programmcode ist sehr gut strukturiert, kommentiert und durch Auslagerung in viele Unterprogramme übersichtlich gehalten. [9], [19]

Die Software der Smartphone-Applikation und die Auswertungssoftware sind nicht verfügbar.

## 2.2 Flipdot - wheelknife

Von einer freien Programmiergruppe aus Kassel wird daran gearbeitet einen Open-Source-Klon des Radmessers des Tagesspiegels zu erstellen. Die Entwicklungen stocken bei diesem Projekt, der Quellcode ist auf Github frei verfügbar, jedoch nur in geringem Umfang kommentiert und deshalb schlecht verständlich. Ein Schaltplan steht nicht zur Verfügung, jedoch können die Hardwarekomponenten dem auf Github beigefügten Text entnommen werden. Wie beim Radmesser des Tagesspiegels, werden zwei Ultraschallsensoren (HC-SR04) verwendet, um den Überholabstand zu bestimmen. Nach rechts wird keine Messung durchgeführt. Als Controller wird der *ESP-32S* verwendet, welcher von einer chinesischen Firma entwickelt und hergestellt wird. Dokumentationen sind hierfür nur in sehr geringem Umfang verfügbar. [20]

## 2.3 Zweirat Stuttgart RadmesserS und OpenBikeSensor

Die Initiative Zweirat hat durch das Radmesser Projekt des Tagesspiegels inspiriert einen eigenen Radmesser entwickelt.

**Hardwareanalyse** Im Gegensatz zum Radmesser aus Berlin wird als Mikrocontroller ein *ESP-32* verwendet, welcher günstiger ist als ein Arduino Nano und als Spannungsversorgung 3,3 V benötigt. Die Spannungsversorgung wird über einen LiFePo4-Akku mit

ebenfalls 3,3V sicher gestellt. So sollen Verluste durch Spannungswandlungen vermieden werden. Um die Batterie sicher zu laden, wird ein gesondertes Batterieschutzmodul verwendet. Als Ultraschallsensor wird der HC-SR04P verwendet, welcher gegenüber dem HC-SR04 nur 3,3V als Versorgungsspannung benötigt. Es ist in der aktuellen Version dieses Radmessers nur ein Ultraschallsensor verbaut, für weitere Versionen ist geplant mit zwei Sensoren zu arbeiten. Mit Hilfe des Bluetooth-Moduls des *ESP32*, werden dann die aktuellen Abstände an ein Smartphone übermittelt. [21]

Eine zu einem späteren Zeitpunkt erstellte Version basiert auf der Anzeige des Abstandswertes nach links und rechts über ein OLED-Display am Lenker. Durch das Drücken eines Tasters wird die zu dem Zeitpunkt gemessene Distanz abgespeichert. Die Montage des Sensorsystems erfordert Vorkenntnisse im Bereich Elektronik und Löten. [22]

**Softwareanalyse** Um den *ESP32* mit einem Smartphone zu verbinden, muss auf diesem die App *RunnerUp* installiert werden. Der Radmesser wird als Pulssensor in der App hinzugefügt. Anstelle der Herzfrequenz übermittelt er jedoch den Abstand. Die GPS-Daten werden von der App *RunnerUP* über das GPS-Modul des Smartphones zur Verfügung gestellt. [21]

In der nachfolgenden Version, wird durch das Drücken des Tasters ein Überholvorgang aufgezeichnet. Die Software für die Visualisierung der Abstandswerte auf dem OLED-Display ist genauso wie die Software zum Speichern der Überholvorgänge nicht verfügbar.

## 2.4 Codaxus LLC C3FT v3

Von Codaxus LLC wurde 2017 die dritte Version des C3FT präsentiert. Hierbei handelt es sich um einen Abstandsmesser, welcher mit einem Sensor den Abstand zu einem vorbeifahrenden KFZ bestimmt und diesen auf einem LED-Display darstellt. Die Verkehrssituation und das Display werden durch eine Kamera fortlaufend erfasst. Eine Auswertung der Daten muss manuell erfolgen und dient in den USA der Sicherung des Radverkehrs und einer rechtlichen Klarstellung im juristischen Fall. [23]

## 2.5 Analyse bestehender Lösungsansätze

Bei den beschriebenen Lösungen bzw. Lösungsansätzen besteht das Problem, dass lediglich zwei der vier beschriebenen Varianten einen vollständig funktionierenden Abstandsmesser vorzuweisen haben. Beim Radmesser des Tagesspiegels ist nur ein Teil des Projekts frei verfügbar, weshalb diese Lösung aktuell nicht auf andere Städte angewendet werden kann. Da der Radmesser auf eine Bluetooth-Verbindung zu einem Smartphone (mit installierter

App) angewiesen ist, kann er nur von einem festgelegten Besitzer verwendet werden. Ein Wechsel des Fahrrads ist nur zusammen mit dem Smartphone möglich. Beim C3FT müssen die Überholvorgänge aufwändig manuell aus dem Videomaterial herausgearbeitet werden. Da die Videolänge der Fahrtdauer entspricht, ist diese Lösung nicht massentauglich sondern lediglich in Sonderfällen praktikabel. Die durch Zweirat Stuttgart beschriebene Lösung erfordert eine manuelle Bedienung während der Fahrt durch den Fahrradfahrer, was eine große Ablenkung während des Fahrens darstellt und die Messgenauigkeit von der Bedienung des Fahrradfahrers abhängig macht. Weiterhin können die Daten einfache manipuliert werden.

Das zu Anfang beschriebene Ziel, die Stellen im Straßenverkehr zu extrahieren, wo Fahrradfahrer besonders eng überholt werden und deshalb weitere Schutzmaßnahmen erforderlich sind, kann mit keinem der Lösungsansätze befriedigend umgesetzt werden.

**Resultierende Aufgabenstellung** Um die Nachteile der beschriebenen Messsysteme zu eliminieren, und um eine einheitliche, großräumige und kostengünstige Messmethode zur Erfassung der Überholabstände zu etablieren, soll ein eigener Radmesser entwickelt werden. Dieser soll die folgenden Kriterien erfüllen:

- Günstiger Preis von 50 €
- Anhand einer Anleitung montierbar und anwendbar
- Austauschbarkeit des Trägerfahrrades
- OpenSource-Software, für jeden nachvollziehbar
- Alle Ergebnisse öffentlich zugänglich
- Komfortable Bedienung

# 3 Zusammenstellung der Hardwarekomponenten

Nach der Analyse der bisher durchgeführten Projekte zur Messung des Überholabstandes von KFZ gegenüber Fahrradfahrern werden nun in diesem Abschnitt die Hardwarekomponenten diskutiert und zusammengestellt. Es ist festzuhalten, dass eine Recheneinheit, ein Entfernungssensor und eine Möglichkeit zur Positionsbestimmung benötigt werden.

## 3.1 Controller oder Computer

Das Projekt *Tagesspiegel Berlin - Radmesser*, welches sich bereits in mehreren Anwendungen bewährt hat, verwendet den Mikrocontroller Arduino Nano. Der im Radmesser verbaute *Arduino Nano ATmega168* ist nicht mehr verfügbar, das Nachfolgemodell mit einem *ATmega 328*-Prozessor ist preislich bei rund 20 € zu verorten. Die Daten müssen aufgrund einer fehlenden Speichermöglichkeit sofort per Bluetooth an ein gesondert angegliedertes Smartphone übermittelt werden. In den anderen untersuchten Projekten wird eine ähnliche Vorgehensweise gewählt. Durch die Bluetooth-Verbindung, welche das Smartphone während der gesamten Messfahrt benötigt und durch das ständige Aufnehmen von Bildern, ist eine Messfahrt zeitlich stark durch den Smartphoneakku beschränkt. Ein autarkes Funktionieren des Messsystems ohne Smartphone ist nicht möglich. Das Entwickeln einer qualitativ hochwertigen Applikation für Smartphone ist mit großem Aufwand verbunden, da hier die Einarbeitung in zwei unterschiedliche Betriebssystem erforderlich ist, Android und iOS. Um dem System eine längere Akkulaufzeit zu verleihen, soll auch eine Funktion ohne Smartphone gewährleistet werden. Hierfür wird zum einen Speicherplatz benötigt und zum anderen die Möglichkeit ein GPS-Modul anzuschließen, um die gemessenen Überolvorgänge geografisch zu verorten.

Hierbei bietet es sich an mit einem Mikrocomputer zu arbeiten. Dieser bietet von sich aus die Möglichkeit Daten auf eine Micro-SD-Karte zu schreiben, auf welcher auch das Betriebssystem läuft. Bei einem Raspberry Pi kann eine Mikro-SD-Karte mit maximal 256 GB eingelegt werden. Die Daten können entweder mit zusätzlicher Software von der Micro-SD-Karte am PC ausgelesen werden oder die Daten können über eine Remote-

desktopverbindung ausgelesen werden. Bei einem Raspberry Pi stehen 40 GPIO-Pins (General Purpose Input Output) zur Verfügung, an welche Sensoren angeschlossen werden können. Ein Teil der Pins dient der Spannungsversorgung und an andere Pins können als serielle Schnittstelle für Sensoren verwendet werden. [24]

Aus Kostengründen wird der günstigste und einfachste Raspberry Pi verwendet, es handelt sich um den Raspberry Pi Zero. Dieser kostet 5€ bzw. 10€ mit WLAN und Bluetooth-verbindung und wird noch bis mindestens Januar 2026 hergestellt [25]. Zudem existieren mehrere kostengünstige Alternativmodelle [26].

Die Spannungsversorgung mit 5V muss über Mikro-USB geliefert werden und kann von einer Powerbank zur Verfügung gestellt werden. Um eine lange Laufzeit zu gewährleisten, wird eine Powerbank mit einer Kapazität von 10 000 mAh verbaut. Diese unterscheidet sich preislich nur geringfügig von Varianten mit geringerer Kapazität.

## 3.2 Ultraschallsensoren

Wie bereits bei allen analysierten Projekten zu erkennen ist, stellt der Ultraschallsensor HC-SR04 den aktuellen technischen Stand im niedrigen Preissegment für Ultraschallsensoren dar. Er wird auch in vielen weiteren Projekten verwendet [27]–[29], bei welchen ein niedriger Preis erzielt werden muss.

Der Einsatz eines Infrarot-Sensors ist aus mehreren Gründen nicht sinnvoll. Die Kosten eines Infrarot-Sensors sind sechsmal höher wie die eines vergleichbaren Ultraschallsensors. Die Richtcharakteristik eines Ultraschallsensors ist wesentlich besser, dieser hat je nach Quelle einen Messwinkel zwischen 20° und 30°, während ein Infrarot-Sensor Objekte in einem Winkel von 75° erfasst. Die Messabweichungen von Ultraschallsensoren sind bei fast allen Materialien geringer als die der Infrarot-Sensoren [30]. Weiterhin ist mit störenden Lichtverhältnissen für die Infrarot-Sensoren auf der Straße zu rechnen. Aus diesem Grund werden für die Konstruktion des Radmessers Ultraschallsensoren verwendet.

Der Sensor HC-SR04 kann direkt mit den Spannungs-Pins des Raspberry Pi (5V) versorgt werden. Ebenfalls kann der Startimpuls direkt von einem General Purpose Input Output (GPIO)-Pin geliefert werden (3,3V). Der Ultraschallsensor liefert als Rücksignal  $U_2 = 5\text{V}$ . An den GPIO-Pins darf jedoch eine maximale Spannung von  $U_1 = 3,3\text{V}$  anliegen. Mit einem Spannungsteiler, wie in Abb. 3, muss die 5V-Spannung auf 3,3V reduziert werden. Als Faktor  $w$  für den Spannungsteiler ergibt sich:

$$w_{Spannungsteiler} = \frac{U_1}{U_2} = \frac{3,3\text{V}}{5\text{V}} = 0,66 \quad (3.1)$$

Wird nun der größere Widerstand des Spannungsteilers, welcher zwischen Ground (GND)

und dem GPIO-Pin geschaltet wird, zu  $1\text{k}\Omega = R_1$  dimensioniert, so ergibt sich folgende Formel:

$$w_{Spannungsteiler} \stackrel{!}{=} \frac{R_1}{R_1 + R_2} = \frac{1\text{k}\Omega}{1\text{k}\Omega + R_2} \quad (3.2)$$

Die Formel wird nach  $R_2$  umgestellt:

$$R_2 = \frac{R_1}{w_{Spannungsteiler}} - R_1 = \frac{1\text{k}\Omega}{0,66} - 1\text{k}\Omega = 515,15\Omega \quad (3.3)$$

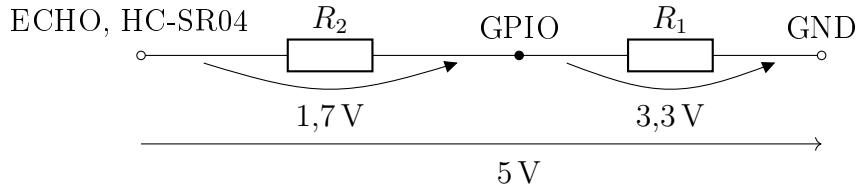


Abbildung 3: Spannungsteiler für Echo-Pin der Ultraschallsensoren HC-SR04 mit Spannungen

Der Widerstand  $R_2$  sollte größer sein als  $515,15\Omega$ , um die GPIO-Pins vor Überspannung zu schützen. Es ergibt sich die in Abb. 3 gezeigte Anordnung der Widerstände mit den dazugehörigen Spannungen.

Die Ultraschallsensoren weisen die beste Erkennungsrate für Objekte auf, welche sich direkt vor den Ultraschallsensoren befinden. Bis zu einer seitlichen Abweichung von  $\beta = 20^\circ$  ist weiterhin eine Objekterkennung gewährleistet, jedoch steigen die Messabweichungen mit zunehmendem Winkel  $\beta$ , also je weiter sich das Objekt von der Messsenkrechten des Ultraschallsensors entfernt befindet, an. [27], [31] (Vgl. Abb. 15)

Von Raza und Monnet wurde bereits ein System bestehend aus mehreren Ultraschallsensoren entwickelt, welches eine vorbeigehende Person erkennt. Hierfür werden mehrere Ultraschallsensoren hintereinander angeordnet. Die Ultraschallsensoren werden dabei um einen Winkel gegeneinander verdreht. Um Überholvorgänge zu erkennen, müssen mehrere Ultraschallsensoren angeordnet werden, hierfür werden diese um einen Winkel voneinander abgeneigt. Dieser Winkel wird in Abschnitt 3.5 bestimmt.

### 3.3 Temperatursensor

Die Schallgeschwindigkeit ist wie in Abschnitt 6.1.2 beschrieben von der Temperatur abhängig. Um die durch eine variable Temperatur entstehende Abweichung der Messwerte zu kompensieren, soll in regelmäßigen zeitlichen Abständen die Temperatur erfasst werden. Für das Erfassen der Temperatur wird ein möglichst günstiger kompakter Temperatursensor mit geringer Energieaufnahme benötigt.

Diese Anforderungen werden vom Sensor DS18B20 erfüllt, welcher per Eindrathbus (1-Wire) an einen Mikrocomputer oder Mikroprozessor angebunden werden kann [32]. Es ist kein Analog-Digital-Wandler (AD-Wandler) notwendig, wodurch Ungenauigkeiten reduziert werden, zusätzliche Hardwarekosten ausbleiben und der Platzbedarf gering bleibt. Der Temperatursensor DS18B20 ist deshalb einem klassischen analogen Temperatursensor mit AD-Wandler vorzuziehen [33]. Der Temperatursensor DS18B20 wurde bereits von Huang, Lei und Bo zur Temperaturkompensation bei Schallgeschwindigkeitsmessungen eingesetzt [34]. Eine Genauigkeit von  $\pm 0,5$  K ist im Temperaturbereich zwischen  $-10^{\circ}\text{C}$  und  $85^{\circ}\text{C}$  gegeben [32]. Dies stellt auch den zu erwartenden Einsatzbereich dar. Die maximal für eine Messung benötigte Zeit liegt bei 750 ms [32], [35]. Die Spannungsversorgung kann entweder über die 5 V oder die 3,3 V Pins des Raspberry Pis erfolgen.

## 3.4 GPS-Modul

Die Position soll mit einem auf der Platine angebrachten GPS-Modul ermittelt werden, welches auch Uhrzeit und Datum liefert. Lediglich in einem der oben genannten Projekte kommt ein GPS-Modul zum Einsatz, es handelt sich um das GPS-Modul *GYGPS6MV2*, welches in der zweiten Version des Radmessers von Zweirat Stuttgart verwendet wird. Die anderen Projekte setzen auf das GPS des Smartphones. Zusätzlich kann auf die Erfahrungen aus anderen studentischen Projekten zurückgegriffen werden, wo das GPS-Modul *UBLOX NEO 6M* verwendet wird [36].

Zwei Faktoren müssen bei der Auswahl des GPS-Moduls beachtet werden. Dies ist erstens der Preis und zweitens die einfache Kompatibilität mit einem Raspberry Pi. U-Blox gilt im Bereich der kostengünstigen GPS-Module als Marktführer. Hierbei empfiehlt es sich aus Kostengründen auf das Modul *NEO-6M* zu setzen, welches über die Universal Asynchronous Receiver Transmitter (UART)-Schnittstelle die Daten als NMEA-Protokoll (National Marine Electronics Association) zur Verfügung stellt. Bereits als einsatzbereite Platine mit Antenne wird das GPS-Modul *GPS6MV2* geliefert. Über die UART-Schnittstelle kann das Modul ohne weitere Anpassungen an den Raspberry Pi angeschlossen werden. Für die Softwareentwicklung muss berücksichtigt werden, dass das GPS-Modul nicht sofort Daten liefert, sondern eine Startzeit von 27 Sekunden aufweist [37].

## 3.5 Entwicklung der Platine

Um die Bauteile möglichst stabil zu montieren und Unterbrechungen im späteren Betrieb zu vermeiden, sollen alle Bauteile auf einer Platine verlötet werden. Für eine genaue Positionierung der Ultraschallsensoren und um die Platine möglichst klein zu gestalten,

wird diese mit einem Platinenlayoutprogramm entwickelt und dann gedruckt. Als Platinenlayoutprogramm wird KiCad gewählt, da das Programm unter der GPL-Lizenz frei verfügbar ist. Es können Schaltpläne erstellt werden und darauf aufbauend Leiterplatten entwickelt werden.

### 3.5.1 Berechnung des Winkels der Ultraschallsensoren

Wie bereits (in 3.2) dargestellt, müssen die Ultraschallsensoren um einen Winkel gegenüber einander verdreht werden.

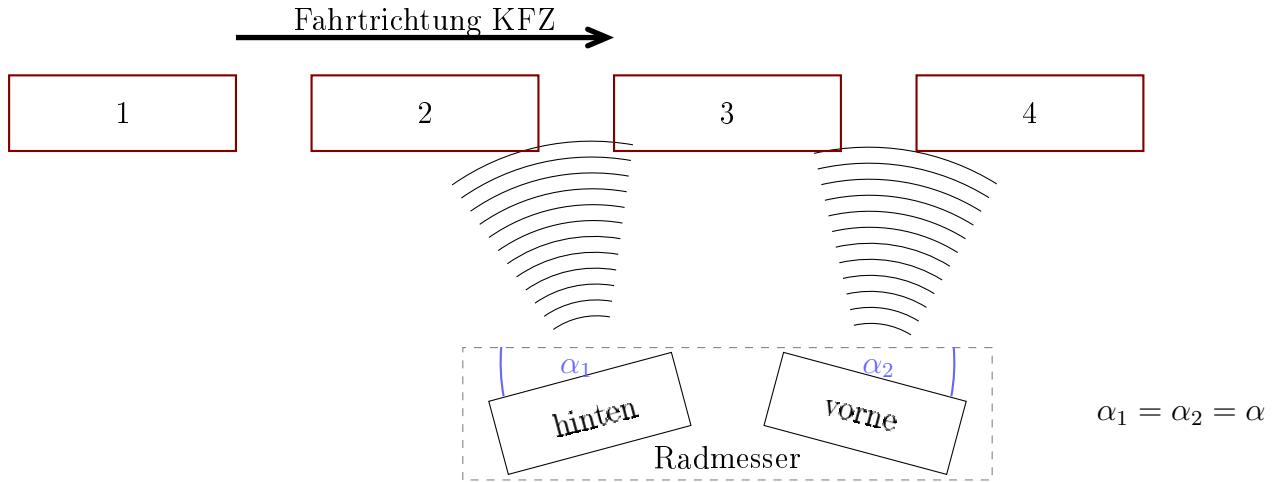


Abbildung 4: Prinzipskizze der Ultraschallsensoren

Dieser Winkel ist in Abb. 4 dargestellt. Im oberen Bildbereich ist ein vorbei fahrendes KFZ skizziert, welches sich von Position 1 zu Position 4 bewegt. Auf Position 1 wird das KFZ von keinem Ultraschallsensor erfasst. Bei einem zeitlichen Fortschreiten erreicht das KFZ Position 2 und wird dort vom hinteren Ultraschallsensor erkannt. Der vordere Ultraschallsensor liefert weiterhin einen sehr großen Wert oder den Maximalwert ( $t_{max} = 38$  ms) zurück. Befindet sich das KFZ auf der gleichen Höhe wie das Fahrrad (Vgl. Position 3), wird es von beiden Ultraschallsensoren erfasst. Beim Weiterfahren verlässt das KFZ den Erfassungsbereich des hinteren Ultraschallsensors und wird auf Position 4 nur noch vom vorderen Ultraschallsensor erfasst. Anhand der zeitlich Differenz bei der Erkennung zwischen hinterem und vorderem Ultraschallsensor, kann später unterschieden werden, ob der Fahrradfahrer überholt wurde, oder ob er ein KFZ auf der rechten Seite überholt hat.

Bereits von Raza und Monnet wurde ein System zur Detektion der Bewegungsrichtung entwickelt. Ein Array aus vier Ultraschallsensoren, mit einer Länge von ca. 80 cm, wobei der vordere und hintere Ultraschallsensor von den anderen Ultraschallsensoren abgeneigt sind, ermittelt permanent die Abstandswerte. Der Radmesser muss wesentlich kompakter konstruiert werden, da auf einem Fahrradgepäckträger der zur Verfügung stehende Platz auf circa 20 cm begrenzt ist. Das von Raza und Monnet entwickelte System ist

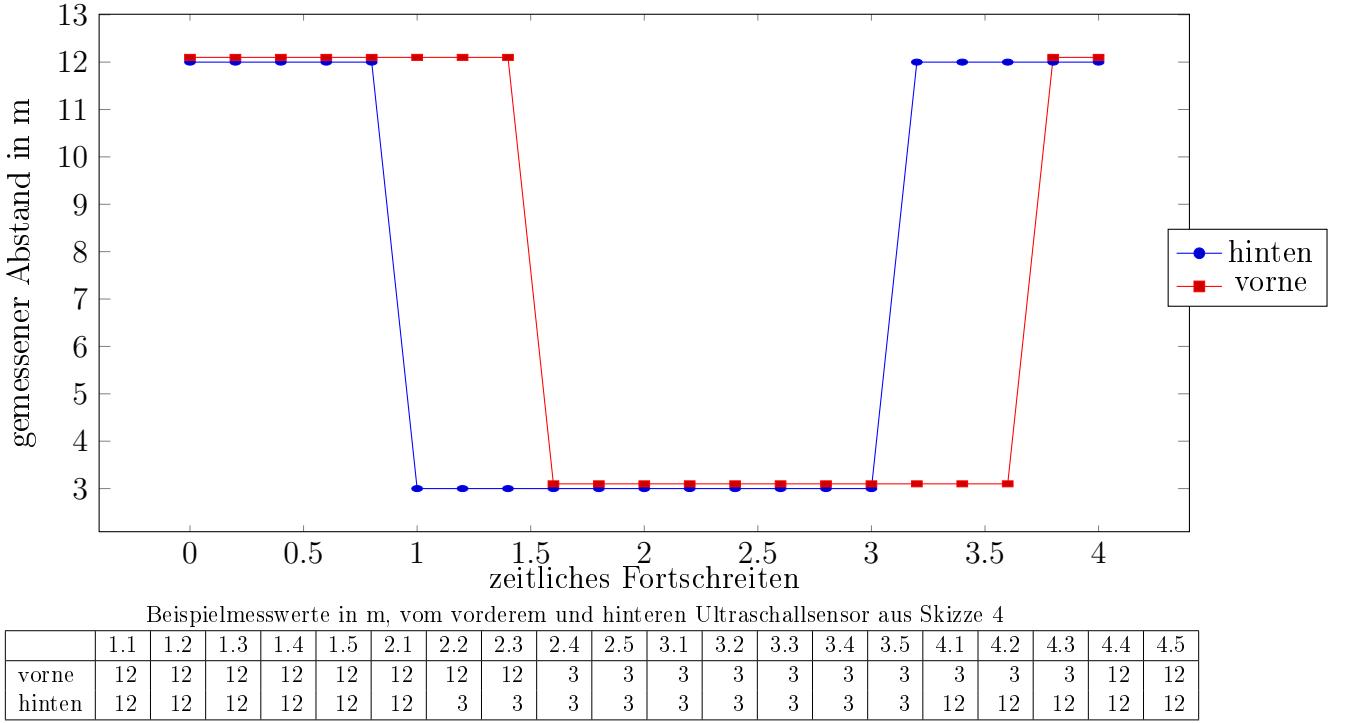


Abbildung 5: Beispieldaten nach Prinzipskizze 4, dargestellt in einer Tabelle und einem Graphen

auf langsame Objekte ausgelegt mit  $v = [1,5; 2,5] \text{ m/s}$ . Es wird diskutiert, wie oft das Objekt durch die Ultraschallsensoren erfasst wird. Dabei ist ersichtlich, dass bei höherer Geschwindigkeit das Objekt weniger oft erfasst wird. [28]

Um den Winkel zu bestimmen, welchen die Ultraschallsensoren voneinander abgewandt sein müssen, sodass zu Beginn nur der hintere Ultraschallsensor das KFZ erfasst (Vgl. Abb. 4), wird vereinfachend angenommen, dass die Ultraschallsensoren geometrisch am gleichen Punkt sind und bei einem Winkel von  $\alpha = 0^\circ$  die Ultraschallsensoren ein Objekt exakt gleichzeitig detektieren. Zuerst muss die minimale Strecke  $l_{min}$  bestimmt werden, welche nur von einem Ultraschallsensor überblickt werden darf, sodass ein KFZ mindestens einmal nur vom hinteren Ultraschallsensor erkannt wird. Wie in 9.2 diskutiert, liegt die maximal mögliche Geschwindigkeitsdifferenz zwischen KFZ und Fahrradfahrer bei  $v_{max\_Land} = 120 \frac{\text{km}}{\text{h}} = 33,33 \text{ m s}^{-1}$ . Mit der maximalen Geschwindigkeit  $v_{max\_Land}$  und der maximalen Zeit zwischen zwei Messungen  $t_{max}$ , kann die Strecke  $l_{min}$  berechnet werden.

$$l_{min} = v_{max\_Land} \cdot t_{max} = 33,33 \text{ m s}^{-1} \cdot 0,038 \text{ s} = 1,266 \text{ m} \quad (3.4)$$

Bei einer Geschwindigkeit von  $120 \frac{\text{km}}{\text{h}}$ , ist mit einem minimalen Überholabstand von 1 m auszugehen. Die Distanz, welche die Ultraschallsensoren messen ist größer als der seitliche Abstand, da die Messung von der Fahrradmitte, bis zum KFZ-Hauptkörper statt findet. Die Wahrscheinlichkeit, dass der Außenspiegel eines KFZ erfasst wird, ist verschwindend gering. Die Distanz, welche die Ultraschallsensoren messen ( $l_{Mess}$ ), wird deshalb rund

1,5 m betragen.

$$\alpha_{total} = 2 \cdot \alpha = \arctan \left( \frac{l_{max}}{l_{Mess}} \right) = \arctan \left( \frac{1,266 \text{ m}}{1,5 \text{ m}} \right) = 45,8^\circ \quad (3.5)$$

Da beide Ultraschallsensoren um den gleichen Winkel nach außen geneigt werden, gilt  $\alpha = \alpha_{total}/2$ . Dies bedeutet nach (3.5), dass die Ultraschallsensoren jeweils um  $22,9^\circ$  nach vorne und hinten geneigt sein müssen. Die Ultraschallsensoren haben jedoch wie in Abschnitt 6.1.5 beschrieben einen Messbereich von maximal  $20^\circ$ . Wird  $\alpha \geq 20^\circ$ , wird ein KFZ auf Position 3 (Abb. 4) nicht mehr ausreichend erkannt. Aus diesem Grund muss ein Kompromiss getroffen werden zwischen der eindeutigen Erkennung eines Überholvorgangs unter extremen Bedingungen ( $v = 120 \frac{\text{km}}{\text{h}}$ ) und der Messgenauigkeit.

Die beschriebenen Extrembedingungen sind nur außerhalb geschlossener Ortschaften zu erwarten. Der Einsatz des Radmessers liegt primär in der Stadt (Abschnitt 9.2). Aus diesem Grund soll für eine Neuberechnung des Winkels die maximale Geschwindigkeit, wie in Abschnitt 9.2.1 definiert, verwendet werden. Diese beträgt  $v_{KFZ\_Land} = 70 \frac{\text{km}}{\text{h}}$ , mit der Formel (3.4) ergibt sich eine minimale Distanz von  $l_{min} = 0,739 \text{ m}$ . Setzen wir diese Distanz in die Formel (3.5) ein, ergibt sich ein Winkel von  $\alpha = 13,83^\circ$ .

Durch einen Winkel  $\alpha \leq 20^\circ$  wird die Messgenauigkeit senkrecht zum Fahrrad kaum beeinflusst ([27] und Abschnitt 6.1.5). Durch Tests wurde gezeigt, dass mit einem Winkel  $\alpha \approx 16^\circ$  die besten Ergebnisse erzielt werden.

### 3.5.2 Schaltplanentwicklung

Der Schaltplan wird mit dem Tool Eeschema erstellt, welches in KiCad enthalten ist. Als zentrales Element dient die 40-Pin-Leiste des Raspberry Pi Zero. An diese werden alle anderen Bauteile angeschlossen. Zuerst werden die beiden Ultraschallsensoren mit den oben berechneten Spannungsteilern eingeplant. Die Spannungsversorgung erfolgt über die 5 V-Pins des Raspberry Pis. Das GPS-Modul wird über die 3,3 V-Pins des Raspberry Pis versorgt. Da das GPS-Modul GY-GPS6MV2 ein serielles Protokoll als Antwort liefert, werden dessen Pins an die UART-Schnittstelle angeschlossen. Der Temperatursensor DS18B20 wird mit dem GPCLK0-Pin (General Purpose Clock), 3,3 V und GND verbunden, wobei ein  $4,7 \text{ k}\Omega$ -Widerstand zwischen die Datenleitung (1-wire, General Purpose Clock (GPCLK)) und die Spannungsversorgung als Pull-Up-Widerstand geschaltet wird. Zur Funktionsüberwachung werden zwei Low-Current-LEDs eingesetzt. Diese haben gegenüber normalen LEDs den Vorteil, dass sie lediglich einen Strom von 2 mA benötigen, was die Belastung des Raspberry Pis senkt und die Akkulaufzeit verlängert. Der Schaltplan ist im Anhang in Abschnitt 9.6.1 zu finden.

### 3.5.3 Layout der Platine

Um sowohl die Kosten für das Gehäuse, als auch für die Platine möglichst gering zu halten, soll diese möglichst kompakt gestaltet werden. Die Platine soll weiterhin möglichst übersichtlich gestaltet werden, indem die Zahl der Durchkontakte gering gehalten wird, da so der Verlauf der Leiterbahnen besser nachvollzogen werden kann und auf diese Weise Anpassungen leichter vorgenommen werden können. Ein kompaktes Platinen-Design wird durch das parallele Anordnen der Widerstände erreicht. Um den Platzbedarf für Leiterbahnen zu reduzieren, wird die Masse (GND) als Massefläche auf der Unterseite der Platine ausgeführt. Eine weitere Reduzierung der Platinengröße ist mit den aktuell verwendeten Bauteilen nicht möglich, da diese einen hohen Eigenplatzbedarf aufweisen. Durch die Verwendung von Surface mounted devices (SMD) Bauteilen kann der Platzbedarf weiter reduziert werden. SMD-Bauteile können jedoch nicht selbst auf die Platine gelötet werden, was die Bestückungskosten wesentlich nach oben treibt.

## 3.6 Gehäuse

Für das Gehäuse sind mehrere Überlegungen zu berücksichtigen:

- Das Gehäuse im entsprechenden Winkel zulaufen lassen und auf diese Weise die Anordnung der Ultraschallsensoren unterstützt
- Das Gehäuse möglichst einfach gestalten
- Die Powerbank auf der Unterseite platzieren, um ein besseres Gleichgewicht zu erzielen
- Die Batterie auf die Rückseite platzieren, um den Platzbedarf zu reduzieren
- Den Raspberry Pi senkrecht zur Platine montieren, um Platz zu sparen
- Den Raspberry Pi flach (parallel) zur Platine montieren, um Höhe zu sparen

Das Gehäuse kann wie von Bauer, Jörg und Wagner (2019) beschreiben durch 3D-Druck produziert. Die von Bauer, Jörg und Wagner beschriebenen Anforderungen, sind an manchen Stellen zu relativieren. Ein spritzwasserdichtes Gehäuse ist insofern nicht erforderlich, da die Ultraschallsensoren bei Regen o.ä. nicht korrekt messen können und zudem durch Feuchtigkeit beschädigt werden können. Ein Betrieb bei Niederschlag ist deshalb nicht möglich. Der Aufwand, um das Gehäuse spritzwassergeschützt zu gestalten entfällt deshalb. Die Powerbank stellt das größte Bauteil dar, deshalb muss das Gehäuse auch auf sie ausgelegt werden. Ein besonderes Augenmerk ist auf den Eingang der Powerbank zu legen, da dieser zum Laden des Radmesser zugänglich sein muss. Wird der Ladeeingang der Powerbank nach außen geführt, sind an dieser Stelle zusätzliche Maßnahmen zur Abdichtung zu treffen. Obwohl die Ultraschallsensoren in einem Winkel zueinander

montiert sind, kann ein quaderförmiges Gehäuse verwendet werden. Bei der Verwendung eines Gehäuses mit ebenen Flächen, werden die Ultraschallsensoren auf der einen Seite etwas weiter aus dem Gehäuse ragen, dies beeinflusst die Messungen nicht. Um dem Gehäuse eine gute Stabilität zu verleihen, soll es kompakt konstruiert sein, die Höhe soll möglichst gering sein und schwere Bauteile sollen möglichst im unteren Bereich verbaut sein. Aus diesem Grund wird die Powerbank auf den Boden des Gehäuses montiert. Die selbst entwickelte Platine mit allen darauf befestigten Bauteilen findet ihren Platz direkt auf der Powerbank. Die minimale Höhe des Gehäuses ergibt sich aus der Höhe der Ultraschallsensoren und der Höhe der Powerbank. Die Breite und Länge des Gehäuses müssen etwas großzügiger gestaltet werden, da dort Platz für das Kabel sein muss, welches die Verbindung zwischen Powerbank und Raspberry Pi herstellt. Dieses enthält den Schalter, mit welchem der Radmesser ein und aus geschaltet wird. Der Schalter muss durch ein Loch in der Gehäusehülle von außen erreichbar sein. Das Kabel überragt sowohl die Powerbank, als auch den Raspberry Pi um rund 3 cm, da es nicht direkt abgeknickt werden kann.

Für die Produktion eines Prototypen ist es ausreichend den Radmesser in einer bereits existierenden Plastikbox zu platzieren (Abb. 6). Die Powerbank ist mit Hilfe von doppelseitigem Klebeband auf dem Boden des Gehäuses befestigt. Um die Platine mit allen Bauteilen sicher montieren zu können, wird diese zuerst auf einer planaren Plastikplatte verklebt, welche wiederum mit doppelseitigem Klebeband auf der Powerbank verklebt wird. An die Stelle der Ultraschallsensoren ist ein Loch in das Gehäuse zu schneiden. Für das Aufladen der Powerbank, sowie das Ein- und Ausschalten des Radmessers muss beim Gehäuse-Prototyp jedes Mal der Deckel abgenommen werden.



Abbildung 6: Prototyp des Radmessers in einer existierenden Plastikbox

# 4 Entwicklung der Radmesser-Software

## 4.1 Softwarekonzept

Auf dem Raspberry Pi wird als bevorzugte Programmiersprache Python 3 verwendet. Für die Anbindung der Sensoren liegen verschiedene Programmierbeispiele vor.

Ein stabiler Programmablauf ist wichtig, da im normalen Betrieb, wenn der Radmesser am Fahrrad angebracht ist, kein Eingriff erfolgen kann. Dies erfordert ausführliche Tests. Die größte Gefahr für einen ungewollten Abbruch des Programms besteht darin, dass die Ultraschallsensoren falsche Rücksignale liefern, beispielsweise weil sie das Trigger-Signal nicht erhalten haben. Durch fehlenden Satelliten-Empfang kann es vorkommen, dass kein GPS-Signal geliefert wird.

Es sollen möglichst viele Messungen innerhalb der zur Verfügung stehenden Zeit durchgeführt werden. Als physikalische Grenze ist die Schallgeschwindigkeit gegeben, welche die benötigte Zeit für Messungen mit den Ultraschallsensoren definiert. Die Ultraschallsensoren messen Entferungen zwischen 4 cm und 5 m. Nachdem der Sensor einen Startimpuls (10 ms *High*) bekommen hat, sendet er mehrere kurze Ultraschallimpulse ( $f = 40\text{ kHz}$ ) aus. Wurden die Schallimpulse ausgesendet, wird der Echo-Pin auf *High* gesetzt. Sobald die Ultraschallimpulse vom Sensor detektiert wurden, fällt der Echo-Pin zurück auf *Low*. Wird kein Ultraschall detektiert, wird der Echo-Pin nach  $t_{max} = 38\text{ ms}$  zurückgesetzt. Eine neue Messung kann erst durchgeführt werden, wenn der Echo-Pin auf *Low* ist. Um die Daten möglichst schnell und ohne Umwege abzuspeichern, werden diese in Dateien auf der Micro-SD-Karte geschrieben. Bei Verwendung einer 16 GB Micro-SD-Karte stehen bis zu 5 GB freier Speicherplatz zur Verfügung.

Um einen möglichst einfachen Start des Radmessers zu realisieren, wird direkt nach dem Hochfahren des Raspberry Pis die Radmesser-Software gestartet.

### 4.1.1 Parallele Sensorsauswertung

Um möglichst viele Messungen in der zur Verfügung stehenden Zeit durchzuführen, sollen die Ultraschallsensoren gleichzeitig messen. Beide Ultraschallsensoren erhalten den gleichen Startimpuls und senden beinahe zeitgleich ihre Ultraschallimpulse. Die beiden Sensoren sind in einem Winkel von jeweils  $16^\circ$  nach vorne und hinten geneigt montiert (siehe Abb. 4). Durch die Neigung der Ultraschallsensoren messen diese unabhängig voneinander und erfassen, ob ein KFZ den Fahrradfahrer überholt oder der Fahrradfahrer auf der rechten Seite überholt bzw. an einem Objekt vorbeifährt. Überholt ein KFZ, dann detektiert zuerst der hintere Sensor einen geringeren Abstand und danach der vordere Sensor. Überholt hingegen ein Fahrradfahrer auf der rechten Seite oder er fährt an einem feststehenden Objekt vorbei, dann stellt zuerst der vordere Sensor eine geringere Distanz fest und dann der hintere.

Nachdem beide Sensoren den Startimpuls erhalten haben, wird mithilfe einer *while*-Schleife gewartet, bis beide Ultraschallsensoren ein *High*-Signal zurückgeben. Innerhalb der *while*-Schleife wird die Startzeit  $t_{Start}$  mit der aktuellen Systemzeit beschrieben. In einer weiteren *while*-Schleife wird auf das Zurückfallen der Ultraschallsensoren auf *Low* gewartet, was spätestens nach  $t_{max} = 38\text{ ms}$  der Fall ist. Solange ein Sensor ein *High*-Signal liefert, wird dessen Endzeit  $t_{End}$  mit der aktuellen Systemzeit beschrieben. Nachdem beide Sensoren die Messung beendet haben, wird die Zeitdifferenz zwischen Endzeit und Startzeit bestimmt ( $t_{diff} = t_{End} - t_{Start}$ ). Aus der Zeitdifferenz und der Schallgeschwindigkeit  $c_{Schall}$  wird die Distanz  $l_{Mess}$  errechnet:

$$l_{Mess} = \frac{1}{2} \cdot c_{Schall} \cdot t_{diff} = \frac{1}{2} \cdot 340 \frac{\text{m}}{\text{s}} \cdot t = 17000 \frac{\text{cm}}{\text{s}} \cdot t_{diff} \quad (4.1)$$

Der Faktor  $1/2$  wird benötigt, da die Zeitdifferenz zwischen Start- und Endzeit sowohl das Hin, als auch das Zurück des Schalls berücksichtigt. Der Temperatureinfluss auf die Schallgeschwindigkeit wird erst durch die Auswertungssoftware korrigiert.

### 4.1.2 Erfassen von Zeit und Position

Die Zeit- und Positionsbestimmung sind weniger zeikritisch, als die Messung des seitlichen Abstands. Bei der Zeit- und Positionsbestimmung werden weniger Daten benötigt, da zwischen den einzelnen Punkten interpoliert werden kann. Stehen zwei Punkte eindeutig fest, kann linearen interpoliert werden. Die Systemzeit des Raspberry Pis kann für eine absolute Zeitangabe nicht verwendet werden, da dem Raspberry Pi gegenüber einem Smartphone, Laptop o.ä. eine batteriebetriebene Uhr fehlt, welche die Zeit auch im ausgeschalteten Zustand weiterzählt. Der Raspberry Pi zählt nach einem Neustart mit seiner Systemzeit dort weiter, wo er beim Herunterfahren aufgehört hat. Als Zeit

muss deshalb die vom GPS-Modul gelieferte Coordinated Universal Time (UTC) verwendet werden. Diese ist auch bei schlechtem Satellitenempfang verfügbar. Die maximalen Zeitabstände  $t_{max\_GPS}$  für zwei Koordinatenabfragen lassen sich mit der maximalen Geschwindigkeit eines Fahrradfahrers und der maximalen Distanz zwischen zwei Punkten, zwischen welchen interpoliert werden kann errechnen. Als maximale Geschwindigkeit des Fahrradfahrers nehmen wir  $v_{max\_Fahrrad} = 30 \frac{\text{km}}{\text{h}} = 8,33 \text{ m s}^{-1}$  an. Um Verwechslungen und Ungenauigkeiten bei der Interpolation zu vermeiden, dürfen zwei GPS-Koordinaten, maximal  $l_{max\_GPS} = 20 \text{ m}$  voneinander entfernt sein. Es ergibt sich folgende Rechnung:

$$t_{max\_GPS} = \frac{l_{max\_GPS}}{v_{max\_Fahrrad}} = \frac{20 \text{ m}}{8,33 \text{ m s}^{-1}} = 2,4 \text{ s} \quad (4.2)$$

Spätestens alle 2,4 Sekunden muss eine GPS-Position bestimmt werden, um Fehler beim Interpolieren zwischen den einzelnen Punkten zu minimieren.

### 4.1.3 Temperatursensor

Die Messung der Temperatur nimmt bezogen auf die Entfernungsmessungen eine große Zeitspanne von 750 ms in Anspruch [35]. Wie in Abschnitt 6.1.2 gezeigt ist die Temperaturmessung unerlässlich für genaue Messergebnisse. Die Lufttemperatur ist eine sehr träge Größe, welche sich während einer normalen Fahrradfahrt von einer Stunde maximal um 5 K ändert. Zwischen verschiedenen Fahrradfahrten und abhängig vom Ort, sind jedoch Temperaturschwankungen von bis zu  $\Delta T = 45 \text{ K}$  möglich. Es ist ausreichend, wenn die Temperatur im Minutentakt erfasst wird. Durch den verwendeten 1-Wire-Bus des Temperatursensors DS18B20 wird dieser automatisch vom Raspberry Pi erkannt. Ein Zugriff auf den Sensor erfolgt ähnlich einem Dateizugriff, wobei das Ergebnis entpackt wird und in Grad Celsius umgerechnet wird.

### 4.1.4 Anzeigen des Betriebszustandes

Wie in Abschnitt 3.5 beschrieben, werden zwei LEDs zur Anzeige des aktuellen Betriebszustandes eingebaut. Eine grüne LED signalisiert den reibungsfreien Betrieb, die rote LED warnt bei Abweichungen und Messfehlern. Anhand der LEDs soll erkennbar sein, ob die Software einwandfrei läuft und Messungen durchführt, welche ein Ergebnis innerhalb des Messbereichs ergeben. Weiterhin soll signalisiert werden, ob ausreichend Speicherplatz auf der Micro-SD-Karte vorhanden ist.

Vor jeder Messfahrt werden alle Sensoren getestet. Während der Tests leuchtet die rote LED, danach blinken die beiden LEDs im Gleichtakt 20 Sekunden lang auf, um dem GPS-Modul ausreichend Zeit zum Starten bzw. Hochfahren zu geben. Danach leuchtet

die grüne LED.

Der verbrauchte Speicherplatz wird auf Grundlage des Zählers für die Dateieinnummerierung ermittelt. Wie in 4.2 definiert, werden in jeder Datei 5000 Messpaare gespeichert. Je nach Messwert, also Abstand der Ultraschallsensoren vom Messobjekt variiert der Speicherbedarf leicht. Im Straßenverkehr, wo große Distanzen gemessen werden, kann eine Datei bis zu 150 KB groß werden. Es stehen 5 GB an Speicherplatz zur Verfügung. Der Speicherplatz reicht für rund 33 000 Dateien. Um die Datensicherheit zu gewährleisten, soll bereits deutlich vor erreichen der 33 000 Dateien vor einem vollen Speicher gewarnt werden. Nach Erstellung der Datei mit der Nummer 25 000 wird die grüne LED ausgeschaltet.

Die Hauptfunktion der roten LED besteht darin, die Richtigkeit der Messungen zu bestätigen. Liegt ein Messwert außerhalb des realistischen Bereiches, also kleiner der Lenkerbreite oder größer der maximalen Messdistanz, wird die rote LED angeschaltet, beim nächsten Messwert innerhalb der Grenzen, wird die rote LED ausgeschaltet. Bei einem kurzen Aufblicken der roten LED ist einmalig ein fehlerbehafteter Messwert aufgetreten, erst wenn die rote LED regelmäßig aufleuchtet oder nicht mehr erlischt, muss eine Fehlersuche und -beseitigung durchgeführt werden.

## 4.2 Dateiformat

Um den Speicherbedarf möglichst gering zu halten, wird eine CSV-Datei erstellt. In einer CSV-Datei werden die Werte bzw. Spalten durch Kommata voneinander getrennt, Punkte hingegen dienen als Dezimaltrennzeichen. In der ersten Spalte wird die Zeit, welche seit Beginn der Messfahrt vergangen ist in Sekunden notiert. Die zweite Spalte wird mit dem Abstandswert des vorderen Ultraschallsensors versehen und die dritte Spalte mit dem Messwert des hinteren Sensors belegt. Der Speicherbedarf kann weiter reduziert werden, wenn nur Überholvorgänge gespeichert werden. Dies erfordert jedoch zusätzliche Rechenzeit auf dem Raspberry Pi, um aus den Messergebnissen die Überholvorgänge zu erkennen, und geht auf Kosten der Häufigkeit der Abstandsmessungen und wird deshalb nicht angewendet.

Für die Bestimmung der Abstände wird wie oben beschrieben von einer konstanten Schallgeschwindigkeit ausgegangen. Dies entspricht nicht der Realität, sorgt jedoch für einen reduzierten Rechenaufwand und erhält die Daten im Originalformat. Aus der Zeit, welche der Ultraschall für die Strecke benötigt, wird mit der konstant angenommenen Schallgeschwindigkeit eine Entfernung gemessen. Die Zeit zu speichern, welche der Schall zwischen Ultraschallsensor und Objekt unterwegs ist, ist nicht praktikabel, da die Zeit wenige Millisekunden umfasst und deshalb viele Nachkommastellen benötigt. Durch das Abspeichern

einer Distanz ist ein schneller Überblick über die Messwerte möglich.

Zu den Sensordaten werden auch die GPS-Daten abgelegt. Um den Speicherbedarf gering zu halten, werden die GPS-Daten nicht in jeder Zeile zu den Abstandswerten hinzugefügt, sondern lediglich in einer einzelnen Zeile abgelegt. Durch die in den GPS-Daten enthaltene Zeichenkombination *RECOMMENDED MINIMUM SPECIFIC GPS/TRANSIT DATA (GPRMC)* (Vgl. Codebeispiel 4.1) wird die Zeile mit den GPS-Daten sofort von der Auswertungssoftware erkannt und entsprechend behandelt. Über die GPS-Daten wird die Uhrzeit synchronisiert. Mit den Temperaturwerten wird auf die gleiche Weise verfahren, jedoch werden diese weniger häufig bestimmt (Vgl. oben).

Wird der Raspberry Pi nicht wie unten beschrieben ordnungsgemäß per Tastendruck heruntergefahren sondern durch das An- und Ausschalten der Versorgungsspannung bedient bzw. es kommt aus einem anderen Grund zu einem Spannungsausfall, muss sichergestellt werden, dass in dem Moment des Spannungsabfalls keine Datei beschrieben wird. Die Datei, welche zum Zeitpunkt des Spannungsabfalls beschrieben wird, geht sonst verloren. Aus diesem Grund werden in regelmäßigen Abständen neue Dateien mit fortlaufender Nummerierung erzeugt und die aktuell beschriebene Datei wird geschlossen. Bei jedem Programmstart wird geprüft, ob die Dateien mit dem Namen *messdatenx.txt* existiert, wobei *x* eine Variable darstellt, welche hochgezählt wird. Erst wenn eine Datei mit dem entsprechenden Namen nicht mehr existiert, wird eine neue Datei erzeugt, in welche die neuen Messwerte gespeichert werden. Nach einer definierten Zahl an Messungen (in unserem Fall 5000), wird die aktuelle Datei geschlossen und eine neue Datei mit einer neuen Nummer erstellt. Durch das regelmäßige Erstellen, Beschreiben und Schließen von Dateien wird sichergestellt, dass auch bei Spannungsausfall die Daten bis zur letzten vollen Minute erhalten bleiben. Die Daten, welche innerhalb der letzten 120 Sekunden gemessen wurden, gehen verloren, dies ist vernachlässigbar.

Wie im Codebeispiel 4.1 skizziert, wird die Temperatur einmal zu Beginn jeder Datei bestimmt und gespeichert. Die GPS-Daten werden alle 100 Abstandsmessungen erfasst. Da eine Entfernungsmessung im Schnitt 20 ms (zwischen 5 ms und 38 ms je nach Abstand) in Anspruch nimmt, werden im Durchschnitt 2 s für 100 Messungen beansprucht. Dies genügt der in Gleichung (4.2) bestimmten maximalen Zeit.

Code Listing 4.1: Radmesser Dateibeispiel

```
1 Temperatur,17.537
2 $GPRMC,102124.00,A,4847.55609,N,00935.87220,E,10.706,281.97,220220,,A*5F
3 1200.8,1198.7
4 1000.5,1187.7
5 :
6 200.5,220.8
7 1100.0,230.2
```

```
8 $GPRMC,121709.00,A,4848.33967,N,00933.99514,E,12.166,121.68,220220,,A*5D
9 :
```

Über eine Remotedesktopverbindung können die gespeicherten Daten auf einen Rechner kopiert werden und gleichzeitig auf der Micro-SD-Karte des Raspberry Pis gelöscht werden. Für den Aufbau einer Remotedesktopverbindung müssen sich der Raspberry Pi und der Rechner, auf welchen die Daten kopiert werden im selben (WLAN)-Netzwerk befinden.

## 4.3 Ausfallsicherheit

Bei erstens Tests lief das Programm zur Messung- und Aufzeichnung des Überholabstands immer für 2 bis 10 Sekunden und führte in dieser Zeit circa 100 bis 1000 Messungen durch. Danach wurden keine Messungen mehr durchgeführt. Wurde der zeitliche Abstand zwischen den Messungen auf mindestens 100 ms erhöht, konnten beinahe beliebig viele Messungen nacheinander durchgeführt werden. Für eine gute Datenlage ist es jedoch erforderlich, dass der zeitliche Abstand zwischen den Messungen minimal ist. Aus diesem Grund ist es nicht möglich, Wartezeiten zu implementieren. Beim Untersuchen des Quellcodes mithilfe der Bildschirmausgabe wurde ermittelt, dass die *while*-Schleife, mit deren Hilfe auf ein *High*-Signal beider Sensoren gewartet wird, der Stopp-Punkt des Programms war. Durch die Überwachung, der Zeit, welche das Programm in der *while*-Schleife verbleibt, wird diese Zeit begrenzt und so ein Stehenbleiben des Programms verhindert. Nach 50 ms springt das Programm mithilfe eines *break*-Befehls aus der *while*-Schleife.

## 4.4 Realisierung der Software

Das Programm lässt sich in drei Teile gliedern. Zu Beginn wird eine Startroutine durchlaufen, danach findet ein Übergang in eine *while*-Schleife statt, in welcher das Programm während der gesamten Betriebszeit verbleibt. Beendet wird das Programm durch Tasterdruck oder Spannungsabfall.

Die Startroutine beginnt mit dem Einbinden aller notwendigen Pakete, der Definition von Variablen und dem Setzen von GPIOs als In- bzw. Outputs. Es wird geprüft, bis zu welcher Nummer bereits Dateien vorhanden sind (siehe oben). Danach wird das GPS-Modul das erste Mal angesprochen, um diesem ausreichend Zeit zum Hochfahren zu geben, bis das erste Mal Daten abgerufen werden. Die Ultraschallsensoren werden getestet und der Temperatursensor wird geprüft. Die Startroutine wird erst durch das Leuchten der roten LED, dann durch das Blinken beider LEDs visualisiert. Wurde die Startroutine erfolgreich durchlaufen, geht das Programm in die *while*-Schleife über, welche während der gesamten Betriebszeit des Radmessers durchlaufen wird. Dort werden in regelmäßigen Abständen

neue Dateien erstellt (Vgl. Datensicherheit). Die Dateien werden immer mit der gleichen Menge an Messwerten befüllt. Die *while*-Schleife ruft die Unterprogramme für die verschiedenen Messungen auf. Ist eine Messfahrt beendet, kann durch Drücken eines Tasters der Radmesser ordnungsgemäß heruntergefahren werden. Der GPIO-Pin, welcher das Herunterfahren des Raspberry Pi auslösen soll, wird stets von einem Hintergrundprozess überwacht.

Die Softwaredistribution für den Raspber Pi kann über zwei Wege erfolgen. Zum einen kann der Python-Quellcode auf den Raspberry Pi heruntergeladen werden. Zum anderen kann das gesamte Betriebssystem des Raspberry Pis mit allen benötigten Dateien auf eine Micro-SD-Karte gespielt werden. Wird das gesamte Betriebssystem auf eine SD-Karte gespielt, wird die Software *Win32 Disk Imager* benötigt. Der Vorteil dieser Lösung besteht darin, dass keine weiteren Einstellungen mehr am Raspberry Pi notwendig sind und nach dem Zusammenbauen der Hardware und dem Einlegen der Micro-SD-Karte sofort mit Messungen begonnen werden kann. Wird hingegen nur der Quellcode heruntergeladen, muss ein Betriebssystem auf der Micro-SD-Karte vorinstalliert sein und weitere Einstellungen müssen vorgenommen werden. Für einen Anwender ohne Linux-Kenntnisse oder Erfahrungen mit einem Raspberry Pi ist dieser Weg nicht zu empfehlen.

Der gesamte erstellte Quellcode ist im Anhang 9.5 zu finden, dort sind auch weiter Erklärungen angefügt.

# 5 Entwicklung der Auswertung-Software

Die Auswertungssoftware soll aus den ermittelten Daten Überholvorgänge erkennen. Um eine bestmögliche Wartbarkeit und die Möglichkeit für Erweiterungen der Auswertungssoftware zu erreichen, wird diese stark modularisiert. Die benötigten Daten werden in *structs* gespeichert.

## 5.1 Eingangs-Datenbearbeitung

Aufgrund der Ausfallsicherheit werden die Daten über die Überholabstände in vielen einzelnen Dateien abgespeichert. Für eine sinnvolle Auswertung werden die Messwerte aus den einzelnen Dateien eingelesen und in einzelne Fahrten aufgeteilt. Die Daten jeder einzelnen Fahrt werden getrennt ausgewertet hinsichtlich Überholvorgänge und gefahrener Strecke.

**GPS-Daten verarbeiten** Die GPS-Daten sind nicht in jeder Zeile enthalten, um Speicherplatz auf dem Raspberry Pi zu sparen. Zusätzlich kann es sein, dass die GPS-Daten nicht bei jedem Aufruf zuverlässig empfangen, verarbeitet und in die Datei geschrieben wurden. Es wird deshalb eine Interpolation zwischen den bestehenden Daten durchgeführt. Zuerst werden die Zeilen gesucht, welche gültige GPS-Daten enthalten. Die GPS-Daten werden dann in Variablen festgehalten und allen folgenden Zeilen zugeordnet, bis ein neuer gültiger GPS-Datensatz gefunden wurde. Wie in Abschnitt 4.2 beschrieben enthält die Zeile der GPS-Daten den Bezeichner *GPRMC*. Anhand der National Marine Electronics Association (NMEA)-Dokumentation über den GPRMC-Standard können aus den codierten Informationen die einzelnen Parameter extrahiert werden. Anhand des Codebeispiels aus 4.1 soll das Zerteilen der GPS-Daten erläutert werden:

```
$GPRMC,102124.00,A,4847.55609,N,00935.87220,E,10.706,281.97,220220,,,A*5F
```

Zuerst wird die Zeile an den Kommata aufgetrennt, anschließend werden die einzelnen Zahlen und Zeichen weiter behandelt. Aus der ersten und letzten Zahlenkombination (`102124.00` und `220220`) lässt sich die Uhrzeit und das Datum zusammensetzen. Es han-

delt sich um den **22. Februar 2020, 10:21** Uhr und **24.00** Sekunden UTC-Zeit. Die UTC-Zeit soll nach Möglichkeit in die lokale Uhrzeit überführt werden. Wird der Radmesser nur innerhalb einer Zeitzone verwendet, kann die Zeitkorrektur anhand einer Variable und unter Berücksichtigung des Datums (Sommer-/Winterzeit) erfolgen. Wird der Radmesser international in verschiedenen Zeitzonen verwendet, ist auf die in Abschnitt 5.4.2 beschriebene Abfragemöglichkeit zurückzugreifen. Weiterhin ist angegeben, dass die Daten gültig sind, dies wird durch den Buchstaben A gekennzeichnet. Sind die Daten ungültig, wird ein V an dieser Stelle zurückgegeben. Daraufhin folgt die geografische Breite in Grad, Minuten und Dezimalminuten und die Auskunft ob es sich um die Nord- oder die Südhalbkugel handelt. Es schließt sich die Länge an, mit der Angabe Ost bzw. West. Um im Folgenden die GPS-Daten darstellen zu können bzw. mit ihnen rechnen zu können werden die Positionsangaben in Dezimalgrade umgewandelt. Die Grad-Angabe, wird durch Division der Angabe des GPS-Moduls mit 100 und Abtrennen der Nachkommastellen erreicht (im Beispiel **4847.55609** → **48** und **00935.87220** → **9**). Der verbleibende Rest wird durch die Division mit 60 in die Nachkommastellen der Dezimalgrade umgewandelt (**47.55609**/60 → **0.7926015** und **35.87220**/60 → **0.59787**). Durch Addition der zuerst bestimmten Grad-Angabe und der danach errechneten Nachkommastellen wird so eine vollständige Angabe in Dezimalgraden erreicht (**48 + 0.7926015** = 48.7926015 bzw. **9 + 0.59787** = 9.59787). Ist die angegebene Position auf der Südhalbkugel bzw. westlich des Nullmeridians, wird die Breitenangabe bzw. die Längenangabe zusätzlich mit einem negativen Vorzeichen versehen. Die **Geschwindigkeit** ist in Knoten angegeben und wird zur besseren Verständlichkeit in Kilometer pro Stunde umgerechnet. Als letzte Angabe erfolgt die Auskunft über die Fahrtrichtung, bezogen auf Norden.

**Aufteilen der Messdaten nach Fahrten** Eine Messfahrt gilt für die Auswertungssoftware als beendet, wenn ein neues GPS-Datum mit einer Uhrzeit gefunden wird, welche um mindesten 110 Sekunden vom vorherigen GPS-Datum, abweicht. Gilt eine Fahrt als beendet, werden die bis zu diesem Zeitpunkt gesammelten Messwerte mit der unten beschriebenen Software ausgewertet. Danach werden wieder alle Messwerte gesammelt, bis die nächste Messfahrt als beendet gilt und diese Daten werden wieder getrennt ausgewertet.

**Glätten und Ausbessern der Messwerte** Um die Messwerte weiterverwenden zu können, werden Messwerte, welche außerhalb des Messbereiches liegen entfernt. Als untere Grenze gilt die Breite des Lenkers zuzüglich der Breite des Außenspiegels ( $l_{Untergrenze} = 0,5\text{ m}$ ) und als obere Grenze die maximale Messdistanz eines Ultraschallsensors ( $l_{Obergrenze} = 12\text{ m}$ ). Bei Messreihen ist es stets möglich, dass ein einzelner Messwert stark von den anderen Messwerten abweicht. Weicht nur ein Messwert vom Mittel der gesamten Messreihe ab, handelt es sich bei diesem um einen Messfehler.

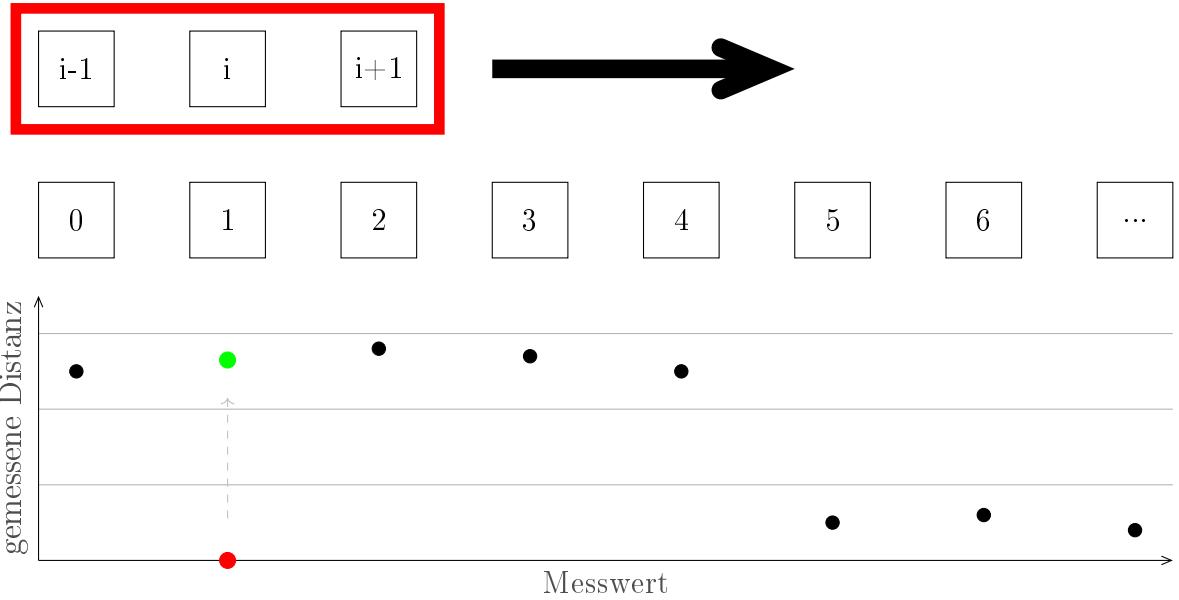


Abbildung 7: Graphische Darstellung des Glättungsalgorithmus mit Beispielabstandsdaten im unteren Bereich - der in Rot dargestellte Messwert wurde durch den grünen korrigierten Messwert ersetzt

Der Algorithmus zur Erkennung und Ausbesserung der Messfehler ist zur besseren Verständlichkeit in Abb. 7 skizziert und mit visualisierten Beispielabstandsdaten unterlegt. Um abweichende Messwerte zu erkennen, werden immer drei Messwerte ( $i - 1, i, i + 1$ ) betrachtet (rot eingerahmt). Mit der Variablen  $i$  wird nun durch alle Messwerte (0, 1, 2, ...) des Datensatzes gezählt. Weicht der Messwert  $i$  von den beiden Messwerten  $i + 1$  und  $i - 1$  um mindestens 10% nach oben oder unten ab, wird er durch den Mittelwert der Messwert  $i + 1$  und  $i - 1$  ersetzt. Im dargestellten Beispiel (Abb. 7) wird der rot eingezeichnete Abstandswert als Messfehler erkannt und zum grünen Abstandswert korrigiert. Durch die Implementierung des Glättungsalgorithmus wird verhindert, dass Überholvorgänge ausgeglättet werden, wie sie ab Messwert Nummer 5 zu sehen sind. Dies wurde durch Tests validiert.

## 5.2 Erkennen eines Überholvorgangs

In Abb. 5 sind Beispilmesswerte eines Überholvorgangs dargestellt. Die dort dargestellten Messwerte sind ideale Messwerte. In der Realität ist stets mit Abweichungen zu rechnen, welche durch die Eingangsdatenbearbeitung eliminiert werden sollen (Abschnitt 5.1). Die erste Änderung, mit welcher ein Überholvorgang beginnt, besteht darin, dass die vom hinteren Ultraschallsensor gemessene Distanz deutlich geringer ist, als die des vorderen Ultraschallsensors. Zudem muss der durch den hinteren Ultraschallsensor gemessene Abstand kleiner sein, wie ein vorher definierter Maximalabstand. Erst wenn der gemessene Abstand kleiner  $l_{U_{max}} = 3,50 \text{ m}$  ist, handelt es sich um einen Überholvorgang, zwar kann

auch in einem noch größeren Abstand überholt werden, davon ist im Normalfall aufgrund der Straßenverhältnisse nicht auszugehen (genaue Erläuterung siehe Anhang 9.4). Im Anhang 9.4 wurde eine maximale Fahrbahnbreite von  $l_{F_{max}} = 7\text{ m}$  definiert. Unter der Annahme, dass ein PKW eine Breite von  $l_{B_{PKW}} = 1,90\text{ m}$  aufweist, ein LKW bzw. Bus eine Breite von  $l_{B_{Schwer}} = 2,55\text{ m}$  und ein Fahrradfahrer eine Breite von  $l_{B_{Fahrrad}} = 0,60\text{ m}$ , wobei jedem Fahrzeug ein Bewegungsspielraum von mindestens  $l_{Regel_{min}} = 0,25\text{ m}$  eingeräumt werden muss [39], entsteht folgende Rechnung für den maximalen Überholabstand:

$$l_{U_{max}} = l_{F_{max}} - l_{B_{PKW}} - l_{B_{Fahrrad}} - 4 \cdot l_{Regel_{min}} = 7\text{ m} - 1,90\text{ m} - 0,60\text{ m} - 4 \cdot 0,25\text{ m} = 3,50\text{ m} \quad (5.1)$$

Aufgrund des in Abschnitt 3.5 berechneten Winkels der Ultraschallsensoren wird mindestens für ein Messwertpaar eine unterschiedliche Distanz von vorderem und hinterem Ultraschallsensor ermittelt. Ist die Differenzgeschwindigkeit zwischen KFZ und Fahrradfahrer jedoch nicht maximal ( $v_{KFZ\_Land} < 70 \frac{\text{km}}{\text{h}}$ ), werden für mehrere Messwertpaare ( $MWP_1$ ) unterschiedliche Distanzen ermittelt. Befindet sich das KFZ direkt neben dem Fahrrad (Abb. 4, Position 3) messen beide Ultraschallsensoren eine näherungsweise gleich große Distanz, welche aber kleiner dem Maximalabstand  $l_{U_{max}}$  ist. Je nach Geschwindigkeitsdifferenz zwischen Fahrradfahrer und überholendem KFZ werden unterschiedlich viele Messwertpaare mit gleich großer Distanz erfasst ( $MWP_2$ ). Die minimale Überholzeit wurde in Abschnitt 9.2 bestimmt und liegt bei  $t_{min} = 70\text{ ms}$ . Es muss ermittelt werden, wie viele Messungen innerhalb der minimalen Überholzeit durch die Ultraschallsensoren durchgeführt werden können. Entscheidend für die Dauer eines Überholvorgangs ist die Zeit  $t_{Schall}$ , welche der Ultraschall vom Radmesser zum KFZ und zurück benötigt. Die Rechenzeit des Mikrocomputers kann vernachlässigt werden, da dessen Prozessortakt bei 1 GHz liegt, was eine Rechenzeit von 1 ns pro Rechenschritt bedeutet. Als einfache Distanz, welche der Schall zurücklegen muss, kann  $l_{Schall} = 2\text{ m}$  angesetzt werden. Mit einer Schallgeschwindigkeit  $v_{Schall} = 340 \frac{\text{m}}{\text{s}}$  ergibt sich eine Zeit von:

$$t_{Schall} = 2 \cdot \frac{l_{Schall}}{v_{Schall}} = 2 \cdot \frac{2\text{ m}}{340 \text{ m s}^{-1}} = 11,76\text{ ms} \approx 12\text{ ms} \quad (5.2)$$

Es können demzufolge mindestens fünf Messwertpaare mit gleicher Distanz aufgezeichnet werden. Im Anschluss erfasst nur noch der vordere Ultraschallsensor das KFZ (Position 4, Abb. 5), während der hintere Ultraschallsensor wieder eine wesentlich größere Distanz misst. Es gilt das umgedrehte Prinzip, wie beim Einfahren in den Messbereich der Ultraschallsensoren. Abschließend sind die Messwerte beider Ultraschallsensor wieder den Werten vor dem Überholvorgang ähnlich und größer der maximalen Überholdistanz  $l_{U_{max}}$ .

### 5.2.1 Bestimmung der Geschwindigkeit des KFZ

Je nach Überholabstand, welcher durch die Ultraschallsensoren erfasst wird, ändert sich auch die Strecke  $l_1$ , auf der das KFZ nur vom hinteren Ultraschallsensor erfasst wird. Da die Ultraschallsensoren in einem voneinander abgewandten Winkel voneinander abgeneigt montiert sind (siehe Abschnitt 3.5), wird die Strecke, auf welcher ein KFZ nur von einem Ultraschallsensor erfasst wird proportional mit dem Überholabstand größer. Aus den gemessenen Abstandswerten  $l_{Mess}$  wird zuerst die Strecke  $l_1$  ermittelt, auf welcher das KFZ nur vom hinteren Ultraschallsensor erfasst wird (5.3).

$$l_1 = l_{Mess} \cdot \sin(\alpha) \quad (5.3)$$

Durch die Zeitangabe, welche jeder Messung zugeordnet ist, kann die Zeit  $t_1$  ermittelt werden, welche das KFZ zum Durchfahren der Strecke  $l_1$  benötigt. Aus der Zeit und der Strecke wird die Geschwindigkeit  $v_{diff\_KFZ}$  des KFZ gegenüber des Fahrradfahrers annähernd berechnet. Mit der Geschwindigkeit des Fahrradfahrers, gegeben durch das GPS-Modul, wird die absolute Geschwindigkeit  $v_{KFZ}$  des KFZ bestimmt. Die Berechnung der Geschwindigkeit erfolgt mit begrenzter Genauigkeit, da die Durchführung einer Messung eine bis mehrere hundertstel Sekunden in Anspruch nimmt und das KFZ während einer der Messung den Messbereich verlassen hat. Der exakte Zeitpunkt kann mit den Messergebnissen nicht bestimmt werden.

### 5.2.2 Unterscheidung der Fahrzeugarten

Befindet sich das KFZ direkt links neben dem Fahrrad (Position 3, Abb. 5), wird die Zeit  $t_2$  ermittelt, welche das Fahrzeug mit seiner Länge  $l_{KFZ}$  benötigt, um am Fahrrad vorbei zu fahren. Es werden die jeder Messung zugeordneten Zeiten zur Bestimmung von  $t_2$  verwendet. Mit der zuvor berechneten zum Fahrradfahrer unterschiedlichen Geschwindigkeit des KFZ  $v_{diff\_KFZ}$  und der ermittelten Zeit  $t_2$  kann die Länge des KFZ berechnet werden.

$$l_{KFZ} = v_{diff\_KFZ} \cdot t_2 \quad (5.4)$$

Die ermittelte Länge  $l_{KFZ}$  ist mit großen Ungenauigkeiten behaftet, da sowohl die Zeit  $t_2$ , als auch die Geschwindigkeit  $v_{diff\_KFZ}$ , welche zur Berechnung verwendet werden stark quantisiert sind. Aus der Fahrzeulgänge wird anschließend auf die in Abschnitt 9.2 festgelegten Fahrzeugkategorien geschlossen. Die in Abschnitt 9.2 ermittelten maximalen Überholzeiten dienen als Faktor zur Verifikation der Überholvorgänge. Werden die dort beschriebenen Überholzeiten überschritten, wird der Überholvorgang entweder einer anderen Fahrzeugkategorie zugeordnet oder, falls die Überholzeit größer ist, als die maximale Überholzeit  $t_2 > t_{Strass\_max}$ , als Überholvorgang aussortiert.

## 5.3 Berechnung der zurückgelegten Strecke

Die gemessenen Überholvorgänge sollen auf einer soliden Basis stehen und nicht in den leeren Raum gestellt werden. Aus diesem Grund wird die Länge der Strecke ermittelt, auf welcher der Fahrradfahrer unterwegs war. Es ist dann eine Aussage darüber möglich, wie oft der Fahrradfahrer pro Kilometer überholt wurde und die gefahrene Gesamtstrecke gibt Auskunft über die Aussagekraft der bisher erzielten Ergebnisse. Die Bestimmung der zurückgelegten Strecke ist mithilfe der aufgezeichneten GPS-Koordinaten möglich. Die Daten liegen in einzelne Fahrten aufgetrennt vor, deshalb kann die Distanz zwischen zwei GPS-Koordinaten bestimmt werden und zur Gesamtdistanz für jede Fahrt aufaddiert werden. Durch das Auftrennen in einzelne Fahrten, ist es möglich, mit dem Radmesser an einer anderen Stelle eine neue Messfahrt zu starten, als die vorherige beendet wurde. Unter Verwendung des Satzes des Pythagoras kann aus der Änderung in nördliche bzw. südliche Richtung  $l_{Breite}$  und der Änderung in West/Ost  $l_{Länge}$  die Distanz zwischen den beiden Koordinaten  $l_{gefahren}$  bestimmt werden.

$$l_{gefahren} = \sqrt{{l_{Breite}}^2 + {l_{Länge}}^2} \quad (5.5)$$

Die Distanzen zwischen den GPS-Koordinaten sind sehr klein bezogen auf den Erdradius, deshalb muss die Erdkrümmung in der Berechnung nicht berücksichtigt werden. Zwischen zwei Breitengraden liegt stets eine Entfernung von 111,3 km. Die Entfernung zweier Längengrade ist jedoch vom Breitengrad abhängig. Am Äquator beträgt sie 111,3 km, während sie am Nordpol auf 0 m schrumpft. Die Berechnung des Abstands der Längengrade erfolgt mit Hilfe des Kosinus in Abhängigkeit des Breitengrades.

Wir nehmen zur Beispielrechnung zwei GPS-Koordinaten mit Startpunkt:  $\alpha_{Länge\_Start}$  und  $\alpha_{Breite\_Start}$  und Endpunkt:  $\alpha_{Länge\_Ende}$  und  $\alpha_{Breite\_Ende}$ . Die Entfernung bei den Breitengraden lässt sich direkt bestimmen zu:

$$l_{Breite} = 111,3 \text{ km} \cdot (\alpha_{Breite\_Ende} - \alpha_{Breite\_Start}) \quad (5.6)$$

während für die Entfernung der Längengrade die Breitengrade berücksichtigt werden:

$$l_{Länge} = 111,3 \text{ km} \cdot \cos(\alpha_{Breite\_Ende}) \cdot (\alpha_{Länge\_Ende} - \alpha_{Länge\_Start}) \quad (5.7)$$

Die Entfernung der Längengrade ist am Startpunkt verschieden zu der am Endpunkt. Da die Koordinaten jedoch sehr nahe beieinander liegen ( $\alpha_{Breite\_Ende} - \alpha_{Breite\_Start} < 0,005^\circ$ ), liegt die Abweichung bei 0,094 %. Deshalb ist die in (5.3) vorgenommen Vereinfachung, dass lediglich mit der Breitenangabe der Endkoordinate der Abstand zwischen den Längengraden errechnet wird, zulässig.

## 5.4 Auswertung und Visualisierung der Daten

Nach der Extraktion der Überholvorgänge aus den aufgezeichneten Daten, müssen diese aufbereitet werden. Eine lange Liste bestehenden aus Überholabstand, GPS-Koordinaten, Uhrzeit, Datum und Geschwindigkeit hat keine Aussagekraft. Die Auswertung kann nach verschiedenen Grundsätzen erfolgen. Zum einen kann eine Analyse und Visualisierung nach Regionen erfolgen, indem definierte Bereiche auf einer Karte eingefärbt werden, bzw. die Überholvorgänge als einzelne Punkte auf einer Karte eingezeichnet werden. Zum anderen können die Überholvorgänge nach der Adresse, der Uhrzeit oder der Geschwindigkeit des Fahrrads bzw. des KFZ sortiert werden und dann in Form eines Balken- oder Säulendiagramms dargestellt werden.

Im Folgenden wird zuerst auf die Möglichkeit der Visualisierung mithilfe einer Karte eingegangen und erläutert, wie aus GPS-Koordinaten eine Adresse erzeugt werden kann. Im Anschluss werden Auswertungs- und Darstellungsmöglichkeiten mittels Diagrammen erörtert. Zum Zeitpunkt der Arbeit standen nicht ausreichend Überholvorgänge zur Verfügung, um die Auswertungs- und Visualisierungsmöglichkeiten vollständig anzuwenden.

### 5.4.1 Dateiformate zur Darstellung von GPS-Koordinaten

Sollen die Überholvorgänge auf einer Karte dargestellt werden, stehen mehrere Dateiformate zur Auswahl:

- Geo-JavaScript Object Notation (GeoJSON)
- GPS Exchange Format (GPX)
- Geography Markup Language (GML)
- Keyhole Markup Language (KML)

Die Dateiformate GPX, GML und KML basieren auf der XML-Syntax (Extensible Markup Language), während GeoJSON auf der JSON-Syntax aufbaut (JavaScript-Object-Notation).

GeoJSON bietet die Möglichkeit verschieden geometrische Objekte (Punkte, Linien und Polygone) darzustellen und ihnen Eigenschaften zu zuweisen, dabei ist das JavaScript-Object-Notation (JSON)-Format gut für den Menschen lesbar. Festgelegt wird das GeoJSON-Format in einem empfohlenen Standard der Internet Engineering Task Force [40]. Es gibt eine große Zahl von Internetseiten, welche eine einfache Visualisierung einer GeoJSON-Datei ermöglichen, beispielsweise werden alle auf GitHub abgelegte GeoJSON-Dateien automatisch auf einer Karte dargestellt [41].

Im GPX-Format besteht die Möglichkeit, Wegpunkte darzustellen, Routen zu planen und Tracks aufzuzeichnen. Im Vergleich zum GeoJSON-Format können hier mehr Eigenschaf-

ten den einzelnen Punkten zugeordnet werden. Es existieren mehrere Programme zur Visualisierung der Daten. [42]

Das GML-Format stellt neben der Darstellungsmöglichkeit von einfachen geometrischen Formen, die Möglichkeit komplexe geometrische Formen und nicht geometrische Daten zu speichern. Hierbei übersteigt die Komplexität des Datei-Formats jedoch deutlich die der zuvor präsentierten Formate. [43]

Mit Hilfe von KML können sowohl einfache als auch komplexe geometrische Objekte dargestellt werden. Da KML und GPX auf XML basieren besteht hier teilweise die Möglichkeit zur einfachen Konvertierung. Besonders attraktiv wird KML durch die Schnittstelle, welche Google Earth zur Verfügung stellt. KML-Dateien, welche 3D-Objekte, Flächen bzw. Linien enthalten können bei der Darstellung an die Erdoberfläche angepasst werden. [44]

Die Entscheidung für ein (oder mehrere) Dateiformat ist von der gewünschten Visualisierung abhängig. Besonders attraktiv ist die Nutzung von KML und GeoJSON, da diese durch einfaches Hochladen der Datei auf eine bekannte Plattform visualisiert werden. Eine einfache Online-Visualisierung wird angestrebt, um Benutzern und Interessenten einen schnellen Überblick zu ermöglichen. Wie bereits in Abschnitt 5.4.1 erläutert stellt GitHub GeoJSON-Dateien automatisch als Karte dar. KML kann in Google Earth (nicht zu wechseln mit Google Maps) hochgeladen und visualisiert werden. Grundlegend kann auch das GPX-Format zur Visualisierung verwendet werden. Es existiert jedoch keine einheitlich und weit verbreitete Software, welche eine einfache Online-Visualisierung bereitstellt.

#### 5.4.2 Analyse der besonders betroffenen Gebiete

Um analysieren zu können, in welchen Gebieten besonders eng überholt wird, gibt es zwei Auswertungsmöglichkeiten. Zum einen können die Überholvorgänge anhand ihrer GPS-Koordinaten in ein fest definiertes Raster eingeordnet werden, welches über die Karte gelegt wird. Es entstehen Quadrate mit einer Seitenlänge von beispielsweise 50 m. Jeder Überholvorgang, welcher aufgrund seiner Koordinaten diesem Quadrat zugeordnet werden kann, wird dort aufgelistet. Es besteht jedoch der Nachteil, dass besonders bei engen Straßenverhältnissen (Kreuzungen und Parallelstraßen) Überholvorgänge nicht genau oder falsch zugeordnet werden. So kann es sein, dass zwei Gefahrenstellen innerhalb eines Rasterquadrates liegen und so die Überholvorgänge nicht eindeutig einer Gefahrenstelle zugeordnet werden können. Durch einen Straßenverlauf in ländlichen Regionen kann es zu einer Verwässerung der Ergebnisse kommen, da eine einzelne Straße mit vielen Überholvorgängen mit Feldern zur Berechnung des Durchschnitts herangezogen wird.

Die zweite Möglichkeit zur Analyse der Überholvorgänge nach Regionen besteht darin, aus den GPS-Koordinaten Adressen (Straße, Postleitzahl, Stadt) zu generieren. Diese

Informationen müssen extern bezogen werden und können nicht selbst erfasst bzw. programmiert werden. Hierfür wird ein Application Programming Interface (API) verwendet, welchem die GPS-Daten übergeben werden und welches danach eine Adresse zurückliefert. Bei der Auswahl einer geeigneten API stehen zwei Punkte im Vordergrund, zum einen eine einfache Handhabung, zum anderen ein möglichst geringer Preis. Im Folgenden soll auf die API von *Google Maps*, *Microsoft Azure* und *OpenCage Geocoder* eingegangen werden. Besonders im Aspekt der Preisgestaltung sind starke Unterschiede zu erkennen. Das Preissystem von *Microsoft Azure* ist kaum verständlich und umfasst weit mehr Services, als tatsächlich benötigt. *Google Maps* rechnet pro API-Abfrage ab - zu einem Preis von 5\$/1000 Abfragen [45]. Von *OpenCage Geocoder* wird eine kostenlose Testversion der API angeboten, mit welcher bis zu 2500 Abfragen pro Tag möglich sind. Sind mehr Abfragen pro Tag notwendig, kann dies durch eine monatliche Zahlung freigeschaltet werden [46]. Die API von *OpenCage* schneidet beim Preis pro Abfrage am besten ab. Weiterhin bestehen für die *OpenCage*-API bereits mehrere Bibliotheken, um die Abfrage direkt aus einem Programm heraus auszuführen. Um statistische Auswertungen einfach zu implementieren, empfiehlt sich die Programmiersprache R, welche als freie Software zur Verfügung steht. In R können statistische Berechnungen durchgeführt werden und aus den Ergebnissen Grafiken erstellt werden. Eine Bibliothek für den Zugriff auf die *OpenCage*-API steht zur Verfügung. [47]

**Zeitkorrektur** Wie bereits oben beschrieben liegt die Zeit, welche durch das GPS-Modul gegeben ist, als UTC-Zeit vor. Um einen sinnvollen Umgang mit der Zeitangabe zu realisieren, muss diese in die lokale Uhrzeit transformiert werden. Mit Hilfe der API-Abfrage wird anhand der Koordinaten die Verschiebung gegenüber der UTC-Zeit ermittelt. Durch einfache Addition kann so aus der UTC-Zeit und der Verschiebung die lokale Zeit bestimmt werden.

#### 5.4.3 Auswertung nach weiteren Parametern

Auf einer Karte können lediglich Gebiete markiert werden, in welchen oft besonders eng überholt wird. Wertet man die Daten nicht nur über eine Karte aus, kann differenziert dargestellt werden, wie viele Überholvorgänge mit welchem seitlichen Abstand durchgeführt werden. Über die oben beschriebene API können Informationen über die Straße bezogen werden, auf welcher der Überholvorgang stattfand. Es stehen Informationen über den Straßentyp zu Verfügung (Hauptstraße, Nebenstraße, Wohnstraße ...), wie sie auf OpenStreetMaps hinterlegt sind, die maximal erlaubte Geschwindigkeit und die Adresse, bestehend aus Postleitzahl, Stadtnamen und Land. Anhand der Informationen, welche über die Straßenverhältnisse (Straßentyp und erlaubte Geschwindigkeit) abgefragt wurden, kann analysiert werden, welchen Einfluss diese Parameter auf den seitlichen Abstand beim

Überholen haben. Werden die Überholvorgänge nach einzelnen Adressen sortiert, können besonders exponierte Gefahrenstellen erkannt werden. Durch eine genauere Analyse dieser Stellen kann ermittelt werden, welche Infrastruktur zu besonders geringen Überholabständen führt. Wird nach Städten und Postleitzahlen ausgewertet, kann ermittelt werden, ob es regionale Unterschiede gibt, welche auf unterschiedliche Verkehrsmentalitäten der Bevölkerung zurückzuführen sind.

#### **5.4.4 Weitere Auswertungsmöglichkeiten der GPS-Daten**

Die GPS-Daten werden in den bisher beschriebenen Auswertungsverfahren nur im Zusammenhang mit den Überholvorgängen ausgewertet. Um die Überholvorgänge besser einordnen zu können, soll aus den ermittelten GPS-Tracks Folgendes analysiert werden:

- Länge der gefahrenen Strecke innerhalb einer definierten Region
- Wie oft wurde welche Straße befahren
- In welcher Richtung wurde die Straße durchfahren
- Wie schnell war der Fahrradfahrer auf einer bestimmten Straße unterwegs
- Wann wurde besonders viel Fahrrad gefahren

Aus den erhaltenen Daten kann nun Folgendes abgeleitet werden:

- Wie oft wurde pro Kilometer in einem bestimmten Bereich überholt
- Wie oft wird man im Durchschnitt auf der jeweiligen Straße überholt
- In welcher Richtung wird man auf der jeweiligen wie oft überholt
- Wann wird man besonders häufig überholt

# 6 Tests

Das einwandfreie Funktionieren des Radmessers und der Auswertungssoftware wird mit Tests verifiziert, welche definiert, durchgeführt und ausgewertet werden. Im ersten Schritt werden die Ultraschallsensoren genauer untersucht, bevor die Radmesser Software getestet wird. Die Auswertungssoftware muss zur Korrektur von Ungenauigkeiten nach den Tests des Radmessers angepasst werden. Die Tests der Auswertungssoftware basieren auf den vom Radmesser aufgezeichneten Daten.

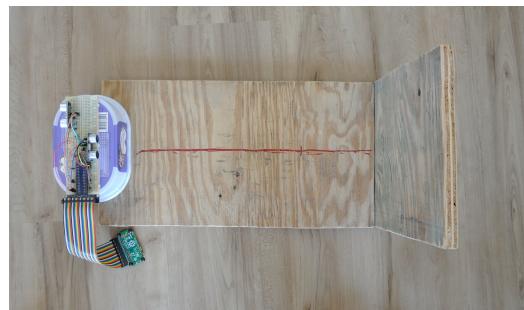
## 6.1 Radmesserhardware

### 6.1.1 Messwerte bei variabler Luftfeuchte

**Testentwicklung** Während die Temperaturabhängigkeit der Schallgeschwindigkeit in vielen Literaturquellen erläutert wird, gibt es nur wenige Aussagen über deren Abhängigkeit von der Luftfeuchtigkeit. Da die Luftfeuchtigkeit je nach Einsatzort und Wetterlage stark schwanken kann, soll diese Abhängigkeit im Folgenden genauer untersucht werden.



(a) Testaufbau Seitenansicht



(b) Testaufbau Draufsicht

Abbildung 8: Testaufbau Radmesserhardware

Ein Ultraschallsensor wird, wie in Abb. 8 zu sehen, in einem Abstand von 47,5 cm zu einer Wand montiert. Die Messungen werden mithilfe der Radmesser-Software durchgeführt und aufgezeichnet. Die Luftfeuchtigkeit wird durch das Einsprühen von fein zerstäubtem Wasser variiert. Das Wasser ist rund 10 K wärmer als die Luft, in welche es eingesprührt

wird, da beim Verdampfen des Wassers der Luft Verdampfungsenergie in Form von Wärme entzogen wird. Durch das wärmere Wasser wird versucht, die Temperaturabweichung möglichst gering zu halten. Die Messreihe wurde durch eine Software ausgewertet, welche den mittleren gemessenen Abstand ermittelt und die Zahl der Messungen, welche vom Mittelwert abweichen dokumentiert.

**Testergebnis** Die Änderung der Luftfeuchte selbst konnte nicht genau bestimmt werden, da diese nur im Schallbereich der Ultraschallsensoren variiert wurde. Eine signifikante Änderung der Schallgeschwindigkeit konnte jedoch nicht festgestellt werden. Im Folgenden soll nun auf theoretischen Modellen basierend versucht werden die Messergebnisse zu validieren.

**Erläuterung der Ergebnisse** Auf Grundlage der allgemeinen Formel der Schallgeschwindigkeit in Gasen (6.1) kann deren Abhängigkeit von der Luftfeuchtigkeit erläutert werden.

$$c_{Schall\_Gas} = \sqrt{\kappa_{Gas} R_{Gas}} \cdot \sqrt{T} \quad [48] \quad (6.1)$$

Der Adiabatenexponent  $\kappa_{Gas}$  kann näherungsweise aus der Zahl der Freiheitsgrade, der Gas-Moleküle bestimmt werden. Die Freiheitsgrade nehmen mit der zunehmenden Anzahl der Atome innerhalb des Moleküls ab. Für einatomige Gase gilt  $\kappa_{Gas_{1-atomig}} = 1,66$ , für zweiatomige Gase gilt  $\kappa_{Gas_{2-atomig}} = 1,40$  und für dreiatomige Gase gilt  $\kappa_{Gas_{3-atomig}} = 1,33$ . [49]

Die Gaskonstante  $R_{Gas}$  lässt sich mit der Molzahl  $M_{Gas}$  und der universellen Gaskonstante  $R$  bestimmen. [48]

$$R_{Gas} = \frac{R}{M_{Gas}} \quad (6.2)$$

Bei Gasgemischen können folgende Formeln zur Berechnung der Gaskonstante

$$R_{Gas} = \sum_i y_i \cdot R_i \quad (6.3)$$

bzw. des Adiabatenexponent

$$\kappa_{Gas} = 1 + \frac{1}{\sum_i \frac{r_i}{\kappa_i - 1}} \quad (6.4)$$

verwendet werden. Hierbei stellt  $r_i$  den Volumenanteil des i-ten Gases dar, welcher sich mit  $r_i = \frac{V_i}{V_{ges}} = y_i \cdot \frac{M_{ges}}{M_i}$  errechnet.  $y_i = \frac{m_i}{m_{ges}}$  stellt den Massenanteil des i-ten Gases dar. [48]

Unsere Luft besteht zu 78 Vol.-% aus Stickstoff ( $N_2$ ), zu 21 Vol.-% aus Sauerstoff ( $O_2$ ) und zu rund 1 Vol.-% aus Argon ( $Ar$ ), weitere Spurengase werden vernachlässigt. Der Anteil von Wasserdampf ( $H_2O$ ) in der Luft wird in der Meteorologie als relative Luftfeuchte

angegeben. Die relative Luftfeuchte gibt an, wie viel Wasser bezogen auf die Sättigungskonzentration in der Luft enthalten ist. Durchschnittlich liegt die relative Luftfeuchte (in Deutschland) zwischen 40 % und 90 % bei Temperaturen zwischen  $-10^{\circ}\text{C}$  und  $30^{\circ}\text{C}$ , was eine absolute Luftfeuchte zwischen  $1 \frac{\text{g Wasser}}{\text{kg Luft}}$  und  $20 \frac{\text{g Wasser}}{\text{kg Luft}}$  ergibt. Die Werte der absoluten Luftfeuchte gehen direkt in den Massenanteil  $y_i$  über, welcher dann bei 0,001 bzw. 0,02 liegt. Zuerst werden der Adiabatenexponent und die Gaskonstante von Luft mit der angenommenen Zusammensetzung (ganz ohne Wasserdampf) berechnet, als Bezugsvolumen dient ein Kubikmeter. Danach wird bestimmt wie stark ein Wasseranteil diese Variablen beeinflusst. Durch die Dichte der einzelnen Gase (Dichte entnommen aus Quelle: [50]) kann mit den Volumina die Masse der einzelnen Gase berechnet werden.

$$m_{\text{O}_2} = \rho_{\text{O}_2} \cdot V_{\text{O}_2} = 1,429 \frac{\text{kg}}{\text{m}^3} \cdot 0,21 \text{ m}^3 = 0,3 \text{ kg} \quad (6.5)$$

$$m_{\text{N}_2} = \rho_{\text{N}_2} \cdot V_{\text{N}_2} = 1,251 \frac{\text{kg}}{\text{m}^3} \cdot 0,78 \text{ m}^3 = 0,976 \text{ kg} \quad (6.6)$$

$$m_{\text{Ar}} = \rho_{\text{Ar}} \cdot V_{\text{Ar}} = 1,784 \frac{\text{kg}}{\text{m}^3} \cdot 0,01 \text{ m}^3 = 0,018 \text{ kg} \quad (6.7)$$

Aus den Massen der einzelnen Gase wird die Gesamtmasse von  $m_{\text{Luft}} = 1,294 \text{ kg}$  bestimmt. Die berechneten Werte können nun in die Formel (6.3) eingesetzt werden, wobei sich die Gaskonstante  $R_i$  mit (6.2) berechnen lässt. Es ergibt sich für die Luft die Gaskonstante  $R_{\text{Luft-trocken}} = 290,64 \frac{\text{Nm}}{\text{kmol}\cdot\text{K}}$ . Der Adiabatenexponent wird mit der Formel (6.4), den Volumenanteilen  $r_i$ , welche durch die prozentuale Volumenzusammensetzung bereits gegeben sind und den oben aufgeführten Werten für  $\kappa$  zu  $\kappa_{\text{Luft-trocken}} = 1,402$  bestimmt. Zur Berechnung der Schallgeschwindigkeit in feuchter Luft nehmen wir als Referenzmasse  $m_{\text{Luft-feucht}} = 1 \text{ kg}$ , wovon 98% die oben beschriebene Luft ( $\text{N}_2$ ,  $\text{O}_2$ ,  $\text{Ar}$ ) sind und 2% Wasserdampf. Dies wird in Formel (6.3) eingesetzt. Mit der Gaskonstante für trockene Luft und Wasserdampf ( $R_{\text{Luft-trocken}}$ ,  $R_{\text{H}_2\text{O}}$ ) wird  $R_{\text{Luft-feucht}} = 294,07 \frac{\text{Nm}}{\text{kmol}\cdot\text{K}}$ . Zur Berechnung des Adiabatenexponenten wird das Volumenverhältnis aus der Massenzusammensetzung bestimmt. Für die trockene Luft ergibt sich ein Volumen von  $V_{\text{Luft-trocken}} = \frac{0,98 \text{ kg}}{1,294 \text{ kg m}^{-3}} = 0,7573 \text{ m}^3$  und für den Wasserdampf von  $V_{\text{Wasserdampf}} = \frac{0,02 \text{ kg}}{0,88 \text{ kg m}^{-3}} = 0,0227$ , was 97,09 Vol.-% bzw. 2,91 Vol.-% entspricht. Setzen wir diese beiden Werte in (6.4) ein, wird  $\kappa_{\text{Luft-feucht}} = 1,399$ . Um den Unterschied der Schallgeschwindigkeit zwischen feuchter und trockener Luft zu errechnen werden einmal die Werte für trockene Luft und einmal für feuchte Luft in (6.1) eingesetzt. Die Temperatur wird bei 273 K gehalten.

$$c_{\text{Luft-trocken}} = \sqrt{290,64 \frac{\text{Nm}}{\text{kg}\cdot\text{K}} \cdot 1,402 \cdot \sqrt{273 \text{ K}}} = 333,529 \frac{\text{m}}{\text{s}} \quad (6.8)$$

$$c_{Luft-feucht} = \sqrt{294,07 \frac{\text{Nm}}{\text{kg} \cdot \text{K}} \cdot 1,399 \cdot \sqrt{273 \text{ kg}}} = 335,132 \frac{\text{m}}{\text{s}} \quad (6.9)$$

Aus den beiden Schallgeschwindigkeiten lässt sich eine prozentuale Abweichung von rund 0,5 % ermitteln. Dies ist der maximale Fehler, welcher durch die Luftfeuchtigkeit entstehen kann. Die durchschnittliche zu erwartende Abweichung ist geringer, da ein Schwanken zwischen 0 % und 100 % relativer Luftfeuchte in unseren Breiten nicht zu erwarten ist.

Wie bereits aus Formel (6.1) ersichtlich ist, besteht kein Zusammenhang zwischen Schallgeschwindigkeit und Luftdruck. Es werden an dieser Stelle keine weiteren Untersuchungen angestrebt.

### 6.1.2 Messwerte bei variabler Temperatur

**Testentwicklung** Die Schallgeschwindigkeit ändert sich mit der Temperatur (6.1). Der Zusammenhang zwischen Lufttemperatur und Schallgeschwindigkeit soll mit den Ultraschallsensoren überprüft werden und die Auswertungssoftware, auf die ermittelte Abhängigkeit eingestellt werden. Es wird ein fester Abstand eingestellt (Vgl. Abb. 8). Durch die Variation der Temperatur  $T$  und dem Durchführen von Temperatur- und Abstandsmessungen werden der Testauswertungssoftware die benötigten Daten zur Verfügung gestellt.

**Testergebnis** Die durchgeführten Messungen haben gezeigt, dass eine Änderung der Temperatur einen signifikanten Einfluss auf die Schallgeschwindigkeit hat. Zur Ermittlung des Temperatureinflusses wurden  $N_1 = 25\,427\,139$  Messungen durchgeführt.  $N_2 = 24\,962\,204$  Messungen lieferten ein sinnvolles Messergebnis, d. h. die gemessene Distanz ist kleiner als die Maximaldistanz, welche mit den Ultraschallsensoren erfasst werden kann und größer als die Minimaldistanz. Dies ergibt einen Anteil von 98,17 % verwendbarer Messergebnisse. Um die Messergebnisse übersichtlich Visualisieren zu können, wurde der Mittelwert der Schallgeschwindigkeit innerhalb einer Temperaturspanne  $\Delta T$  ermittelt und gegenüber der Temperatur in Abb. 9 aufgetragen. Als Temperaturspanne wurde  $\Delta T_1 = 0,3 \text{ K}$  und  $\Delta T_2 = 1 \text{ K}$  verwendet. Bei Verwendung einer größeren Temperaturspanne werden Ausschläge reduziert.

**Genauigkeit der Messergebnisse** In Abb. 10 ist die Verteilung der Entfernungsmesswerte bei konstant angenommener Schallgeschwindigkeit und variabler Temperatur dargestellt. Es zeigt sich eine starke Abweichung der Messergebnisse bei einem Temperaturunterschied von  $\Delta T = 30 \text{ K}$ . Weiterhin ist ersichtlich, dass 95 % der Messergebnisse maximal 1 cm vom Mittelwert abweichen. Größere Abweichungen treten nur selten auf. Nur 1 % der Messwerte weicht über 10 % vom Mittelwert ab.

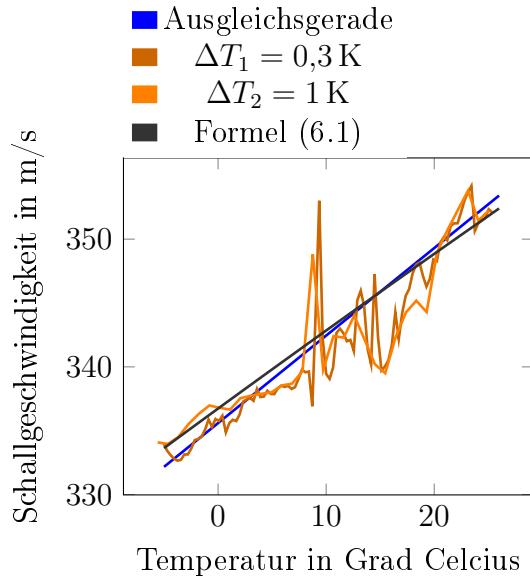


Abbildung 9: Schallgeschwindigkeit in Luft in Abhängigkeit der Temperatur

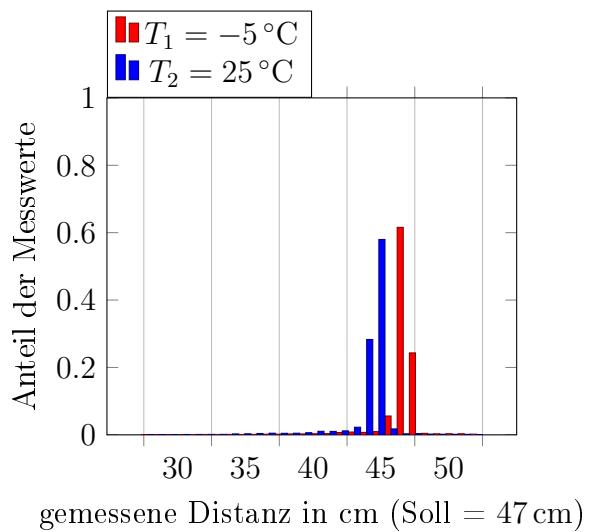


Abbildung 10: Genauigkeit der Messungen mit den Ultraschallsensoren, unter Annahme einer konstanten Schallgeschwindigkeit von 343 m/s

**Erläuterung der Messergebnisse** Die in Abb. 9 dargestellte Temperaturabhängigkeit der Schallgeschwindigkeit in Luft muss kompensiert werden, da sonst über die gesamte mögliche Verwendungstemperatur des Radmessers (zwischen  $-10^\circ\text{C}$  und  $35^\circ\text{C}$ ) eine Messabweichung von 10 % zu erwarten ist. In Abb. 9 ist sowohl eine Ausgleichsgerade eingezeichnet, wie auch der durch (6.1) beschriebene Zusammenhang. Es ist ersichtlich, dass durch die Verwendung einer Ausgleichsgerade ein kleiner Fehler ( $< 1\%$ ) entsteht, dieser ist vernachlässigbar. Werden die Messergebnisse temperaturkompensiert, ist eine sehr hohe Genauigkeit und für die Realisierung eines Radmessers ausreichende Präzision erreicht.

### 6.1.3 Messwerte bei externer Schalleinströmung

**Testentwicklung** Im Straßenverkehr liegt stets eine Geräuschkulisse vor. Je nach aktueller Situation kann der Schallpegel einzelner Geräusche, wie beispielsweise laute Musik, sehr hohe Werte annehmen. Um den Einfluss der externen Schalleinströmung in den Messbereich zu untersuchen, werden die Ultraschallsensoren auf eine feste Distanz eingestellt (Vgl. Abb. 8). Der Schall wird von verschiedenen Winkel und Positionen eingestreut. Zur Erzeugung von Schall wird ein Bluetooth-Lautsprecher verwendet (Sony SRS-XB21). Folgende Einstreupositionen sollen untersucht werden:

- Von der Seite ( $90^\circ$ )

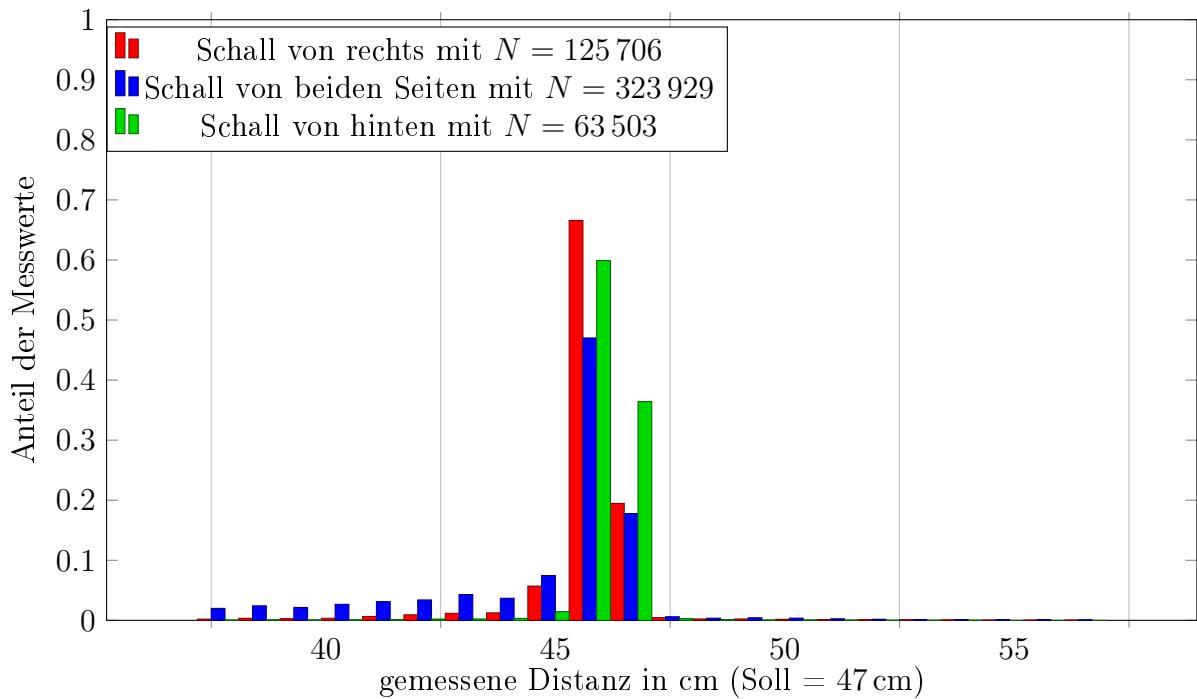


Abbildung 11: Einfluss von externer Schalleinstreuung auf die Präzision von Messungen mit Ultraschallsensoren, mit  $N$  als Zahl der durchgeführten Messungen

- Lautsprecher werden auf beiden Seiten des Ultraschallsensors platziert
- Lautsprecher steht hinter den Ultraschallsensoren

Es wird der in Abb. 8 dargestellte Aufbau zur geometrischen Feststellung der Ultraschallsensoren verwendet.

**Testergebnisse** Um den Einfluss externer Schalleinstreuung auf die Genauigkeit der Messergebnisse visualisieren zu können, wurde das Diagramm 11 mit demselben Algorithmus erstellt wie das in Abb. 10 dargestellte.

Es ist zu erkennen, dass durch eine Schallquelle, welche senkrecht zur Messrichtung der Ultraschallsensoren gerichtet ist keinen Einfluss auf die Genauigkeit der Messergebnisse hat. Werden zwei Schallquellen - auf jeder Seite des Ultraschallsensors eine - hinzugefügt, sinkt die Messgenauigkeit erkennbar ab. Lediglich 70 % der Messwerte weichen weniger wie 1 cm von der tatsächlichen Messgröße ab. Die Messwerte sind jedoch fast alle kleiner als die tatsächliche Distanz. Der Einfluss einer Schallquelle, welche in dieselbe Richtung, wie der Ultraschallsensor gerichtet ist, ist geringer als der Einfluss von zwei Schallquellen. Der Einfluss ist vergleichbar dem einer Schallquelle senkrecht der Messrichtung der Ultraschallsensoren.

Im Straßenverkehr ist mit einer großen Zahl von Schallquellen zu rechnen. Der im Versuchsaufbau beschriebene Fall, dass zwei Schallquellen sich direkt gegenüber und auf beiden Seiten des Ultraschallsensors befinden, stellt einen Extremfall dar und ist unwahr-

scheinlich. Die Wahrscheinlichkeit, dass von einer Schallquelle Schall eingestreut wird, ist sehr groß. Bei dieser Konstellation waren im durchgeföhrten Experiment keine signifikanten Ungenauigkeiten zu verzeichnen.

#### 6.1.4 Messwerte bei bewegten Luftmassen

**Testentwicklung** Im Straßenverkehr ist aufgrund des Wetters und vorbei fahrender KFZ mit bewegten Luftmassen zu rechnen. Inwiefern dies die Messgenauigkeit der Ultraschallsensoren beeinflusst, muss untersucht werden, um ggf. Gegenmaßnahmen zu ergreifen. Es wird wieder ein fester Abstand zwischen Messobjekt und Ultraschallsensor eingestellt (Vgl. Abb. 8). Mit Hilfe eines Kaltluftföhns wird seitlicher Wind simuliert. Wind aus der Richtung des Messobjekts ist nicht zu berücksichtigen, da der Wind dort bei einem Überholvorgang von einem KFZ abgehalten wird. Der Messaufbau (Abb. 8) wird zudem an einem windigen Tag ins Freie verbracht. Auf diese Weise wirken zufällige Windgeschwindigkeiten und Richtungen auf die Messung der Ultraschallsensoren ein.

**Testauswertung** Die Messungen wurden mit derselben Software, wie oben ausgewertet. Aus den in Abb. 12 dargestellten Daten, ist zu erkennen, dass bei böigem Wind noch knapp 90 % der Messwerte um maximal 1 cm vom tatsächlichen Wert abweichen.

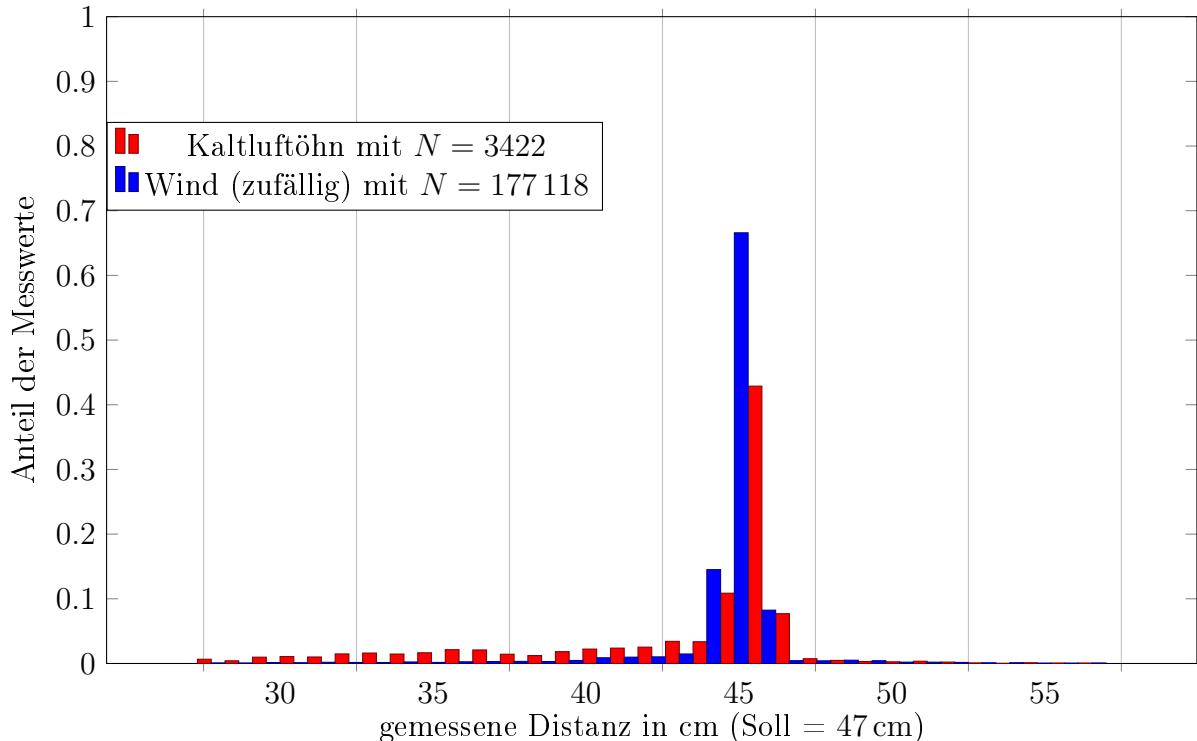


Abbildung 12: Einfluss von Wind auf die Präzision von Messungen mit Ultraschallsensoren, mit  $N$  als Zahl der durchgeföhrten Messungen

Wird mit einem Föhn die Luft auf der Messeinrichtung bewegt, verschlechtert sich die Genauigkeit der Messungen deutlich. Nur noch rund 60 % der Messungen weichen weniger als 1 cm vom Ist-Wert ab. Die Messwerte welche mehr als 1 cm abweichen, sind stets kleiner als der Ist-Wert. Die Datenbasis mit  $N = 3422$  Messungen ist jedoch gegenüber den anderen Messreihen klein, da der Föhn von Hand neben die Ultraschallsensoren gehalten werden musste. Da der Anteil der abweichenden Messungen sehr hoch ist, sind an dieser Stelle weiter Untersuchungen notwendig. Von einer Verwendung des Radmessers bei Sturm (-böen) ist abzusehen.

### 6.1.5 Messbereich der Ultraschallsensoren

**Testentwicklung** Da die Ultraschallsensoren nicht senkrecht zu einem vorbeifahrenden KFZ messen, sondern leicht nach vorne und hinten geneigt sind, muss untersucht werden, bis zu welchem Winkel die Ultraschallsensoren Objekte erkennen können und welchen Einfluss eine schräge Fläche auf die Messung hat. In den oben beschriebenen Messungen wurde stets die Entfernung zu einer planaren, senkrechten Fläche ermittelt. Um den Messbereich der Ultraschallsensoren zu ermitteln, wird ein Objekt in den in Abb. 8 dargestellten Messaufbau eingesetzt und dort auf verschiedenen Positionen platziert (Vgl. Abb. 13). Für die Analyse des Einflusses, welche eine schräge Fläche auf die Messgenauigkeit der Ultraschallsensoren hat, wird eine Fläche mit verschiedenen Winkel ( $\gamma$ ) und unterschiedlichen Abständen zu den Ultraschallsensoren platziert (Vgl. Abb. 14).

**Testergebnisse** Wie in Abb. 13 dargestellt, wurde ein Objekt mit unterschiedlichen seitlichen Abweichung  $l_b$  in den Messbereich der Ultraschallsensoren eingeführt. Jeder in Abb. 15 dargestellte Punkt basiert auf 40 000 bis 60 000 Messungen. Die gemessene Distanz stellt den Durchschnittswert der Messungen dar, bei welchen das Objekt erkannt wurde. Bei allen Messungen und Berechnungen kann es wie oben zu Quantisierungsfehlern kommen.

Im Anhang in Abschnitt 9.9 sind weitere, ergänzende Diagramme dargestellt.

Wie in Abb. 20 und Abb. 21 zu erkennen ist, sinkt die Erkennungsrate mit zunehmender seitlicher Abweichung  $l_b$  ab. In Abb. 15 sind die Auswirkungen der seitlichen Abweichung gegenüber dem Winkel  $\beta$  (Vgl. Abb. 13) dargestellt. Ein besonders starkes Abfallen der Erkennungsrate ist bei einem Winkel von  $\beta \geq 20^\circ$  zu erkennen. Es ist weiterhin ein unsymmetrisches Verhalten bei einer Abweichung nach links und nach rechts zu beobachten. Die gemessene Distanz hingegen steigt mit zunehmendem seitlichen Abstand an und weicht zunehmend von der tatsächlichen Messdistanz  $l_c$  ab, welche über den Satz des Pythagoras aus der neuen Messdistanz  $l_a$  und der seitlichen Abweichung  $l_b$  bestimmt wird ( $l_c = \sqrt{l_a^2 + l_b^2}$ ). Mit einer steigenden Messdistanz  $l_a$  steigt die absolute und relative

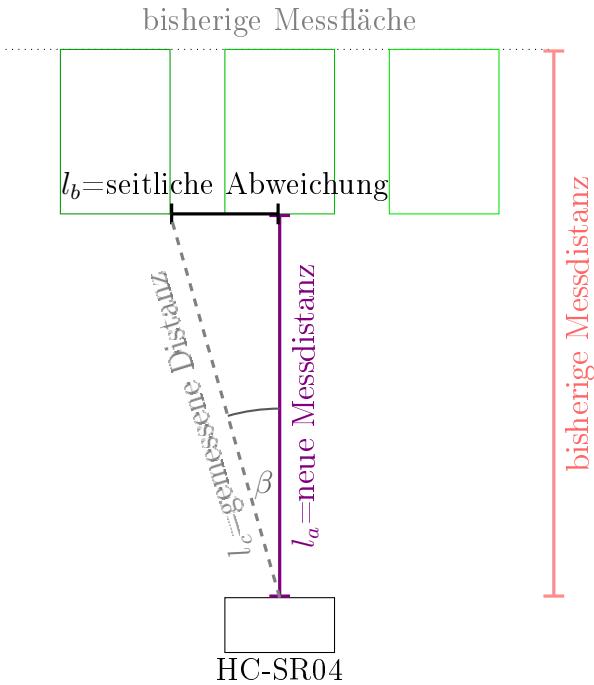


Abbildung 13: Messbereich der Ultraschallsensoren HC-SR04, in Farbe die eingesetzten Objekte

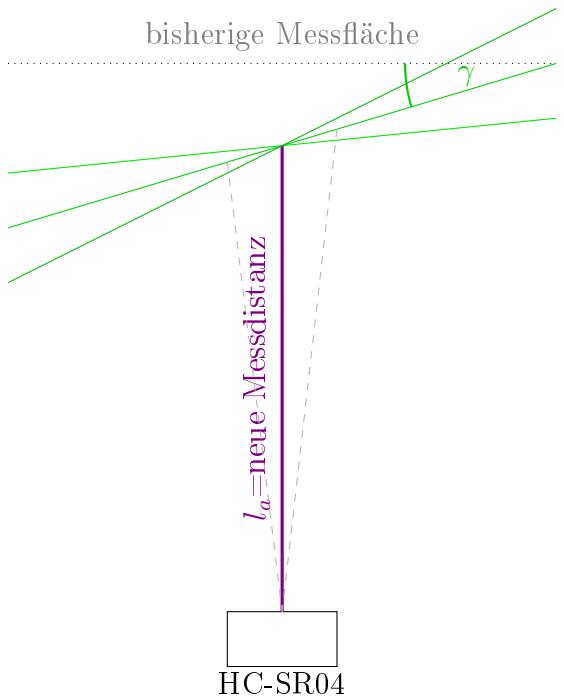


Abbildung 14: Messgenauigkeit bei schrägen Flächen, in Farbe die neuen Messflächen

Messabweichung. In Abb. 15 ist zu erkennen, dass bei einer Messdistanz  $l_a = 22\text{ cm}$ , die Messabweichung bereits bei einem Winkel  $\beta = 12^\circ$  stark ansteigt. Wird die Messdistanz auf  $l_a = 31,2\text{ cm}$  erhöht, bleibt die Messabweichung bis  $\beta = 19^\circ$  gering und näherungsweise konstant. Durch eine Positionierung des Objekts mithilfe eines Maßbandes sind auch Ungenauigkeiten bei den seitlichen Abständen zu berücksichtigen.

Bei einer schrägen Messfläche, wie in Abb. 14 dargestellt wurde der Winkel in  $10^\circ$ -Schritten erhöht und bei jeder Einstellung mindestens 40 000 Messungen durchgeführt. Die Messwerte wurden wie oben beschrieben auf die Erkennungsrate hin untersucht. Die Messabweichung wurde aus den Messungen, in welchen das Objekt erkannt wurde, bestimmt und in Abb. 16 dargestellt. Dabei fällt auf, dass bis zu einem Winkel von  $\gamma = 30^\circ$  bei über 80 % der Messungen das Objekt erkannt wird. Aus den Daten der Messabweichung ist zu erkennen, dass eine stärkere Schrägstellung ( $\gamma$  größer) zu kleineren gemessenen Distanzen führt und somit zu einer größeren (negativen) Messabweichung. Mit zunehmender Messdistanz  $l_a$  und einer konstant schrägen Messfläche ( $\gamma = \text{const}$ ) ist eine Zunahme der Messabweichung zu verzeichnen.

**Erläuterung der Messergebnisse** Von den Ultraschallsensoren breitet sich der Schall in eine Richtung aus. Durch Streuung wird mit zunehmender Entfernung auch ein grö-

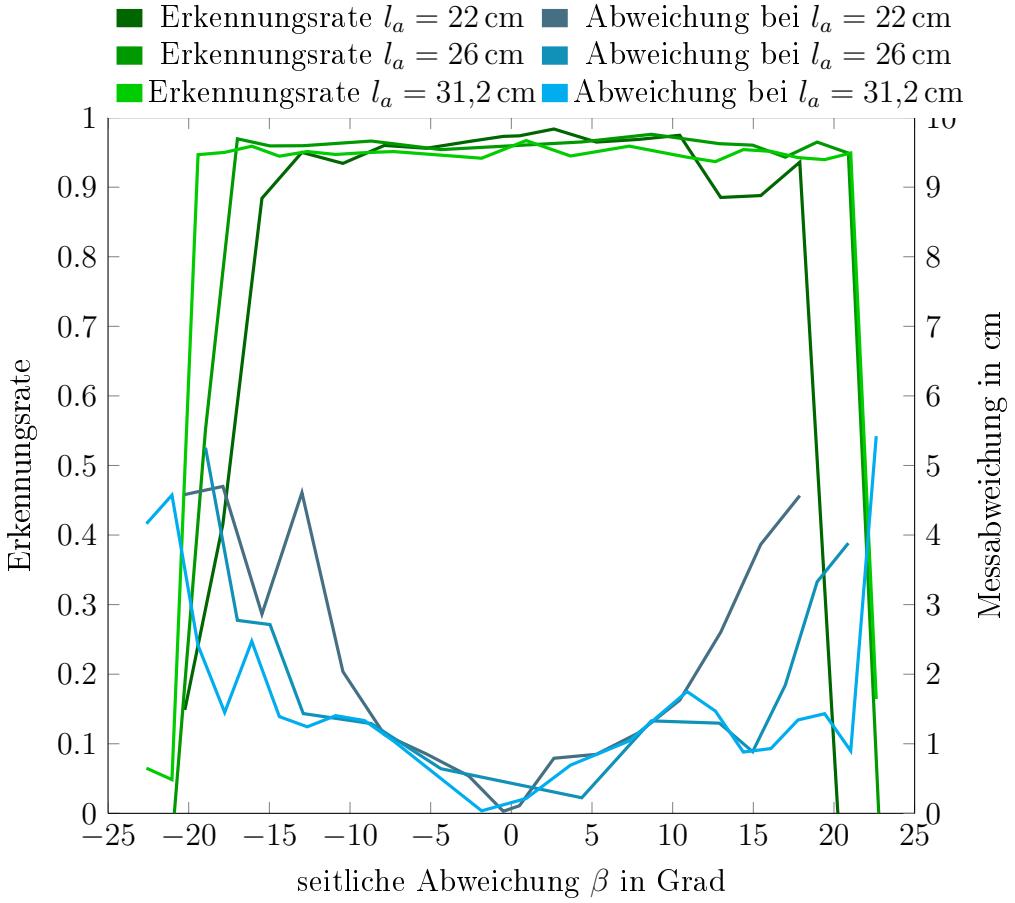


Abbildung 15: Messbereich der Ultraschallsensoren bei verschiedenen  $l_a$

ßerer Bereich erfasst.  $\beta = 19^\circ$  kann als maximaler Erkennungswinkel betrachtet werden. Der Erkennungswinkel stimmt mit dem von Singh und Borschbach ermittelten Winkel überein. Die Messdistanz ( $l_a$ ) lag bei den Tests in „Effect of external factors on accuracy of distance measurement using ultrasonic sensors“ einmal bei 10 cm und einmal bei 20 cm. Die Distanz, mit welcher die oben beschriebenen Tests durchgeführt wurden, ist größer. Weiterhin führt Singh und Borschbach nicht an, auf wie vielen Tests die Messergebnisse basieren. Wie in Abbildung 20 dargestellt und den Daten von Singh und Borschbach aufgezeigten Ergebnissen ist eine Asymmetrie zu beobachten, dass heißt, links werden andere Ergebnisse ermittelt wie rechts [27]. Die Asymmetrie kann auf den unsymmetrischen Aufbau der Ultraschallsensoren zurückgeführt werden, da Sender und Empfänger 27 mm in der Horizontalen voneinander entfernt liegen.

Steht die Messfläche nicht mehr senkrecht zu Messrichtung der Ultraschallsensoren, weichen auch die Messwerte von der Messdistanz  $l_a$  ab (Diagramm 22). Dies ist mit dem Messbereich der Ultraschallsensoren zu begründen, wie er in Abb. 14 dargestellt ist (grau). Mit zunehmendem Winkel  $\gamma$  rückt eine Seite der Messfläche näher an den Ultraschallsensor und wird deshalb auch früher erfasst, was zu einer geringeren gemessenen Distanz

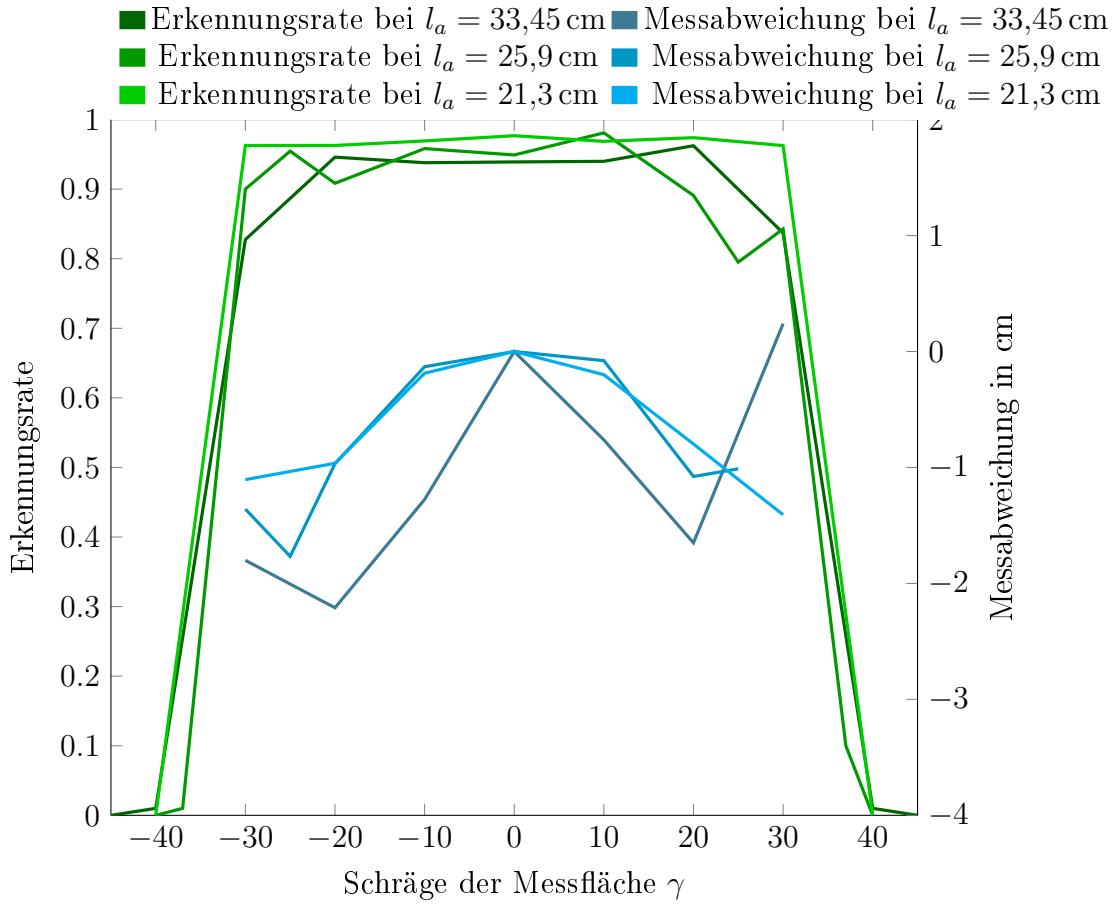


Abbildung 16: Genauigkeit der Messungen mit Ultraschallsensoren bei schrägen Reflexionsflächen

führt. Da mit zunehmender Messdistanz  $l_a$  die von den Ultraschallsensoren erfasste Fläche zunimmt, rückt ein Teil der Messfläche, welcher durch den Ultraschallsensor erfasst wird, näher an den Ultraschallsensoren. Aus diesem Grund steigt mit der Messdistanz  $l_a$  die Messabweichung an. Ab einem Winkel von  $\gamma = 30^\circ$  wird der Ultraschall nicht mehr zurück zum Ultraschallsensor reflektiert, sondern wird in eine andere Richtung reflektiert (Vgl. Reflexionsgesetz).

### 6.1.6 Weitere Tests der Ultraschallsensoren

#### Hindernismaterial

Von Adarsh et al. wurde experimentell der Einfluss des Materials des Reflexionsobjekts auf die Genauigkeit von Messungen mit Ultraschallsensoren untersucht. Die ermittelte Messabweichung ist besonders gering bei Schwämmen, Holz, Gummi und Dachziegeln. Lediglich bei Papier sind die Messabweichungen sehr hoch [30]. Die Messabweichungen können durch die einfache Schwingfähigkeit eines Blatt Papiers begründet werden. Bei

Metall, wie der Autokarosserie sind ähnliche Messabweichungen, wie bei Dachziegeln, Holz und Gummi zu erwarten.

In den oben durchgeführten Tests wurde als Hindernismaterial Holz oder ein Buch verwendet. Die Ergebnisse und ermittelten Abweichungen stimmen mit den von Adarsh et al. beschrieben Ergebnissen überein.

## 6.2 Radmessersoftware

Für den Radmesser ist es essenziell, dass die Software unter keinen Umständen stehen bleibt oder abbricht. Zudem muss ein möglichst großer Teil der Datensätze korrekt sein, da die gemessenen Situationen einmalig sind und eine Wiederholung der Verkehrssituation nicht möglich ist. Aus diesen Gründen sind ausführliche Tests notwendig.

Um die Funktionsfähigkeit der Radmesser Software auch unter vom Standard abweichen den Bedingungen zu verifizieren, reicht es aus zu zeigen, dass weiterhin Daten aufgezeichnet werden. Sollten diese nicht korrekt sein, müssen sie von der Auswertungssoftware als fehlerbehaftet erkannt werden und einer Sonderbehandlung unterzogen werden. Als Beleg für die Genauigkeit der Messergebnisse müssen mehrere Testreihen unter verschiedenen Bedingungen durchgeführt werden.

### 6.2.1 Langzeittest

**Testdefinition** Es muss ein Langzeittest mit dem Programm durchgeführt werden, um Fehler, welche erst während des Programmablaufs auftreten zu erkennen. Hierfür soll eine konstante Distanz gemessen werden, welche durch geometrisches Feststellen definiert wird. Das Sensor-System wird im Abstand von einem Meter von einer Wand entfernt platziert. Das Programm soll stets Messungen durchführen und deren Ergebnis aufzeichnen, ein Programmabbruch stellt einen fatalen Fehler dar.

**Testergebnis** Sowohl in dem oben beschriebenen Langzeittest, als auch bei allen anderen durchgeführten Messungen kam es nie zu einem Programmabbruch. In Abschnitt 6.1.2 wird genauer auf Abweichungen der gemessenen Distanz gegenüber der Ist-Distanz eingegangen.

### 6.2.2 Fehlende GPS-Daten

**Testdefinition** Das Programm muss auf sein Verhalten bezüglich fehlender GPS-Daten getestet werden. Zum einen kann es aufgrund von tiefen Häuserschluchten in Städten zu

einem Abbruch des GPS-Signals kommen, zum anderen kann das GPS-Modul aufgrund eines technischen Defekts ausfallen. Der Schwerpunkt liegt bei diesem Testszenario auf dem Fortlauf des Programms, um weiterhin Abstandsdaten ermitteln zu können. Um das Nicht-Vorhanden-Sein des GPS-Signals zu simulieren, wird der Radmesser innerhalb eines Gebäudes getestet, wo auch ein Smartphone kein GPS-Signal erhält. Der komplette Ausfall des GPS-Moduls wird durch ein physikalisches Trennen des GPS-Moduls vom Radmesser erreicht.

**Testergebnis** Das Nicht-Vorhanden-Sein von GPS-Daten stellt kein Problem für die Radmesser-Software dar. Das GPS-Modul meldet in diesem Fall von sich selbst zurück, dass die aktuellen Angaben ungültig sind (Vgl. Abschnitt 5.1). Ungültig gekennzeichnete GPS-Daten werden durch die Auswertungssoftware aussortiert. Auch ein Abtrennen des GPS-Moduls vom Radmesser hatte keinen Einfluss auf den Programmfortlauf, es wurden lediglich keine weiteren GPS-Daten gespeichert. Wenn es nicht zu einem vollständigen Ausfall des GPS-Moduls kommt und lediglich eine geringe Zahl an GPS-Daten fehlt, kann zwischen den vorhandenen GPS-Daten interpoliert werden. Auf einen Ausfall des GPS-Moduls ist der Nutzer mit Hilfe der LEDs zur Statusanzeige aufmerksam zu machen.

### 6.2.3 Abgedeckter Ultraschallsensor

**Testdefinition** Während des Fahrradfahrens kann es vorkommen, dass durch schlechte Gepäckbefestigung, lose Kleidungsstücke oder Verschmutzung einer oder beide Ultraschallsensoren abgedeckt sind und keine oder falsche Messwerte liefern. Es muss sicher gestellt werden, dass das Programm weiter läuft und nicht stehen bleibt. Zudem müssen die fehlerhaften oder nicht vorhandenen Messwerte in der Auswertungssoftware erkannt und aussortiert werden. Für den Test werden die Ultraschallsensoren mit einem Tuch abgedeckt.

**Testergebnis** Ist ein Ultraschallsensor - beispielsweise durch ein Kleidungsstück - abgedeckt, so läuft auch in diesem Fall die Software ordnungsgemäß weiter und liefert Messwerte zurück. Da die Messwerte entweder bei 12 m oder wenigen Zentimetern liegen, können diese durch die Auswertungssoftware aussortiert werden. Überholvorgänge werden nicht aufgrund des Abgedecktseins erkannt.

### 6.2.4 Unabhängige Funktion der Ultraschallsensoren

**Testdefinition** Anhand der unterschiedlichen Messwerte, welche die beiden Ultraschallsensoren ermitteln, sollen Überholvorgänge erkannt werden. Für das Erkennen der Über-

holtvorgänge ist es dabei essenziell, dass beide Ultraschallsensoren unabhängig voneinander Messungen durchführen (siehe Abb. 5). Um das unabhängige Arbeiten und Funktionieren der Sensoren zu identifizieren, wird ein Sensor vollständig mithilfe eines Klebebands abgedeckt, während die Werte des anderen auf ihre Plausibilität geprüft werden. Eine exakte Prüfung der Werte ist nicht erforderlich, da eine unabhängige Arbeitsweise bereits durch ein entsprechendes Abweichen der Messwerte nachgewiesen ist.

**Testergebnis** Wird ein Ultraschallsensor abgedeckt, misst der andere Ultraschallsensor - unabhängig davon - weiterhin seine Distanz. Für die Radmesser Software ist es unerheblich, wenn ein Ultraschallsensor komplett ausfällt. Wie in Abschnitt 4.3 beschrieben, wird nach 50 ms automatisch die nächste Messung initiiert. Liegen nur noch die Messergebnisse eines Ultraschallsensors vor, ist eine Auswertung durch die Auswertungssoftware nicht mehr möglich, diese ist auf die Messwerte beider Ultraschallsensoren angewiesen. Durch die LEDs für das Anzeigen des Betriebszustandes muss dem Nutzer im Falle des Ausfalls eines Ultraschallsensors signalisiert werden, dass ein Fehler vorliegt.

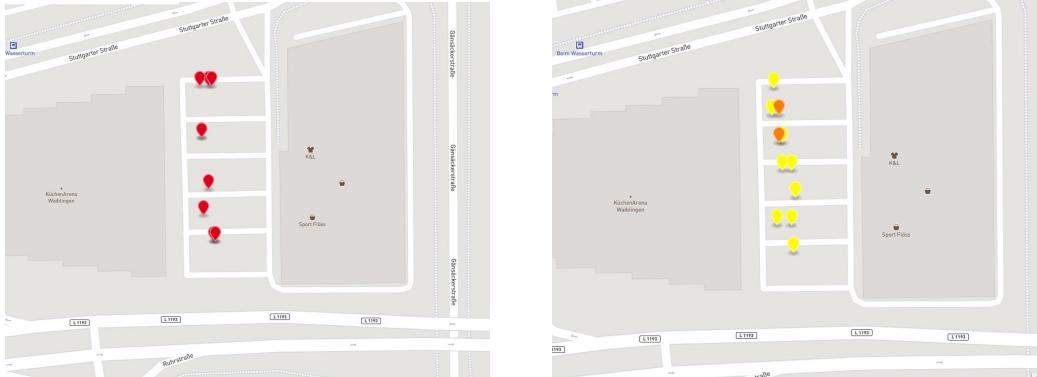
## 6.3 Auswertungssoftware

### 6.3.1 Erkennen von Überholvorgängen

**Testentwicklung** Die Auswertungssoftware wird mit gezielt aufgenommenen Messreihen getestet, bei welchen ein Fahrradfahrer in einem festgelegten Abstand überholt wurde. Um andere Einflussfaktoren auszuschließen und eine möglichst genaue Distanz zwischen KFZ und Fahrradfahrer einzuhalten werden die Überholvorgänge auf einem großen Parkplatz durchgeführt. Zu beachten ist, dass aufgrund von kleinen Lenkfehlern, von Auto- und Fahrradfahrer, und dem in Abschnitt 1.2.2 beschriebenen Lenkereinschlag des Radfahrers aufgrund des Kippens aus der Senkrechten heraus, mit Abweichungen von rund 10 % zu rechnen ist. Es ist zudem ein Test erforderlich, bei welchem die Auswertungssoftware mit Dateien getestet wird, welche keine Überholvorgänge enthalten. Auf diese Weise wird ausgeschlossen, dass die Software nicht zu viele Überholvorgänge extrahiert oder an Stellen einen Überholvorgang erkennt, wo keiner ist.

**Testergebnis** In Abb. 17 sind die durch die Auswertungssoftware erkannten Überholvorgänge dargestellt. Die Visualisierung wird wie in Abschnitt 5.4.1 als GeoJson-Datei von der Auswertungssoftware ausgegeben und kann auf der Internetseite [geojson.io](http://geojson.io) visualisiert werden.

Anhand der differenzierten Farbgebung der Überholvorgänge in Abb. 17 aufgrund des Überholabstands, ist ein schneller Überblick möglich. Bei der Analyse des exakten seitli-



(a) Überholabstand von 70 cm, auf *geoj-  
son.io* dargestellt

(b) Überholabstand von 120 cm, auf *geoj-  
son.io* dargestellt

Abbildung 17: Überholvorgänge durch die Auswertungssoftware erkannt und auf einer Karte dargestellt

chen Überholabstands wird die oben beschriebene Abweichung von 10 % vollständig ausgeschöpft. Es war aber bereits während der Testdurchführung ersichtlich, dass es mit großen Schwierigkeiten verbunden ist ein Auto bzw. ein Fahrrad exakt geradeaus zu fahren.

### 6.3.2 Erkennen verschiedener Fahrten

**Testentwicklung** Die Auswertungssoftware wurde mit Daten getestet, welche eine, zwei oder vier Fahrten enthielten.

**Testergebnis** Die einzelnen Fahrten wurden erkannt und an den korrekten Punkten aufgetrennt. Innerhalb der einzelnen Fahrten wurde die Länge der zurückgelegten Strecke korrekt berechnet und die Überholvorgänge erkannt.

## 6.4 Ergebnis der Tests

### Radmesserhardware

Die Tests für die Radmesserhardware, besonders für die Ultraschallsensoren HC-SR04 haben gezeigt, dass diese zu einem ausreichend großen Anteil (95 %) korrekte Messergebnisse zurückliefern. Durch das Untersuchen der Schallgeschwindigkeit hinsichtlich Temperatur- und Luftfeuchtigkeitsabhängigkeit, wurde festgestellt, dass ein Temperatursensor notwendig ist. Auf einen Luftfeuchtigkeitssensor kann hingegen verzichtet werden. Weitere Messungen sind mit bewegten Luftmassen notwendig, da aufgrund begrenzter technischer Möglichkeiten keine Aussage mit ausreichender wissenschaftlicher Genauigkeit möglich

ist. Festzuhalten ist, dass der Radmesser nicht bei Sturm (-böen) eingesetzt werden soll. Die Ultraschallsensoren können, um bis zu  $19^\circ$  verdreht werden ohne eine Einschränkung der Objekterkennung zu verzeichnen. Dies gilt sowohl in Bezug auf den Erkennungswinkel  $\beta$  der Ultraschallsensoren, als auch bezüglich der Schrägen der Messfläche  $\gamma$ .

### **Radmessersoftware**

Anfängliche Probleme wies die Radmessersoftware bei der unabhängigen Funktionsweise der beiden Ultraschallsensoren auf. Dies konnte durch eine Syntax-Anpassung behoben werden, danach liefen alle Tests reibungslos ab. Es kam zu keinem Zeitpunkt zu einem Programmabsturz oder Programmabbruch. Ausgefallene Komponenten müssen erkannt und durch die in Abschnitt 4.1.4 beschriebene Anzeige des Betriebszustandes dem Nutzer mitgeteilt werden.

### **Auswertungssoftware**

Die Auswertungssoftware hat bei den zur Verfügung stehenden Daten die Überholvorgänge erkannt. Aus Daten, welche keine Überholvorgänge enthalten, wurden richtigerweise keine Überholvorgänge erkannt. Mit ersten Messdaten von der Straße, muss die Unterscheidung zwischen verschiedenen Fahrzeugarten erprobt werden.

# 7 Fazit

**Erreichen des Ziels** Das zu Beginn der Arbeit definierte Ziel wurde zu einem großen Teil erreicht. Es wurde ein Radmesser entwickelt, zusammengebaut und getestet. Da keine Bluetooth-Verbindung mehr zu einem Smartphone benötigt wird, kann das Trägerfahrrad getauscht werden und ein Radmesser auf diese Weise von mehreren Personen benutzt werden. Sofern Lötkenntnisse vorhanden sind und ein sicherer Umgang mit Windows 10 sowie Linux gegeben ist, kann der Radmesser vollständig selbstständig zusammengebaut werden. Nach der erfolgreichen Inbetriebnahme des Radmessers kann dieser komfortabel bedient werden und bis zum Auslesen der Daten besteht kein Handlungsbedarf. Das Auslesen der Daten kann über eine Remote-Desktop-Verbindung erfolgen. Die vollständige Software ist für jeden einsehbar, nachvollziehbar und kann von jedem weiter entwickelt werden. Wird kostenoptimiert eingekauft, kann ein Preis circa 50 € erzielt werden.

**Verbesserungsmöglichkeiten** Die im Rahmen dieser Arbeit durchgeführten Tests sind zwar sehr umfangreich, jedoch noch nicht vollständig ausreichend. Es muss der Einfluss von Wind bzw. Luftbewegung auf die Ultraschallmessung intensiver untersucht werden. Hierfür sollte entweder ein Windkanal verwendet werden oder aber eine feste Aufhängung für einen Ventilator. Die Auswertungssoftware muss mit mehr Daten getestet werden, um Feinjustierungen vornehmen zu können, und um die Software als bewährte Software zu qualifizieren. An dieser Stelle empfiehlt es sich Fahrten mit dem Radmesser auf normalen Straßen durchzuführen und diese mit einer Kamera aufzuzeichnen.

Damit auch Laien die Möglichkeit haben, ihre aufgezeichneten Daten auszuwerten, bzw. vom Raspberry Pi auszulesen, muss eine vereinfachte Möglichkeit des Datentransfers entwickelt werden. Da die Daten zentral gesammelt werden sollen, muss eine Plattform entwickelt werden, auf welcher alle mit dem Radmesser aufgezeichneten Daten hochgeladen werden können. Nach dem Hochladen der Daten, soll dann nach Möglichkeit eine automatisierte Auswertung starten. Dies ist insbesondere bei einer steigenden Zahl an Radmessern wichtig.

In Zusammenhang mit der automatisierten Auswertung der Daten soll auch eine automatisierte Aktualisierung der Datenvisualisierung einhergehen, welche dem Betrachter eine Übersicht über die Gesamtsituation ermöglicht sowie Detailinformationen auf Wunsch zur Verfügung stellt. In Anlehnung an den Radmesser des Tagesspiegel kann der in dieser Ar-

beit entwickelte Radmesser um einen Ultraschallsensor auf die rechte Seite hin ergänzt werden, mit welchem dann der Abstand zu bspw. parkenden Autos am rechten Fahrbahnrand ermittelt wird.

**Weiteres Vorgehen** Es ergeben sich die folgenden Schritte:

1. Durchführen der verbleibenden Tests
  - a) Test im Windkanal
  - b) Test der Auswertungssoftware mit mehr Daten
  - c) Abschließende Verifikation
2. Kleinserie
  - a) Beschaffung / Zusammenbau einer kleinen Zahl (10) an Radmessern
  - b) Durchführen von Messungen in einem begrenzten geografischen Gebiet
  - c) Regelmäßige Überprüfung der aufgezeichneten Daten
  - d) Abschließende Auswertung der im Rahmen der Kleinserie aufgenommenen Daten
3. Entwicklung Plattform Datensammlung
  - a) Entwicklung komfortable Auslesemöglichkeit Raspberry Pi
  - b) Entwicklung einer Plattform zur Sammlung der Daten
  - c) Test der Auslesemöglichkeit und der Datensammelplattform mit den Radmessern der Kleinserie
  - d) Massenauswertung der Daten etablieren
4. Datenerhebung im Großraum Stuttgart
  - a) Beschaffung / Zusammenbau von circa 100 Radmessern
  - b) Verteilen der Radmesser an freiwillige Radfahrer\*innen
  - c) Sammeln der Daten
  - d) Auswertung der gesammelten Daten und Erstellung einer Beispieldokumentation
  - e) Begutachtung der besonders gefährlichen Stellen
5. Verbreitung des Radmessers in europäische Städte

## 8 Literatur

- [1] J. Anke, L.-M. Schaefer und A. Francke. (3. Mai 2020). „Befragung: Wie verändert Corona unsere Mobilität langfristig?“, Verkehrspychologie an der TU Dresden, Adresse: <https://tu-dresden.de/bu/verkehr/ivs/vpsy/forschung/corona-mobilitaet> (besucht am 21.05.2020).
- [2] C. Eisenmann, V. Kolarova und C. Nobis. (5. Mai 2020). „DLR-Befragung: Wie verändert Corona unsere Mobilität?“, Adresse: <https://verkehrsforschung.dlr.de/de/news/dlr-befragung-wie-veraendert-corona-unsere-mobilitaet> (besucht am 21.05.2020).
- [3] A. Czeh. (18. Mai 2020). „Pop-up infrastructure for active mobility in Berlin, COVID-19 demands urgent changes to public spaces to enable safe mobility. The story map shows examples from Berlin and other cities.“, Institute for Advanced Sustainability Studies e. V. (IASS), Adresse: <https://storymaps.arcgis.com/stories/9f47ef654c7841e1a8d35034088d75b7>.
- [4] C. Wanner. (18. Mai 2020). „Nur so verhindern die globalen Metropolen den Infarkt“, WELT, Adresse: <https://www.welt.de/wirtschaft/article207912995/Verkehr-Nur-so-verhindern-die-globalen-Metropolen-den-Infarkt.html> (besucht am 21.05.2020).
- [5] J. Senders. (27. Apr. 2020). „Radverkehrsplanung und der Gesundheitsschutz – neue Radwege in Pandemiezeiten und darüber hinaus?“, Zukunft Mobilität, Adresse: <https://www.zukunft-mobilitaet.net/171346/analyse/corona-temporaere-radwege-stvo-langfristig-rechtliche-situation/> (besucht am 21.05.2020).
- [6] Bundesministerium für Umwelt, Naturschutz und nukleare Sicherheit (BMU), *Klimaschutzplan 2050, Klimaschutzpolitische Grundsätze und Ziele der Bundesregierung*. Berlin, 2016.
- [7] C. Hochfeld, A. Jung, A. Klein-Hithaß, U. Maier und K. Meyer, *Mit der Verkehrswende die Mobilität von morgen sichern, 12 Thesen zur Verkehrswende*. Berlin: Agora Verkehrswende, 2017.
- [8] Fachausschuss Radverkehr von ADFC und SRL, *Fachwissen für den Fahrradalltag, Seitliche Abstände*, 2010.

- [9] Tagesspiegel Online. (2018). „radmesser“, Verlag Der Tagesspiegel GmbH, Adresse: <https://interaktiv.tagesspiegel.de/radmesser/index.html> (besucht am 12.12.2019).
- [10] Juristische Zentrale ADAC, *Fahrradfahren – aber richtig!, Regeln, Informationen und Tipps*, 2018. Adresse: <https://www.adac.de/-/media/pdf/rechtsberatung/fahrradfahren.pdf?la=de-de&hash=3DA5CBA8E731EC3B22D24096B216BB9D> (besucht am 12.12.2019).
- [11] D. Saße. (2007). „Warum fahren Fahrräder so stabil?“, Deutsche Physikalische Gesellschaft e.V., Adresse: <https://www.weltderphysik.de/thema/hinter-den-dingen/stabilitaet-von-fahrraedern/> (besucht am 12.12.2019).
- [12] W. Suhr und H. J. Schlichting, „Gleichgewicht auf zwei Rädern, Physik des Fahrradfahrens“, *Physik unserer Zeit*, 5 Aug. 2007. DOI: 10.1002/piuz.200601149.
- [13] T. Richter, O. Beyer, J. Ortlepp und M. Schreiber, *Forschungsbericht Nr. 59, Sicherheit und Nutzbarkeit markierter Radverkehrsführungen*, 2019.
- [14] Gesamtverband der Deutschen Versicherungswirtschaft e.V., *Unfallforschung kompakt Nr. 89, Sicherheit und Nutzbarkeit markierter Radverkehrsführungen*, 2019.
- [15] M. Strehl. (5. Sep. 2015). „Radfahrer – Schutzstreifen“, Adresse: <https://www.verkehrskommentar.de/2015/09/radfahrer-schutzstreifen/> (besucht am 18.05.2020).
- [16] Amt für Mobilität und Tiefbau der Stadt Münster. (2020). „Verkehrssicherheit, Maßnahmen zur Verbesserung der Verkehrssicherheit“, Adresse: <https://www.stadt-muenster.de/verkehrsplanung/verkehrssicherheit/massnahmen.html> (besucht am 18.05.2020).
- [17] A. K. Huemer, *Wie beeinflusst die Infrastruktur für Radfahrer das Überholverhalten von Autofahrern?, Ingenieur- und Verkehrspychologie*, 2019.
- [18] H. Wittlich. (2019). „Data Journalism Award, Radmesser“, Adresse: <https://datajournalismawards.org/projects/radmesser/> (besucht am 11.02.2020).
- [19] Tagesspiegel. (2019). „radmesser, opensensor“, Adresse: <https://github.com/tagesspiegel/radmesser/tree/master/opensensor> (besucht am 22.01.2020).
- [20] flipdot. (2018). „Sicherheit für Fahrradfahrer in Kassel“, Adresse: <https://flipdot.org/blog/archives/421-Sicherheit-fuer-Fahrradfahrer-in-Kassel.html> (besucht am 20.01.2020).
- [21] Zweirat Stuttgart. (2019). „Radmesser“, Adresse: <https://github.com/tobst/radmesserS> (besucht am 28.01.2020).

- [22] Zweirat Stuttgart. (2020). „Friends of Open Bike Sensor, Citizen Science Project by Zweirat Stuttgart“, Adresse: <https://github.com/Friends-of-OpenBikeSensor> (besucht am 28.05.2020).
- [23] Codaxus LLC. (2017). „C3FT v3“, Adresse: <http://codaxus.com/c3ft/c3ft-v3/> (besucht am 22.01.2020).
- [24] Raspberry Pi Foundation. (o.J.). „Raspberry Pi hardware“, Adresse: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md> (besucht am 29.01.2020).
- [25] Raspberry Pi Foundation. (2020). „Raspberry Pi Zero W“, Adresse: <https://www.raspberrypi.org/products/raspberry-pi-zero-w/> (besucht am 28.05.2020).
- [26] Team Beebom. (2017). „Top 5 Raspberry Pi Zero Alternatives You Can Buy“, Beebom, Adresse: <https://beebom.com/raspberry-pi-zero-alternatives/> (besucht am 31.01.2020).
- [27] N. A. Singh und M. Borschbach, „Effect of external factors on accuracy of distance measurement using ultrasonic sensors“, in *2017 International Conference on Signals and Systems (ICSigSys)*, 2017, S. 266–271.
- [28] K. A. Raza und W. Monnet, „Moving objects detection and direction-finding with HC-SR04 ultrasonic linear array“, in *2019 International Engineering Conference (IEC)*, 2019, S. 153–158.
- [29] D. Vakula und Y. K. Kolli, „Low cost smart parking system for smart cities“, in *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, 2017, S. 280–284.
- [30] S Adarsh, S. M. Kaleemuddin, D. Bose und K. I. Ramachandran, „Performance comparison of Infrared and Ultrasonic sensors for obstacles of different materials in vehicle/ robot navigation applications“, *IOP Conference Series: Materials Science and Engineering*, Jg. 149, Sep. 2016. DOI: [10.1088/1757-899x/149/1/012141](https://doi.org/10.1088/1757-899x/149/1/012141).
- [31] ITead Studio, *Ultrasonic ranging module : HC-SR04*, 2010.
- [32] Maxim Integrated Products, Inc., *DS18S20, High-Precision 1-Wire Digital Thermometer*, Version 3, 2015.
- [33] W. Jingzhuo und G. Chenglong, „Research on 1-Wire Bus Temperature Monitoring System“, in *2007 8th International Conference on Electronic Measurement and Instruments*, 2007, S. 3–722–3–726. DOI: [10.1109/ICEMI.2007.4351019](https://doi.org/10.1109/ICEMI.2007.4351019).
- [34] B. Huang, J. Lei und Y. Bo, „The reading data error analysis of 1-wire bus digital temperature sensor DS18B20“, in *2012 Proceedings of International Conference on Modelling, Identification and Control*, Wuhan, Hubei, China, Juni 2012, S. 433–436, ISBN: 978-1-4673-1524-1.

- [35] Li Pengfei, Li Jiakun und Jing Junfeng, „Wireless temperature monitoring system based on the ZigBee technology“, in *2010 2nd International Conference on Computer Engineering and Technology*, Bd. 1, 2010, S. V1–160–V1–163. DOI: 10.1109/ICCET.2010.5486248.
- [36] H. Fu, G. Xiong, S. Chen, M. Qiu, H. Xiong, Z. Shen, X. Dong, X. Su und X. Guo, „Design of Low-cost Position Differential Positioning System Based on STM32“, in *2018 Chinese Automation Congress (CAC)*, 2018, S. 1420–1425. DOI: 10.1109/CAC.2018.8623492.
- [37] u-blox AG, *NEO-6, u-blox 6 GPS Modules*, Data Sheet, Thalwil, 2011.
- [38] F. Bauer, F. Jörg und B. Wagner, „Konstruktion eines Geäuses zur Fahrradabstandsmessung“, Duale Hochschule Baden-Württemberg, Stuttgart, Studienarbeit T3200, 2019.
- [39] D. Mauermaier, „Verkehrswesen“, in *Wendehorst Bautechnische Zahlentafeln*, 36. Aufl. Wiesbaden: Springer Vieweg, 2018, Kap. 18, ISBN: 978-3-658-17935-9. DOI: 10.1007/978-3-658-17936-6.
- [40] H. Butler, I. Hobu, M. Daly, Cadcorp, A. Doyle, S. Gillies, Mapbox, S. Hagen, T. Schaub und Planet Labs, *The GeoJSON Format*, Proposed Standard, Internet Engineering Task Force, 2016. Adresse: <https://tools.ietf.org/html/rfc7946> (besucht am 19.04.2020).
- [41] B. Balter. (13. Juni 2013). „There's a map for that“, The Github Blog, Adresse: <https://github.blog/2013-06-13-there-s-a-map-for-that/> (besucht am 19.04.2020).
- [42] D. Foster, *GPX 1.1 Schema Documentation*, 2020. Adresse: <https://www.topografix.com/GPX/1/1/> (besucht am 19.04.2020).
- [43] Open Geospatial Consortium, *Geography Markup Language (GML), Extended schemas and encoding rules*, hrsg. von E. Protele, OpenGIS® Implementation Standard, Version 3.3.0, 2012. Adresse: <http://www.opengis.net/spec/GML/3.3> (besucht am 19.04.2020).
- [44] Open Geospatial Consortium, *KML 2.3*, hrsg. von D. Burggraf, OpenGIS® Implementation Standard, Version 1.0, 2015. Adresse: <http://www.opengis.net/doc/IS/kml/2.3> (besucht am 19.04.2020).
- [45] GoogleDevelopers. (2020). „Geocoding API Usage and Billing“, Adresse: <https://developers.google.com/maps/documentation/geocoding/usage-and-billing> (besucht am 12.05.2020).
- [46] OpenCage GmbH. (2020). „OpenCage Geocoder“, Adresse: <https://opencagedata.com/> (besucht am 12.05.2020).

- [47] The R Foundation. (2020). „The R Project for Statistical Computing“, Adresse: <https://www.r-project.org/> (besucht am 13.05.2020).
- [48] R. Sinambari und S. Sentpali, *Ingenieurakustik, Physikalische Grundlagen, Anwendungsbeispiele und Übungen*, 6. Aufl. Wiesbaden: Springer Vieweg, 2020, ISBN: 978-3-658-27288-3. DOI: 10.1007/978-3-658-27289-0.
- [49] Spektrum. (1998). „Adiabatenexponent“, Spektrum Akademischer Verlag, Adresse: <https://www.spektrum.de/lexikon/physik/adiabatenexponent/204> (besucht am 23.04.2020).
- [50] B. Lippold. (2020). „Liste der Dichte gasförmiger Stoffe“, Lumitos AG, Adresse: [https://www.chemie.de/lexikon/Liste\\_der\\_Dichte\\_gasf\\"ormiger\\_Stoffe.html](https://www.chemie.de/lexikon/Liste_der_Dichte_gasf\\) (besucht am 26.04.2020).
- [51] Kraftfahrt-Bundesamt, *Bundeseinheitlicher Tatbestandskatalog, Straßenordnungswidrigkeiten*, Version 12, 1. Nov. 2017.
- [52] Bundesanstalt für Straßenwesen. (2020). „RAL - Die neuen Stoffentypen für Landstraßen“, Adresse: [https://www.bast.de/BAST\\_2017/DE/Verkehrstechnik/Fachthemen/v1-strassentypen.html?nn=1817946](https://www.bast.de/BAST_2017/DE/Verkehrstechnik/Fachthemen/v1-strassentypen.html?nn=1817946) (besucht am 12.04.2020).

# 9 Anhang

## 9.1 Abmessungen von KFZ

| Fahrzeugart      | minimale Länge | maximale Länge | durchschnittliche Länge | Breite  |
|------------------|----------------|----------------|-------------------------|---------|
| Leichtkrafträder | 2200 mm        | 3170 mm        | 2600 mm                 | 900 mm  |
| PKW              | 2695 mm        | 6165 mm        | 4900 mm                 | 1900 mm |
| LKW              | 5000 mm        | 26500 mm       | 13000 mm                | 2500 mm |
| Bus              | 8000 mm        | 18750 mm       | 12000 mm                | 2550 mm |
| Straßenbahn      | 11370 mm       | 75000 mm       | 22000 mm                | 2800 mm |

Tabelle 9.1: Länge von verschiedenen KFZ

Die durchschnittlichen Distanzen von Personenkraftwagen (PKW), LKW und Bus entstammen dem Buch „Verkehrswesen“ *Wendehorst Bautechnische Zahlentafeln* [39].

## 9.2 Definition eines Überholvorgangs

Um Überholabstände erkennen zu können, müssen Geschwindigkeiten, Zeiten und Abstände definiert werden, nach welchen entschieden wird, ob es sich bei einem aktuellen Messergebnis um einen Überholvorgang handelt. Bei der Definition eines Überholvorgangs wird zwischen Stadtverkehr und Landstraßen (Straßen außerhalb geschlossener Ortschaften) unterschieden. Auch wenn das Einsatzgebiet des Radmessers primär in der Stadt zu sehen ist, sind auch Überlandfahrten zu erwarten. Bei der Definition müssen verschiedene KFZ-Arten berücksichtigt werden.

### 9.2.1 Stadtverkehr

#### Erlaubte Geschwindigkeit

Im Stadtverkehr ist eine Maximalgeschwindigkeit von  $50 \frac{\text{km}}{\text{h}}$  erlaubt (StVO §3 Abs.3 Nr.1). An vielen Stellen wird die erlaubte Maximalgeschwindigkeit auf bis zu  $30 \frac{\text{km}}{\text{h}}$  reduziert. Mit dem Einhalten der Maximalgeschwindigkeiten ist durch die KFZ-Führer nicht garantiert. Geringe Überschreitungen sind als Alltag zu bewerten. Geschwindigkeitsüber-

schreitungen bis zu  $20 \frac{\text{km}}{\text{h}}$  stellen bei PKW keine schwerwiegende Zuwiederhandlung dar und werden deshalb nur mit einem Bußgeld von bis zu 35 N geahndet [51]. Es ist davon auszugehen, dass dies von manchen KFZ-Führern billigend in Kauf genommen wird. Eine weiterführende Erklärung ist in Abschnitt 9.3.1 zu finden. Dort werden Unterschiede bei den einzelnen Fahrzeugarten genauer erläutert.

Verkehrsbedingt ist eine deutliche Unterschreitung der Maximalgeschwindigkeit möglich (bsp. Stop and Go).

Durch bestimmte Verkehrssituationen kann es vorkommen, dass Fahrradfahrer trotz fließenden KFZ-Verkehrs nicht weiter fahren können, weshalb die Differenz der Geschwindigkeiten zwischen Fahrradfahrer und KFZ den maximalen und minimalen Geschwindigkeit von KFZ entspricht. Im Stadtverkehr muss als Minimalgeschwindigkeit  $v_{minStadt} = 10 \frac{\text{km}}{\text{h}} = 2,78 \text{ m s}^{-1}$  berücksichtigt werden.

## Überholzeit PKW

Wie in Abschnitt 9.1 aufgelistet, ist eine Überschneidung bei der Länge von Leichtkrafträder und PKW zu verzeichnen. Der kürzeste PKW ist kürzer als das längste Leichtkraftrad. PKW und Leichtkrafträder werden bei der Auswertung als eine Kategorie angesehen. Es ist jedoch der Fall, dass PKW eine im Gegensatz zu Leichtkrafträder, sehr gleichmäßige Oberfläche haben, was deren Erkennung aus den Messergebnissen vereinfacht.

Zur Berechnung der minimale Überholzeit  $t_{PKW\_min}$  wird die minimale Länge von Leichtkrafträder  $l_{min\_Leichtkraft} = 2,2 \text{ m}$  berücksichtigt. Als maximale Geschwindigkeit wird  $v_{max\_Stadt\_PKW} = 70 \frac{\text{km}}{\text{h}} = 19,44 \text{ m s}^{-1}$  herangezogen:

$$t_{PKW\_min} = \frac{l_{min\_Leichtkraft}}{v_{max\_Stadt\_PKW}} = \frac{2,2 \text{ m}}{19,44 \text{ m s}^{-1}} = 113,14 \text{ ms} \approx 0,11 \text{ s} \quad (9.1)$$

Als maximale Überholzeit  $t_{PKW\_max}$  wird mit der maximalen Länge von PKW  $l_{max\_PKW} = 6,165 \text{ m}$  gerechnet:

$$t_{PKW\_max} = \frac{l_{max\_PKW}}{v_{min\_Stadt}} = \frac{6,165 \text{ m}}{2,78 \text{ m s}^{-1}} = 2219,4 \text{ ms} \approx 2,22 \text{ s} \quad (9.2)$$

## Überholzeit LKW

Als minimale Überholzeit  $t_{LKW\_min}$  wird nun unter Berücksichtigung der minimalen Länge eines LKW  $l_{min\_LKW} = 5 \text{ m}$  und der maximalen Geschwindigkeit  $v_{max\_Stadt\_LKW} = 65 \frac{\text{km}}{\text{h}} = 18,05 \text{ m s}^{-1}$  Folgendes berechnet:

$$t_{LKW\_min} = \frac{l_{min\_LKW}}{v_{max\_Stadt\_LKW}} = \frac{5 \text{ m}}{18,05 \text{ m s}^{-1}} = 276,9 \text{ ms} \approx 0,28 \text{ s} \quad (9.3)$$

Als maximale Überholzeit  $t_{LKW\_max}$  wird auf Grundlage der maximalen Länge von LKW  $l_{max\_LKW} = 26,5$  m folgendes berechnet:

$$t_{LKW\_max} = \frac{l_{max\_LKW}}{v_{min\_Stadt}} = \frac{26,5 \text{ m}}{2,78 \text{ m s}^{-1}} = 9540 \text{ ms} \approx 9,54 \text{ s} \quad (9.4)$$

## Überholzeit Kraftomnibusse

Zur Berechnung der minimalen Überholzeit  $t_{Bus\_min\_Stadt}$  von Bussen wird die maximale Geschwindigkeit von  $v_{max\_Stadt\_Bus} = 60 \frac{\text{km}}{\text{h}} = 16,67 \text{ m s}^{-1}$ , die minimale Länge von  $l_{min\_Bus} = 8$  m:

$$t_{Bus\_min\_Stadt} = \frac{l_{min\_Bus}}{v_{max\_Stadt\_Bus}} = \frac{8 \text{ m}}{16,67 \text{ m s}^{-1}} = 480 \text{ ms} \approx 0,48 \text{ s} \quad (9.5)$$

Für die maximale Überholzeit  $t_{Bus\_max\_Stadt}$  wird die maximale Länge  $l_{max\_Bus} = 18,75$  m von Bussen berücksichtigt:

$$t_{Bus\_max\_Stadt} = \frac{l_{max\_Bus}}{v_{min}} = \frac{18,75 \text{ m}}{2,78 \text{ m s}^{-1}} = 6750 \text{ ms} \approx 6,75 \text{ s} \quad (9.6)$$

## Überholzeit Straßenbahn

Bei Straßenbahnen ist von einer genaueren Einhaltung der Maximalgeschwindigkeit auszugehen, für die Berechnung wird deshalb als maximale Geschwindigkeit  $v_{max\_Strass} = 60 \frac{\text{km}}{\text{h}} = 16,67 \text{ m s}^{-1}$  angenommen.

Als minimale Überholzeit  $t_{Strass\_min}$  wird ergibt sich unter Berücksichtigung der minimalen Länge einer Straßenbahn  $l_{min\_Strass} = 11,37$  m:

$$t_{Strass\_min} = \frac{s_{min\_Strass}}{v_{max\_Strass}} = \frac{11,37 \text{ m}}{16,67 \text{ m s}^{-1}} = 682,2 \text{ ms} \approx 0,68 \text{ s} \quad (9.7)$$

Die maximale Überholzeit  $t_{Strass\_max}$  wird mit der maximalen Länge einer Straßenbahn  $l_{max\_Strass} = 75$  m berechnet:

$$t_{Strassmax} = \frac{s_{maxStrass}}{v_{minStadt}} = \frac{75 \text{ m}}{2,78 \text{ m s}^{-1}} = 27\,000 \text{ ms} \approx 27 \text{ s} \quad (9.8)$$

### 9.2.2 Außerhalb geschlossener Ortschaften

Außerhalb geschlossener Ortschaften ist mit Straßenbahnen im normalen Verkehr nicht zu rechnen.

## Erlaubte Geschwindigkeit

Auf Landstraßen, welche von KFZ und Fahrradfahrern gemeinsam genutzt werden, beträgt die maximal erlaubte Geschwindigkeit  $100 \frac{\text{km}}{\text{h}}$  (StVO §3 Abs.3 Nr.2 Buchst. c). Bei Überschreitung der maximal erlaubten Geschwindigkeit, um bis zu  $20 \frac{\text{km}}{\text{h}}$  ist, wie im Stadtverkehr lediglich mit einer Geldstrafe von bis zu 30 N zu rechnen [51]. Als Maximalgeschwindigkeit ist also  $v_{max\_Land} = 120 \frac{\text{km}}{\text{h}} = 33,33 \text{ m s}^{-1}$  zu berücksichtigen. Für die minimale Geschwindigkeit ist die selbe Grenze, wie im Stadtverkehr 9.2.1 anzunehmen. Eine ausführliche Erklärung der angenommenen Geschwindigkeiten findet sich im Anhang, Abschnitt 9.3.1. Um alle Situationen abzudecken, muss auch außerhalb geschlossener Ortschaften davon ausgegangen werden, dass der Fahrradfahrer verkehrsbedingt nicht weiter fahren kann, während KFZ mit ihren definierten maximalen und minimalen Geschwindigkeiten überholen. Die Differenz der Geschwindigkeiten entspricht also der maximalen und minimalen Geschwindigkeit der jeweiligen KFZ.

## Überholzeit PKW

Die maximale Überholzeit aus dem Stadtverkehr kann übernommen werden, es muss lediglich eine neue minimale Überholzeit  $t_{PKW\_min\_Land}$  berechnet werden:

$$t_{PKW\_min\_Land} = \frac{l_{min\_Leichtkraft}}{v_{maxLand}} = \frac{2,2 \text{ m}}{33,33 \text{ m s}^{-1}} = 66 \text{ ms} \approx 0,07 \text{ s} \quad (9.9)$$

## Überholzeit Schwerfahrzeuge

Die maximale Überholzeit kann wieder aus dem Stadtverkehr übernommen werden. Für die minimale Überholzeit müssen jedoch für LKW  $t_{LKW\_min\_Land}$  und Busse  $t_{Bus\_min\_Land}$  nun getrennte Berechnungen aufgestellt werden. Mit der minimalen Länge von Bussen  $l_{min\_Bus} = 8 \text{ m}$  und LKW  $l_{min\_LKW} = 5 \text{ m}$  sowie der maximalen Geschwindigkeit  $v_{max\_Land\_Bus} = v_{max\_Land\_LKWk} = 95 \frac{\text{km}}{\text{h}} = 26,38 \text{ m s}^{-1}$  ergibt sich:

$$t_{Bus\_min\_Land} = \frac{l_{min\_Bus}}{v_{max\_Land\_Bus}} = \frac{8 \text{ m}}{26,38 \text{ m s}^{-1}} = 303,16 \text{ ms} \approx 0,3 \text{ s} \quad (9.10)$$

$$t_{LKW\_min\_Land} = \frac{l_{minLKW}}{v_{max\_Land\_LKWk}} = \frac{5 \text{ m}}{26,38 \text{ m s}^{-1}} = 189,47 \text{ ms} \approx 0,19 \text{ s} \quad (9.11)$$

Es wird lediglich die maximale Geschwindigkeit für kleine LKW  $v_{max\_Land\_LKW\_k}$  verwendet, da gilt  $v_{max\_Land\_LKW\_k} > v_{max\_Land\_LKW\_g}$ .

| Fahrzeugart | maximale Zeit | minimale Zeit Stadt | minimale Zeit Land |
|-------------|---------------|---------------------|--------------------|
| PKW         | 2,22 s        | 0,11 s              | 0,07 s             |
| LKW         | 9,54 s        | 0,28 s              | 0,19 s             |
| Bus         | 6,75 s        | 0,48 s              | 0,3 s              |
| Straßenbahn | 27 s          | 0,68 s              | entfällt           |

Tabelle 9.2: Zusammenfassung der minimalen und maximalen Überholzeiten verschiedener Arten von KFZ

### 9.2.3 Zusammenfassung der Überholzeiten

Wie aus der Zusammenfassung der Überholzeiten in Tabelle 9.2 deutlich wird, besteht bei den Überholzeiten zwischen allen Fahrzeugkategorien Überschneidungen. Es ist nicht ausreichend die Überholvorgänge nur anhand der Überholzeiten zu unterscheiden. Eine exaktere Berechnung, wie in Abschnitt 5.2.2 gezeigt ist erforderlich. Die Überholzeiten werden aber weiterhin als Zeitbegrenzungen in der Software berücksichtigt.

## 9.3 Erläuterung der in Kaufnahme von Bußgeldern

### 9.3.1 Geschwindigkeitsüberschreitungen

Zur Berechnung der Überholzeiten werden von den gegebenen Geschwindigkeitsbegrenzungen teilweise Überschreitungen angenommen, welche sich auf die durch den Bundeseinheitlichen Tatbestandskatalog vorgenommene Kategorisierung und Sanktionierung beziehen. Hierzu wird der BTK in der 12. Auflage vom 1.11.2017 herangezogen [51]. Bezugnehmend auf die Tatbestands Nummer (TBNR) 103202 bis 103204 (innerorts), welche auf STVO §3 Abs.3 Nr.1 aufbauen und die TBNR 103208 bis 1032010 (außerorts), welche auf StVO §3 Abs.3 Nr.2 aufbauen, sind Geschwindigkeitsüberschreitungen bis zu  $20 \frac{\text{km}}{\text{h}}$  von KFZ, welche keine LKW o.ä. sind und keine gefährlichen Güter transportieren (also alle normalen PKW) weder der Wertungskategorie weniger schwerwiegend noch der Wertungskategorie schwerwiegende Zuwiderhandlung zugeordnet und deshalb nur mit einem Bußgeld in Höhe von maximal 35 € belegt.

Zusammengefasst werden die bis jetzt aufgeführten TBNR in der Tabelle 9.3. Aus Tabelle 9.3 wird ersichtlich, dass ab einer Geschwindigkeitsüberschreitung von  $21 \frac{\text{km}}{\text{h}}$  dieser Verstoß als schwerwiegende Zuwiderhandlung kategorisiert wird. Die jeweils obere Zeile innerhalb eines Geschwindigkeitsblocks behandelt die durch die StVO §3 Abs.3 definierten maximalen Geschwindigkeiten, die untere Zeile behandelt die Geschwindigkeitsüberschreitung bei Vorschriftenzeichen gemäß StVO §41 Abs. 1 iVm Anlage 2. Zur Berechnung der Überholzeiten in Abschnitt 9.2 wird für PKW deshalb innerorts eine Maximalgeschwindigkeit von  $v_{max\_Stadt\_PKW} = 70 \frac{\text{km}}{\text{h}}$  angenommen, außerorts gilt  $v_{max\_Land} = 120 \frac{\text{km}}{\text{h}}$  als

**Überschreiten**

- der zulässigen Höchstgeschwindigkeit,

| Tabellen-Nr.:<br>(innerhalb)<br>703010<br>741007 |         | (außerhalb)<br>703011<br>741008 |         | mit anderen als den vorstehend aufgeführten Kraftfahrzeugen |        |           |         |     |        |  |
|--|---------|---------------------------------|---------|---|--------|-----------|---------|-----|--------|--|
|  |         | innerhalb                       |         |   |        | außerhalb |         |     |        |  |
| Tatbestand                                       | BKat    | TBNR                            | FaP-Pkt | FV  | Euro   | TBNR      | FaP-Pkt | FV  | Euro   |  |
| <b>Überschreitung in km/h</b>                    |         |                                 |         |   |        |           |         |     |        |  |
| bis 10   | 11.3.1  | 103202                          | 0       |   | 15,00  | 103208    | 0       |     | 10,00  |  |
|  | 11.3.1  | 141236                          | 0       |   | 15,00  | 141239    | 0       |     | 10,00  |  |
| von 11-15  | 11.3.2  | 103203                          | 0       |   | 25,00  | 103209    | 0       |     | 20,00  |  |
|  | 11.3.2  | 141237                          | 0       |   | 25,00  | 141240    | 0       |     | 20,00  |  |
| von 16-20  | 11.3.3  | 103204                          | 0       |   | 35,00  | 103210    | 0       |     | 30,00  |  |
|  | 11.3.3  | 141238                          | 0       |   | 35,00  | 141241    | 0       |     | 30,00  |  |
| von 21-25  | 11.3.4  | 103762                          | A - 1   |   | 80,00  | 103774    | A - 1   |     | 70,00  |  |
|  | 11.3.4  | 141712                          | A - 1   |   | 80,00  | 141721    | A - 1   |     | 70,00  |  |
| von 26-30  | 11.3.5  | 103763                          | A - 1   |   | 100,00 | 103775    | A - 1   |     | 80,00  |  |
|  | 11.3.5  | 141713                          | A - 1   |   | 100,00 | 141722    | A - 1   |     | 80,00  |  |
| von 31-40  | 11.3.6  | 103764                          | A - 2   | 1 M   | 160,00 | 103776    | A - 1   |     | 120,00 |  |
|  | 11.3.6  | 141714                          | A - 2   | 1 M   | 160,00 | 141723    | A - 1   |     | 120,00 |  |
| von 41-50  | 11.3.7  | 103765                          | A - 2   | 1 M   | 200,00 | 103777    | A - 2   | 1 M | 160,00 |  |
|  | 11.3.7  | 141715                          | A - 2   | 1 M   | 200,00 | 141724    | A - 2   | 1 M | 160,00 |  |
| von 51-60  | 11.3.8  | 103766                          | A - 2   | 2 M   | 280,00 | 103778    | A - 2   | 1 M | 240,00 |  |
|  | 11.3.8  | 141716                          | A - 2   | 2 M   | 280,00 | 141725    | A - 2   | 1 M | 240,00 |  |
| von 61-70  | 11.3.9  | 103767                          | A - 2   | 3 M   | 480,00 | 103779    | A - 2   | 2 M | 440,00 |  |
|  | 11.3.9  | 141717                          | A - 2   | 3 M   | 480,00 | 141726    | A - 2   | 2 M | 440,00 |  |
| über 70  | 11.3.10 | 103768                          | A - 2   | 3 M   | 680,00 | 103780    | A - 2   | 3 M | 600,00 |  |
|  | 11.3.10 | 141718                          | A - 2   | 3 M   | 680,00 | 141727    | A - 2   | 3 M | 600,00 |  |

Tabelle 9.3: Tabelle 103010 und 103011 aus Bundeseinheitlicher Tatbestandskatalog (BTK) [51] für normale PKW

Berechnungsgrundlage.

Für Kraftfahrzeuge wie LKW gilt eine Geschwindigkeitsüberschreitung schon ab  $16 \frac{\text{km}}{\text{h}}$  als schwerwiegende Zuwiderhandlung. Dies gilt sowohl innerorts, als auch außerorts und ergibt sich aus den TBNR 103178, 103179 und 103184, 103185, zusammengefasst in den Tabellen 703006 und 703007 [51].

Gemäß StVO §3 Abs.3 Nr. 2 Buchst. a gilt für LKW mit einer zulässigen Gesamtmasse von bis zu 7,5 t außerorts eine Geschwindigkeitsbegrenzung von  $80 \frac{\text{km}}{\text{h}}$ . Eine zulässige Gesamtmasse von unter 7,5 t ist bei kleinen LKW der Fall. Bei einer zulässigen Gesamtmasse von über 7,5 t greift StVO §3 Abs.3 Nr.2 Buchst. b, welche außerorts eine Maximalgeschwindigkeit von  $60 \frac{\text{km}}{\text{h}}$  festlegt. (StVO: Stand März 2019).

Als Berechnungsgrundlage wird innerorts eine einheitliche Maximalgeschwindigkeit von  $v_{max\_Stadt\_LKW} = 65 \frac{\text{km}}{\text{h}}$  festgesetzt. Außerhalb geschlossener Ortschaften muss jedoch differenziert werden zwischen LKW  $< 7,5$  t, für welche gilt  $v_{max\_Land\_LKW\_k} = 95 \frac{\text{km}}{\text{h}}$  und LKW  $> 7,5$  t, für welche die Berechnungsgrundlage  $v_{max\_Land\_LKW\_g} = 75 \frac{\text{km}}{\text{h}}$ . Hierbei werden LKW  $< 7,5$  t mit der minimalen Länge gleichgesetzt und LKW  $> 7,5$  t mit der

maximalen Länge.

Für Kraftomnibusse mit Fahrgästen ergibt sich, wenn alle Fahrgäste einen Sitzplatz haben aus StVO §3 Abs.3 Nr.2 Buchst. a eine zulässige Höchstgeschwindigkeit von  $80 \frac{\text{km}}{\text{h}}$ . Stehen jedoch nicht allen Fahrgästen Sitzplätze zur Verfügung greift StVO §3 Abs.3 Nr.2 Buchst. b und definiert die eine Maximalgeschwindigkeit von  $60 \frac{\text{km}}{\text{h}}$ . Es ist für die Berechnung von der höheren Geschwindigkeitsbegrenzung auszugehen. Bei Kraftomnibussen werden Tatbestände anders kategorisiert, wie bei LKW oder PKW. Innerorts wird durch TBNR 103190 schon ein Überschreiten der zulässigen Höchstgeschwindigkeit um  $11 \frac{\text{km}}{\text{h}}$  als schwerwiegende Zuwiderhandlung eingestuft, außerhalb von Ortschaften jedoch nach TBNR 103196 und 103197 erst ab einer Überschreitung von  $16 \frac{\text{km}}{\text{h}}$  [51]. Zur Berechnung der Überholzeiten in Abschnitt 9.2 sollen folgende Geschwindigkeiten berücksichtigt werden: maximale Geschwindigkeit innerorts  $v_{max\_Stad\_tBus} = 60 \frac{\text{km}}{\text{h}}$  und maximale Geschwindigkeit außerorts  $v_{max\_Land\_Bus} = 95 \frac{\text{km}}{\text{h}}$ .

## **9.4 Erläuterung der Straßenbreite auf Grundlage der Richtlinien für die Anlage von Landstraßen (RAL) und der Richtlinien für die Anlage von Stadtstraßen (RASt)**

Die Breite von Verkehrsstraßen, welche für unsere Untersuchung von Bedeutung sind, weil sie sowohl von KFZ, als auch von Fahrradfahrern benutzt werden, sind in der RASt und der RAL geregelt.

Die Breite und Ausprägung von Straßenanlagen ist von deren Nutzungsdichte und deren Verbindungsbedeutung abhängig. Die RAL definiert auf dieser Grundlage vier Entwurfsklassen (bei Landstraßen) (EKL) für Landstraßen. Die Fernstraße und (EKL 1) und die Überregionalstraße (EKL 2) sind für KFZ vorgesehen und der Radverkehr soll 'auf fahrbahnbegleitenden Sonderwegen geführt werden' [52]. Lediglich auf Regionalstraßen (EKL 3) und Nahbereichsstraßen (EKL 4) teilen sich KFZ und Fahrradfahrer die Fahrbahn. Auf Straßen der EKL 3 soll ein Fahrstreifen eine Breite von 3,5 m aufweisen (mit zwei Fahrstreifen - jeder für eine Richtung - ergibt sich eine Fahrbahnbreite von 7 m) und auf Straßen der EKL 4 soll die Fahrbahn eine Breite von 5 m aufweisen. [52]

Während die RAL lediglich vier Kategorien von Straßen berücksichtigt, wird es innerhalb von Städten mit der RASt, welche zwölf Straßenkategorien berücksichtigt, relativ schnell unübersichtlich. Auch innerhalb von Städten wird die Straßenkategorie aufgrund des Verkehrsaufkommens und der Verbindungsstufe gewählt. Weiterhin ist entscheidend, ob auf der Straße Linienbusverkehr, Straßenbahnverkehr oder aber kein ÖPNV geführt wird.

Die folgenden zwölf Straßenkategorien sind innerhalb deutscher Städte zu finden.

- **Wohnweg:**  
Aufgrund der geringen Länge (<100 m) und der geringen Breite der Fahrgasse (<5 m) sind kaum Überholvorgänge zu erwarten
- **Wohnstraße:**  
Mit bis zu 300 m Länge und bis zu 400 KFZ/h sowie einer größeren Breite bis zu 6,5 m sind Überholvorgänge wahrscheinlicher, jedoch nicht in großen Zahlen zu erwarten
- **Sammelstraße:**  
Verkehrsdichte und Länge steigen an, je nach Ausführung der Straße ist ein Schutzstreifen für Fahrradfahrer möglich, Überholvorgänge wahrscheinlich
- **Quartierstraße:**  
Geringere Länge als Sammelstraße, jedoch höhere Nutzungsdichte, Anlage von Schutzstreifen für Fahrradfahrer oder Gehweg für Fahrrad frei möglich, Überholvorgänge wahrscheinlich
- **Dörfliche Hauptstraße:**  
Durch eine mögliche Länge von bis zu mehreren Kilometern sind viele Überholvorgänge zu erwarten
- **Örtliche Einfahrtsstraße:**  
Zu erwartende Nutzung größer als bei der dörflichen Hauptstraße, Fahrradinfrastruktur auf der Straße geführt, sehr viele Überholvorgänge zu erwarten
- **Örtliche Geschäftsstraße:**  
Hohe Verkehrslast, zudem Straßenbahnverkehr möglich, Radweg meist getrennt von Hauptfahrbahn geführt, viele Überholvorgänge erwartet, wenn Fahrradstreifen auf der Hauptfahrbahn, ansonsten keine Überholvorgänge zu messen (Trennung KFZ und Fahrrad)
- **Haupugeschäftsstraße:**  
Zunehmende Verkehrslast, Straßenbahnen möglich, Radweg teilweise getrennt von Hauptfahrbahn geführt, Bussonderstreifen können für Radverkehr freigegeben werden, sehr viele Überholvorgänge zu erwarten (wenn Fahrrad auf Hauptfahrbahn)
- **Gewerbestraße:**  
Etwas geringere Verkehrslast, als bei den zuvor genannten, Radfahrstreifen teilweise auf der Fahrbahn ausgeführt, viele Überholvorgänge zu erwarten (bei Radfahrstreifen, nicht bei baulicher Trennung)
- **Industriestraße:**  
Radweg meist getrennt von Hauptfahrbahn geführt, Überholvorgänge sind nur bei gemeinsamer Verkehrsführung möglich, insgesamt wenig Überholvorgänge zu erwarten

- Verbindungsstraße:

Radweg meist getrennt von Hauptfahrbahn, zusätzlicher Busstreifen zwischen Hauptfahrbahn und Radweg, Überholvorgänge sind zu erwarten

- Anbaufreie Straße:

Radweg immer getrennt von Hauptfahrbahn, Straßenbahnverkehr möglich, jedoch nicht als überholendes Fahrzeug, keine Überholvorgänge aufgrund baulicher Trennung zu erwarten

Keine Straße weist eine Fahrbahnbreite von mehr als 7 m auf, welche für den KFZ-Verkehr vorgesehen ist. Einzige Ausnahme bilden hier Industriestraßen mit Linienbusverkehr und mittelstarker Nutzungsdichte (800-1800 KFZ/h) ohne Mittelstreifen und Verbindungsstraßen mit Linienbusverkehr, großer Nutzungsdichte (>1600 KFZ/h) ohne Mittelstreifen.

Die in diesem Abschnitt verwendeten Quellen sind nicht die Primärquellen. Da es sich jedoch nicht um eine zentrale Aussage der Arbeit handelt und die Primärquellen aus der Universitätsbibliothek der Universität Stuttgart zum Recherchezeitpunkt aufgrund der pandemiebedingten Schließung aller Hochschulen in Baden-Württemberg nicht zur Verfügung standen, werden die verwendeten Quellen als ausreichend betrachtet.

## 9.5 Quellcode Radmesser Python

```
1 import RPi.GPIO as GPIO
2 import time
3 import os
4 import serial
5 import pynmea2
6 import string
7 import glob
8
9 # Temperatursensor erkennen und ansprechen , ist er nicht da
10 # -> keine Startroutine
11 base_dir='/sys/bus/w1/devices/'
12 device_folder = glob.glob(base_dir + '28*')[0]
13 print(device_folder)
14 device_file=device_folder+ '/w1_slave'
15
16 # Das erste Mal die serielle Schnittstelle abfragen
17 # Startzeit rund 17 Sekunden laut Datenblatt
18 serialPort = serial.Serial("/dev/ttyS0", 9600, timeout=2)
19
20 # File-Pointer definieren
```

```

21 file=open( 'testmessung00.txt' , 'w')
22 f=file
23 j=0
24 test="messdaten"
25 # bis eine neue Datei geschrieben werden kann
26 x=0
27 while (os.path.exists( "{}{}{}.txt".format( test , x ))):
28     x=x+1
29
30 # Voreinstellungen
31 GPIO.setmode(GPIO.BCM)
32 GPIO.setwarnings(False)
33 # Variablen definieren
34 PIN_TRIGGER=19
35 PIN_ECHOVorne=24
36 PIN_ECHOHinten=23
37 PIN_RICHTIG=21
38 PIN_FALSCH=20
39 PIN_DOWN=6
40 # PINs korrekt belegen
41 GPIO.setup(PIN_TRIGGER, GPIO.OUT)
42 GPIO.setup(PIN_ECHOVorne, GPIO.IN)
43 GPIO.setup(PIN_ECHOHinten, GPIO.IN)
44 GPIO.setup(PIN_RICHTIG, GPIO.OUT)
45 GPIO.setup(PIN_FALSCH, GPIO.OUT)
46 GPIO.setup(PIN_DOWN, GPIO.IN)
47 # Zu Beginn die OUTPUTs einstellen
48 GPIO.output(PIN_FALSCH, GPIO.HIGH)
49 GPIO.output(PIN_RICHTIG, GPIO.LOW)
50 GPIO.output(PIN_TRIGGER, GPIO.LOW)
51 time.sleep(0.5)
52 begin=time.time()
53
54 # Definition des Exit-Falls
55 def shutdown(channel):
56     f.close()
57     os.system("sudo shutdown -h now")
58 # Definition des Exit-Events
59 GPIO.add_event_detect(PIN_DOWN, GPIO.FALLING, callback=shutdown)
60
61 # Unterprogramm: ruft den Temperatursensor auf

```

```

62 def read_temp_raw():
63     ft=open(device_file,'r')
64     lines=ft.readlines()
65     ft.close()
66     return lines
67 # Unterprogramm wandelt Rueckmeldung des Temperatursensors in lesbares Format
68 def read_temp():
69     lines=read_temp_raw()
70     while (lines[0].strip()[-3:]!= 'YES'):
71         time.sleep(0.2)
72         lines=read_temp_raw()
73     equals_pos=lines[1].find('t=')
74     if (equals_pos!=-1):
75         temp_string=lines[1][equals_pos+2:]
76         temp_c=float(temp_string)/1000.0
77         # Temperatur wird um den Faktor 1000 zu gross
78         # -> siehe Datenblatt DS18B20
79     return temp_c
80
81 #Unterprogramm um beide Ultraschallsensoren gleichzeitig verwenden zu konnen
82 def UP():
83     #Trigger-Impuls fur beide Ultraschallsensoren setzen
84     GPIO.output(PIN_TRIGGER, GPIO.HIGH)
85     time.sleep(0.00001)
86     GPIO.output(PIN_TRIGGER, GPIO.LOW)
87     #Zeitvariablen definieren
88     t_begin=time.time()
89     t_start=time.time()
90     t_endvorne=t_start
91     t_endhinten=t_start
92     #Warten bis beide Echo-Pins 1 sind
93     while ((GPIO.input(PIN_ECHOVorne)==0)|(GPIO.input(PIN_ECHOHinten)==0)):
94         t_start=time.time()
95         #nach Zeituberschreitung (100ms) Schleife verlassen
96         if (t_start-t_begin>0.1):
97             break
98     #Warten bis beide wieder Rucksignal bekommen haben
99     while ((GPIO.input(PIN_ECHOVorne)==1)|(GPIO.input(PIN_ECHOHinten)==1)):
100         if (GPIO.input(PIN_ECHOVorne)==1):
101             t_endvorne=time.time()
102         if (GPIO.input(PIN_ECHOHinten)==1):

```

```

103         t_endhinten=time.time()
104 #Zeitdifferenz berechnen
105 t_diffvorne=t_endvorne-t_start
106 t_diffhinten=t_endhinten-t_start
107 #Strecke berechnen mit v=342m/s
108 wayvorne=t_diffvorne*17150
109 wayhinten=t_diffhinten*17150
110 #Allgemeine Zeit berechnen
111 t_gesamt=t_start-begin
112 #In Datei schreiben
113 f.write("%1.3f,%t_gesamt")
114 f.write("%2.1f,%wayvorne")
115 f.write("%2.1f,%wayhinten")
116 f.write("\n")
117 #LEDs zur Statusanzeige ansteuern
118 if((wayvorne<5)|(wayvorne>2000)|(wayhinten>2000)|(wayhinten<5)):
119     GPIO.output(PIN_FALSCH, GPIO.HIGH)
120 else:
121     GPIO.output(PIN_FALSCH, GPIO.LOW)
122 return 0
123
124 #Startroutine, Ultraschallsensoren werden auf Funktion getestet
125 GPIO.output(PIN_FALSCH, GPIO.HIGH)
126 messwert=0
127 while((messwert<10)|(messwert>500)):
128     GPIO.output(PIN_TRIGGER, GPIO.HIGH)
129     time.sleep(0.00001)
130     GPIO.output(PIN_TRIGGER, GPIO.LOW)
131     t_startup=time.time()
132     t_endup=t_startup
133     while(GPIO.input(PIN_ECHOVorne)==0):
134         t_startup=time.time()
135     while(GPIO.input(PIN_ECHOVorne)==1):
136         t_endup=time.time()
137     t_diffup=t_endup-t_startup
138     messwert=t_diffup*17150
139 messwert=0
140 while((messwert<10)|(messwert>500)):
141     GPIO.output(PIN_TRIGGER, GPIO.HIGH)
142     time.sleep(0.00001)
143     GPIO.output(PIN_TRIGGER, GPIO.LOW)

```

```

144     t_startup=time.time()
145     t_endup=t_startup
146     while(GPIO.input(PIN_ECHOHinteren)==0):
147         t_startup=time.time()
148         while(GPIO.input(PIN_ECHOHinteren)==1):
149             t_endup=time.time()
150             t_diffup=t_endup-t_startup
151             messwert=t_diffup*17150
152     #Test der Ultraschallsensoren erfolgreich abgeschlossen
153     GPIO.output(PIN_FALSCH, GPIO.LOW)
154     #Das GPS-Modul wird im Folgenden hoch gefahren
155     z=0
156     while(z<30):
157         serialPort = serial.Serial("/dev/ttyS0", 9600, timeout=2)
158         GPIO.output(PIN_RICHTIG, GPIO.HIGH)
159         GPIO.output(PIN_FALSCH, GPIO.HIGH)
160         time.sleep(0.5)
161         GPIO.output(PIN_RICHTIG, GPIO.LOW)
162         GPIO.output(PIN_FALSCH, GPIO.LOW)
163         time.sleep(0.5)
164         z=z+1
165     GPIO.output(PIN_RICHTIG, GPIO.HIGH)
166     GPIO.output(PIN_FALSCH, GPIO.LOW)
167     #an dieser Stelle ist noch ein Test des GPS-Moduls sinnvoll
168
169     while True:
170         messzahl=#Zahler fur Messungen
171         file=open("{0}{1}.txt".format(test, x), 'w')
172         # Datei mit entsprechender Nummer offnen
173         f=file
174         # verkuerzter Filepointer
175         temp_c=read_temp()
176         # Temperaturabfrage und in Datei schreiben
177         f.write("Temperatur,")
178         f.write("%f %temp_c")
179         f.write("\n")
180         while(messzahl<5000):
181             # 5000 Messungen in einer Datei
182             if((messzahl%100)==0):
183                 # alle 100 Messwerte wird GPS-Position hinzugefugt, ca. alle 2,5 Sekunden
184                 serialPort = serial.Serial("/dev/ttyS0", 9600, timeout=2)

```

```

185     # neue GPS-Position abfragen
186     str = serialPort.readline()
187     # Übermitteltes Protokoll lesen
188     f.write(str.decode())
189     # in Datei schreiben, encode -> Schnittstelle liefert Bytes
190     UP()#Messung durchföhren
191     messzahl=messzahl+1#Zahler für Messungen hochzählen
192     x=x+1#Zahler für nächste Datei
193     f.close()#alte Datei schließen
194     if(x>25000):
195         #wenn mehr als 25000 Dateien erstellt wurden gibt es einen Alarm
196         GPIO.output(PIN_RICHTIG, GPIO.LOW)
197         GPIO.output(PIN_FALSCH, GPIO.HIGH)
198     else:
199         GPIO.output(PIN_RICHTIG, GPIO.HIGH)

```

- Pakete einbinden Zeile 1 bis 7
- Temperatursensor ansprechen Zeile 11 bis 14, ist der Temperatursensor nicht vorhanden, kann es zu einem fatalen Fehler kommen und das Programm läuft nich los  
→ Nutzer erkennt dies daran, dass keine LED leuchtet
- GPS-Modul ansprechen um ausreichend Startzeit einzuräumen Zeile 18
- Den Dateizähler auf den aktuellen Wert setzen: Zeile 21 bis 28
- Pins einstellen und korrekt belegen, ist gegeben durch den Schaltplan und die Platine; Zeile 31 bis 52
- Definition des regulären Herunterfahrens mit Datei schließen und Befehl zum Shutdown; Zeile 55 bis 59
- Temperatursensor auslesen Zeile 62 bis 65, ähnlich Dateiaufruf, danach Rückmeldung in lesbares Format wandeln Zeile 68 bis 79
  - vorheriges UP aufrufen (Zeile 62 bis 65)
  - Solange die Rückmeldung ungültig ist ( $\neq$  YES) erneut abrufen
  - Korrekte Position des Temperaturwertes ermitteln (73)
  - Exakte Position des Temperaturwertes ansteuern (75)
  - Temperatur mit 1000 dividieren (76); Vgl. Datenblatt
- Ultraschallsensoren ansteuern Zeile 82 bis 122
  - 10 ms Impuls für beide Ultraschallsensoren (84-86)
  - Berechnungsvariablen definieren, alle Variablen zu Beginn definieren, um Re-

chenfehler am Ende zu vermeiden (bspw. *undefined t-end vorne*)

- Warten bis beide Sensoren den Ultraschall losgeschickt haben (93-97), wenn ein Sensor nicht erfolgreich war, wird nach 100 ms weitergegangen mit *break*-Befehl (96-97)
- Warten bis Ultraschall wieder detektiert wird von beiden Ultraschallsensoren (99); solange ein Ultraschallsensor keinen Schall detektiert hat die Endzeit aktualisieren (100-101 bzw. 102-103)
- Distanz bestimmen, Zeitdifferenz berechnen(105-106), mit Schallgeschwindigkeit multiplizieren (108-109) (leicht abweichend von Abschnitt 4.1.1)
- Alles formatiert in die Datei schreiben (113-116), Die Zeit mit drei Nachkommastellen und die Entfernungsangabe in cm mit einer Nachkommastelle, abschließend ein Zeilenumbruch
  - ist ein Wert kleiner als 5cm oder größer 12m leuchtet die Fehler-LED auf
- Test des vorderen Ultraschallsensors, Messwert muss zwischen 10cm und 5m liegen (126-138), währenddessen leuchtet die rote LED (125)
- Test des hinteren Ultraschallsensors, Messwert muss zwischen 10cm und 5m liegen (139-151), währenddessen leuchtet die rote LED (125); bleibt das Programm stehen, weil ein Ultraschallsensor fehlt, leuchtet die rote LED immer, ist der Test erfolgreich dann geht sie aus (153)
- Dem GPS-Modul wird die Startzeit eingeräumt und dem Nutzer der Start durch Blinken beider LEDs signalisiert (155-164)
- Während des Betriebs leuchtet nur die grüne LED (165-166)
- Hauptprogramm läuft immer ab (169-199)
  - Korrekte Datei öffnen (171-173) mit Nummer aus Zeile (21-28)
  - Temperatur in der Datei speichern (Vgl. 4.2) (175-179)
  - Alle 100 Abstandsmesswerte (182) wird die GPS-Position ermittelt (184-188)
  - 5000 Messwerte aufnehmen (180,190-191)
  - Zähler für die nächste Datei vorbereiten (192) und alte Datei schließen (193)
  - Bei Speicherüberfüllung warnen (25 000 Dateien ) (194-199), grüne LED geht aus

## 9.6 Inbetriebnahme Radmesser

### 9.6.1 Materialliste

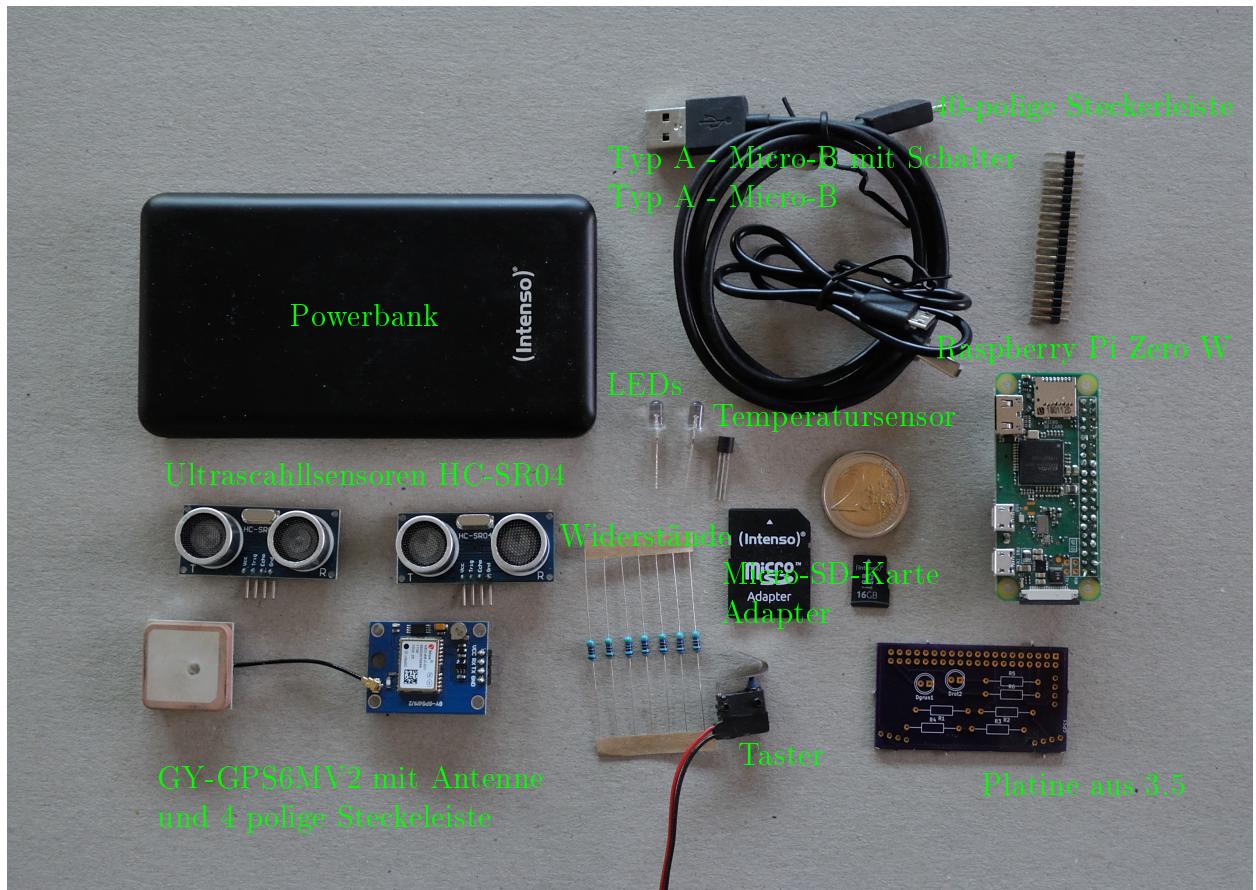


Abbildung 18: Benötigtes Material - zum Größenvergleich eine 2€-Münze

Bei manchen Produkten besteht die Möglichkeit leicht andere Varianten zu wählen. Bei den Widerständen ist die Farbcodierung in Klammern angegeben.

1. Raspberry Pi Zero (ab V 1.3) oder Raspberry Pi Zero W (ab V 1.3)
2. Powerbank mindestens 5000 mAh
3. Micro-SD-Karte mindestens 16 GB, maximal 256 GB, *Speed Class* 10 bzw. *Ultra High Speed Class* I
4. Kabel USB Typ A Stecker auf USB Micro-B-Stecker mit Schalter (Powerbank zu Raspberry PI, Spannungsversorgung)
5. Taster, Herunterfahren Raspberry Pi
6. Kabel USB Typ A Stecker auf USB Micro-B-Stecker (Aufladen der Powerbank)

7. Platine (siehe 3.5)
8. 2 Stück: Ultraschallsensoren HC-SR04
9. GPS-Modul NEO GPS6MV2
10. Temperatursensor DS18B20
11. 40 (2x20) polige Steckerleiste
12. 4 polige Steckerleiste
13. Widerstand  $4,7\text{k}\Omega$  (**GelbViolettRot** oder **GelbViolettSchwarzBraun**)
14. Low-Current LED grün und rot (maximal 2 mA)
15. 2 Stück Widerstand  $100\text{\Omega}$  (**BraunSchwarzBraun** oder **BraunSchwarzSchwarzSchwarz**)
16. 2 Stück Widerstand  $510\text{\Omega}$  (**GrünBraunBraun** oder **GrünBraunSchwarzSchwarz**)
17. 2 Stück Widerstand  $1\text{k}\Omega$  (**BraunSchwarzRot** oder **BraunSchwarzSchwarzBraun**)

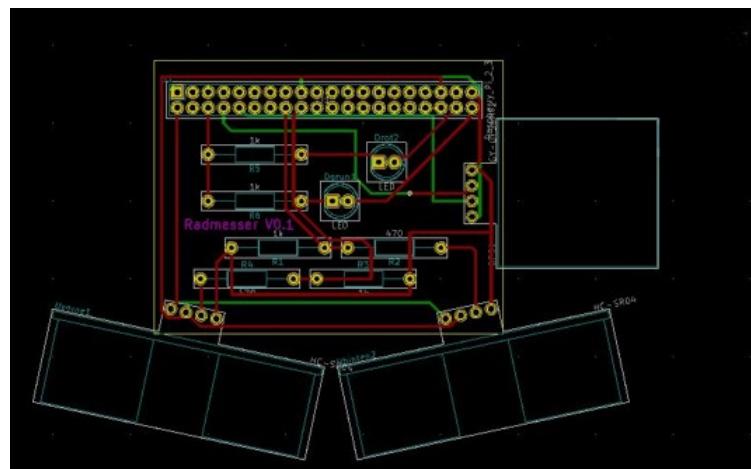
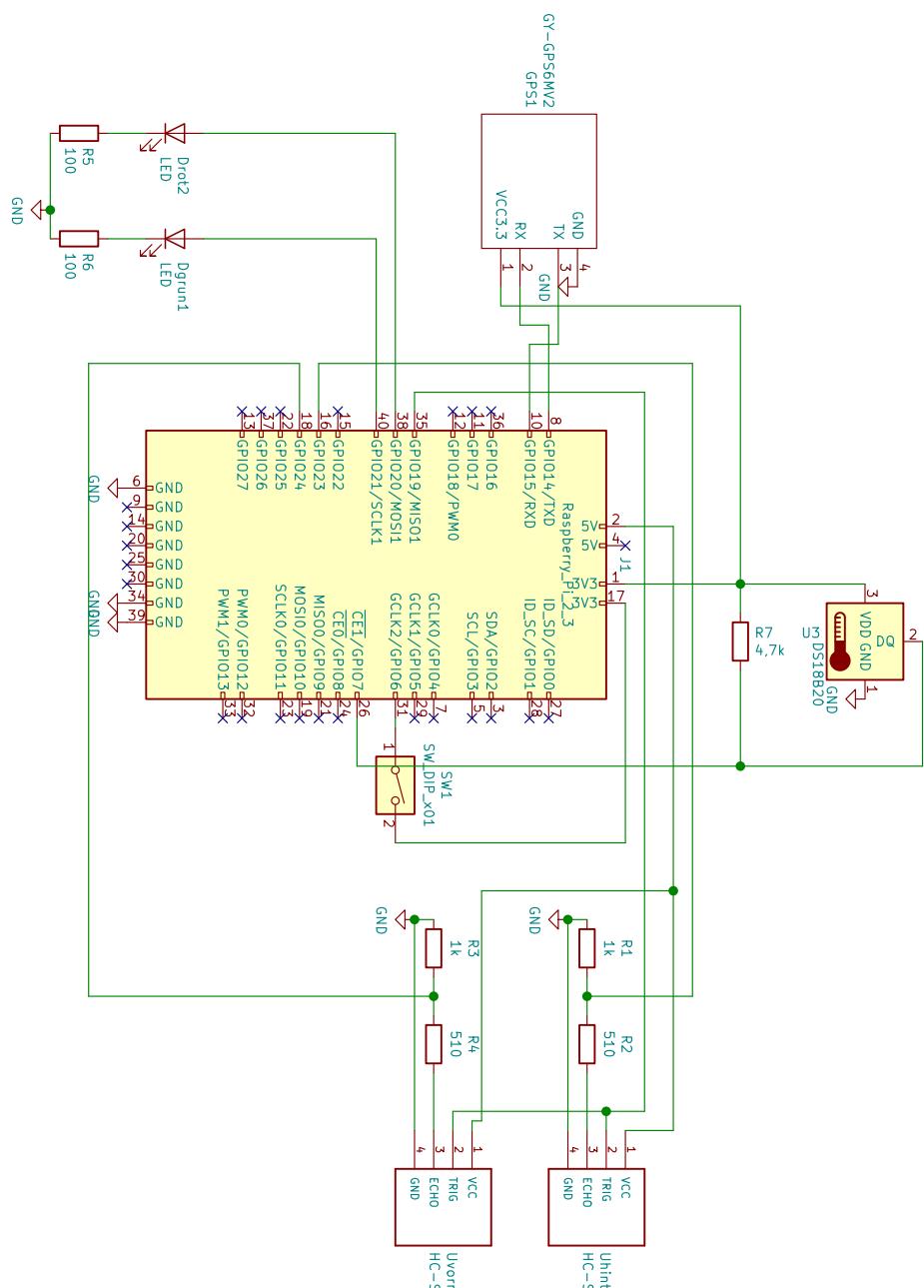


Abbildung 19: Leiterplatten Layout



|  |
|--|
| erstellt von Jakob Seitner             |
| <b>DHBW Stuttgart – Kesseleinstieg</b> |
| Sheet: /                               |
| File: Radmesser.sch                    |
| <b>Title: Radmesser</b>                |
| Size: A4                               |
| Date: 2020-05-19                       |
| Rev. V1.2                              |
| KICad E.D.A. kicad (5.0.2)-1           |
| Id: 1/1                                |
| 6                                      |

## **9.6.2 Bestückung der Platine**

Bei der Bestückung der oben (9.6.1) aufgeführten Bauteileliste ist auf die korrekte Reihenfolge zu achten. Es wird mit den Bauteilen, welche die geringe Höhe aufweisen begonnen:

1. Widerstände
2. Steckerleisten
  - a) GPS-Modul auf 4 polige Stiftleiste löten
  - b) Raspberry Pi auf 40 polige Stiftleiste löten
3. Temperatursensor
4. LEDs
5. Ultraschallsensoren
6. Taster

Der Raspberry Pi kann auch zu einem späteren Zeitpunkt ergänzt werden, wenn dieser über eine Steckverbindung angeschlossen wird

## **9.6.3 SD-Karte vorbereiten**

Für das Vorbereiten der Micro-SD-Karte ist ein Adapter (Mirco-SD auf SD) erforderlich und ein SD-Karten-Slot an einem System mit Windows 10. Auf Windows 10 muss dann die frei verfügbare Software *Win 32 Disk Imager* installiert werden. Mit Hilfe der Software wird dann eine *Image* (.img) Datei auf die SD-Karte geschrieben, diese enthält bereits das Betriebssystem des Raspberry Pis, die Radmesser-Software, alle notwendigen Softwareinstellungen (WLAN Verbindungen, Erweiterungspakete, Passwörter, Auto-Start) und Software um Fehler zu erkennen.

Wurde die Micro-SD-Karte erfolgreich beschrieben, kann diese in den Micro-SD-Karten-Slot des Raspberry Pis eingelegt werden, wo sie dauerhaft verbleibt.

## **9.6.4 Funktionsnachweis Radmesser**

Nach der Installation des Radmessers ist es nun erforderlich die einzelnen Funktionen des Radmessers zu prüfen, um ein problemloses Funktionieren sicherzustellen. Zu Fehlern kann es aufgrund einer falschen Hardwareinstallation kommen.

Es stehen zwei Möglichkeiten zur Verfügung die vollständige Funktion des Radmessers zu verifizieren, die eine basiert auf einer Remote-Desktop-Verbindung zum Raspberry Pi, die andere setzt auf das Anschließen eines Bildschirms über HDMI und das Anbinden von Maus und Tastatur an den Raspberry Pi.

Hierbei wird mit einem gesonderten Programm zuerst die Funktionsweise der beiden Ul-

traschallsensoren unabhängig voneinander getestet, danach wird die Funktion des Temperatursensors geprüft, abschließend wird das GPS-Modul getestet.

Für das Durchführen der Tests steht ein gesondertes Programm auf dem Raspberry Pi zur Verfügung.

```
1 import RPi.GPIO as GPIO
2 import time
3 import os
4 import serial
5 import pynmea2
6 import string
7 import glob
8
9 serialPort = serial.Serial("/dev/ttyS0", 9600, timeout=2)
10
11 # Voreinstellungen
12 GPIO.setmode(GPIO.BCM)
13 GPIO.setwarnings(False)
14 # Variablen definieren
15 PIN_TRIGGER=19
16 PIN_ECHOVorne=24
17 PIN_ECHOHinter=23
18 PIN_RICHTIG=21
19 PIN_FALSCH=20
20 PIN_DOWN=6
21 # PINs korrekt belegen
22 GPIO.setup(PIN_TRIGGER, GPIO.OUT)
23 GPIO.setup(PIN_ECHOVorne, GPIO.IN)
24 GPIO.setup(PIN_ECHOHinter, GPIO.IN)
25 GPIO.setup(PIN_RICHTIG, GPIO.OUT)
26 GPIO.setup(PIN_FALSCH, GPIO.OUT)
27 GPIO.setup(PIN_DOWN, GPIO.IN)
28 # Zu Beginn die OUTPUTs einstellen
29 GPIO.output(PIN_FALSCH, GPIO.HIGH)
30 GPIO.output(PIN_RICHTIG, GPIO.HIGH)
31 GPIO.output(PIN_TRIGGER, GPIO.LOW)
32 def shutdown(channel):
33     f.close()
34     os.system("sudo shutdown -h now")
35
36 GPIO.add_event_detect(PIN_DOWN, GPIO.FALLING, callback=shutdown)
37 # LEDS auf Anzeige pruefen
```

```

38 print("LED-TEST")
39 time.sleep(3)
40 GPIO.output(PIN_FALSCH, GPIO.LOW)
41 GPIO.output(PIN_RICHTIG, GPIO.LOW)
42
43 messwert=0
44 while ((messwert<10)|(messwert>500)):
45     GPIO.output(PIN_TRIGGER, GPIO.HIGH)
46     time.sleep(0.00001)
47     GPIO.output(PIN_TRIGGER, GPIO.LOW)
48     t_startup=time.time()
49     t_endup=t_startup
50     while(GPIO.input(PIN_ECHOVorne)==0):
51         t_startup=time.time()
52     while(GPIO.input(PIN_ECHOVorne)==1):
53         t_endup=time.time()
54     t_diffup=t_endup-t_startup
55     messwert=t_diffup*17150
56 print(messwert)
57 print("Vorne erfolgreich getestet")
58 messwert=0
59 while ((messwert<10)|(messwert>500)):
60     GPIO.output(PIN_TRIGGER, GPIO.HIGH)
61     time.sleep(0.00001)
62     GPIO.output(PIN_TRIGGER, GPIO.LOW)
63     t_startup=time.time()
64     t_endup=t_startup
65     while(GPIO.input(PIN_ECHOHinteren)==0):
66         t_startup=time.time()
67     while(GPIO.input(PIN_ECHOHinteren)==1):
68         t_endup=time.time()
69     t_diffup=t_endup-t_startup
70     messwert=t_diffup*17150
71 print(messwert)
72 print("Hinten erfolgreich getestet")
73
74 def read_temp_raw():
75     ft=open(device_file,'r')
76     lines=ft.readlines()
77     ft.close()
78     return lines

```

```

79 def read_temp():
80     lines=read_temp_raw()
81     while (lines[0].strip()[-3:]!= 'YES'):
82         time.sleep(0.2)
83         lines=read_temp_raw()
84     equals_pos=lines[1].find('t=')
85     if (equals_pos!=-1):
86         temp_string=lines[1][equals_pos+2:]
87         temp_c=float(temp_string)/1000.0
88     return temp_c
89
90 base_dir='/sys/bus/w1/devices/'
91 device_folder = glob.glob(base_dir + '28*')[0]
92 print(device_folder)
93 device_file=device_folder+'/w1_slave'
94 temp_c=read_temp()
95 print("%f"%temp_c)
96 if ((temp_c<50)&(temp_c>5)):
97     print("Temperatur erfolgreich getestet")
98 else:
99     print("Temperatur fehlerhaft")
100
101 serialPort = serial.Serial("/dev/ttyS0", 9600, timeout=2)
102 str =serialPort.readline()
103 print(str.decode())
104 if(str!=""):
105     print("GPS-Modul vorhanden")
106 else:
107     while(str!=""):
108         serialPort = serial.Serial("/dev/ttyS0", 9600, timeout=2)
109         str =serialPort.readline()

```

## 9.6.5 Handhabung im täglichen Einsatz

Die Handhabung im täglichen Einsatz ist denkbar einfach, um den Radmesser zu starten, wird der Schalter in der Stromzufuhr des Raspberry Pis auf ein gestellt. Während die beiden LEDs (rot und grün) im Gleichtakt blinken startet der Radmesser, dies dauert circa 30 Sekunden. Danach leuchtet nur noch die grüne LED, dann ist der Radmesser in seinem normalen Betriebszustand. Leuchtet die rote LED hingegen dauerhaft ist ein Fehler aufgetreten, selbiges gilt, wenn die grüne LED nicht mehr leuchtet. Dann muss eine

spezifische Fehlersuche durchgeführt werden.

Zum Beenden einer Messungen wird der Taster betätigt, kurz darauf erlöschen die LEDs, nach weiteren 20 Sekunden kann die Stromversorgung dann über den Ein/Aus-Schalter getrennt werden.

**Sammeln der Daten** Entweder wird auf den Raspberry Pi über eine anfänglich konfigurierte Remote-Dektop-Verbindung zugegriffen und die Daten dann über einen gemeinsamen Ordner von Raspberry Pi und Computer übertragen.

Ansonsten kann die Micro-SD-Karte ausgebaut werden und an eine Person weitergereicht, welche den oben beschriebenen Prozess durchführt.

## 9.7 Quellcode Auswertungssoftware D

```
1 module main;
2
3 import std.stdio;
4 import std.typecons;
5 import std.csv;
6 import std.file;
7 import std.array;
8 import std.string;
9 import std.conv;
10 import std.algorithm.searching;
11 import std.math;
12
13 const float amind=60; //mindest Abstand, muss bei einem Fahrrad
14 // circa 60cm sein (Lenker und Aussenspiegel)
15 const float amax=1400; //Grenze muss höher gesetzt werden, aber
16 // tests wurden in kleiner Wohnung durchgefuehrt
17
18 const float amaxuber=400;
19 const float lower=0.95;
20 const float higher=1.05;
21 const int empfindlichkeit=15;
22 const int empfindlichkeitstart=1;
23 const float mindanachuber=1000;
24 const float strasse =350.0;//Strassenbreite
```

```

24 int globalcount=0;
25 struct measuredata{
26     int number;
27     float time;
28     float vorne;
29     float hinten;
30     float utctime;
31     float north;
32     float east;
33     float date;
34     float speed;
35     float orientation=-1.0;
36     float temp=17.0;
37
38     float overtaking;
39     float speedKFZ;
40     float lenghtKFZ;
41 } ;
42
43 struct geodata{
44     float utctime;
45     float north;
46     float east;
47 } ;
48
49
50 struct fahrt{
51     measuredata [] inputdata;
52     geodata [] geodaten;
53     float gesamtstrecke;
54     measuredata [] overtaking;
55 } ;
56
57 fahrt current_drive;
58 float [][] readdata; //alle Messwerte
59 float [][] gpsdata;
60
61 int countentiredata;
62 fahrt [] entiredata;

```

```

63
64 float coordinateconvert( string eingang){
65     float startvalue=to!double(eingang);
66     int intvalue=to!int(startvalue/100);
67     float floatvalue=startvalue-intvalue*100;
68     float returnvalue=intvalue+floatvalue/60;
69     return returnvalue;
70 }
71
72 measuredata[] readcsv( string filename){ //CSV–Datei einlesen
73     measuredata[] inputdata;
74     File file=File(filename, "r"); //Dateiname kommt aus
        Hauptprogramm
75     float utctime;
76     float north;
77     float east;
78     float date;
79     float speed;
80     float orientation=-1.0;
81     float temp=17.0;
82     float oldutctime;
83
84     while (!file.eof()){ //Es wird immer bis zum Dateiende
        gelesen
85         globalcount++; //Zahler fur alle Zeilen
86
87         string line=file.readln(); //Einzelne Zeile Einlesen als
            String
88         if (canFind(line, "Temperatur")){
89             auto split1=line.split(",");
90             temp=to!double(split1[1]);
91         } else if (count(line, ",")>2){ //Die letzte Zeile jeder
            Datei ist leer, deshalb nur die vollen Zeilen
            verwenden
92             auto split1=line.split(","); //Da es sich um eine CSV
                –Datei handelt Spalten an Komma trennen
93             //write(split1); //Ausgabe der aufgetrennten Zeile
94             //writeln(" test"); //useless

```

```

95      if (split1[0]==="GPS" || canFind(line , "N") || canFind
96          (line , "*") || canFind(line , "E") || canFind(line
97          , "V") || canFind(line , "F") || canFind(line , "C")
98          || canFind(line , "B") || canFind(line , "GPRMC")){
99          //Zeile wird als GPS-Inhalt erkannt und gesondert
100         behandelt
101
102         if (canFind(line , "GPRMC")===1){
103             auto splitgps=line .split (" , ");
104             if (splitgps [2]==="A"){
105                 oldutctime=utctime ;
106                 utctime=to !double (splitgps [1]);
107                 if (splitgps [4]==="N"){
108                     north=coordinateconvert (splitgps [3])
109                     ;
110                 }
111                 if (splitgps [4]==="S"){
112                     north=-1*coordinateconvert (splitgps
113                         [3]) ;
114                 }
115                 if (splitgps [6]==="E"){
116                     east=coordinateconvert (splitgps [5]);
117                 }
118                 if (splitgps [6]==="W"){
119                     east=-1*coordinateconvert (splitgps
120                         [5]) ;
121                 }
122                 float speedk=to !double (splitgps [7]);
123                 speed=speedk /0.53996;
124                 date=to !double (splitgps [9]);
125                 if (splitgps [8]!=""){
126                     orientation=to !double (splitgps [8]);
127                 }
128                 else {
129                     orientation=-1.0;
130                 }
131             }
132             //write (splitgps );

```

```

125         if( utctime-oldutctime <0 || utctime-oldutctime
126             >150){
127             current_drive=automatisch_erkennen(
128                 current_drive);
129             entiredata~ =current_drive;
130             destroy( current_drive);
131             //current_drive=[];
132         }
133         geodata newgeodata;
134         newgeodata.utctime=utctime;
135         newgeodata.north=north;
136         newgeodata.east=east;
137         current_drive.geodaten~ =newgeodata;
138     }
139 } else{//Zeile enthält normale Messwerte, nicht GPS
140     auto splittime1=(split1[0]).split(" ");
141     double time=0;
142     int leerzeichenout;
143     for(int leerzeichen; (splittime1[leerzeichen])==
144         " ";leerzeichen++){
145         leerzeichenout=leerzeichen+1;
146         //write(leerzeichen);
147     }
148     if(canFind(splittime1, "\n")){}
149     else{
150         time=to!double(splittime1[leerzeichenout]);
151         auto split2=(split1[1]).split(" ");
152         auto split3=(split1[2]).split(" ");
153         if(split2[0]!=""&&split3[0]!=""){
154             double f1=to!double(split2[0]); //hinten
155             double f2=to!double(split3[0]); //vorne
156             if(f1>amind&&f2>amind&&f1<amax&&f2<amax)
157                 { //Distanz kleiner mindestanforderung
158                     , also auserhalb Fahrradlenker oder
159                     groesser wie maximalmoeglich,
160                     technische Grenze HC-SR04 oder
161                     Begrenzung der Testumgebung: delete
162                     measuredata input;
163                     input.time=time;
164                 }
165             }
166         }
167     }
168 }

```

```

156         input.vorne=f1;
157         input.hinten=f2;
158         input.utctime=utctime;
159         input.north=north;
160         input.east=east;
161         input.date=date;
162         input.temp=temp;
163         input.speed=speed;
164         input.orientation=orientation;
165         inputdata~=input;
166         current_drive.inputdata~=input;
167     }
168 }
169 }
170 }
171 }
172 }
173 }
174 file.close();
175 return inputdata;
176 }
177
178 fahrt automatisch_erkennen(fahrt aktuelle_fahrt){
179     aktuelle_fahrt.inputdata=glattung(aktuelle_fahrt.inputdata);
180     aktuelle_fahrt.overtaking=erkennen(aktuelle_fahrt.inputdata);
181     ;
182     aktuelle_fahrt.gesamtstrecke=calcualtedistance(
183         aktuelle_fahrt.geodaten);
184     aktuelle_fahrt.inputdata=null;
185     aktuelle_fahrt.geodaten=null;
186     return aktuelle_fahrt;
187 }
188
187 measuredata[] glattung(measuredata[] ret){
188     int zeilen=count(ret);
189     float faktor=0.9;
190     float faktor2=1.1;
191     for(int i=1; i<(zeilen-1); i++){

```

```

192     if ((ret[i-1].vorne*faktor)>ret[i].vorne && (faktor*ret[i
193         +1].vorne)>ret[i].vorne){
194         ret[i].vorne=(ret[i-1].vorne+ret[i+1].vorne)/2;
195     }
196     if ((ret[i-1].hinten*faktor)>ret[i].hinten && (faktor*ret
197         [i+1].hinten)>ret[i].hinten){
198         ret[i].hinten=(ret[i-1].hinten+ret[i+1].hinten)/2;
199     }
200     if ((ret[i-1].vorne*faktor2)<ret[i].vorne && (faktor2*ret
201         [i+1].vorne)<ret[i].vorne){
202         ret[i].vorne=(ret[i-1].vorne+ret[i+1].vorne)/2;
203     }
204 }
205
206     return ret;
207 }
208
209 int writecsv( string filename , float [][] writeit){ //UP zur
210     Ausgabe einer CSV–Datei, Name wird in Main angegeben —
211     automatisch adaptierte Breite
212     int zeilencount ;
213     File file=File(filename , "w");
214     foreach(i ; writeit){
215         int zeilencount2 ;
216         foreach(j ; writeit[zeilencount]){
217             file . write( writeit[zeilencount][zeilencount2] , " , " );
218             zeilencount2++;
219         }
220         file . write( "\n" );
221         zeilencount++;
222     }
223     file . close();
224     return zeilencount ;
225 }

```

```

225 int writecsvfromstruct( string filename , measuredata[] inputdata ,
226   bool inorub){
227   File f=File( filename , "w" );
228   if( inorub ){
229     f . writeln( " Zeitstempel , vorne , hinten , temp , Zeit , Datum ,
230       Lange , Breite , Geschw , Orientierung " );
231     foreach( measuredata input; inputdata ){
232       f . write( input . time , " , " );
233       f . write( input . vorne , " , " );
234       f . write( input . hinten , " , " );
235       f . write( input . temp , " , " );
236       f . write( input . utctime , " , " );
237       f . write( input . date , " , " );
238       f . write( input . north , " , " );
239       f . write( input . east , " , " );
240       f . write( input . speed , " , " );
241       f . write( input . orientation , " , " );
242       f . write( "\n" );
243     }
244   } else {
245     f . writeln( " Abstand , Zeit , Datum , Lange , Breite ,
246       Geschwindigkeit Fahrrad , Geschwindigkeit KFZ , Lange
247       KFZ , " );
248     foreach( measuredata input; inputdata ){
249       f . write( input . overtaking , " , " );
250       f . write( input . utctime , " , " );
251       f . write( input . date , " , " );
252       f . write( input . north , " , " );
253       f . write( input . east , " , " );
254       f . write( input . speed , " , " );
255       f . write( input . speedKFZ , " , " );
256       f . write( input . lenghtKFZ , " , " );
257       f . write( "\n" );
258     }
259   }
260   f . close();
261   return 100;
262 }
```

```

259 int writecsv2(string filename, float[][] writeit){ //UP zur
  Ausgabe einer CSV–Datei, Name wird in Main angegeben
260   int zeilencount;
261   File file=File(filename, "w");
262   foreach(i; writeit){
263     file.writeLn(writeit[zeilencount][0], ",",
      writeit[zeilencount][1], ",",
      writeit[zeilencount][2], ",",
      writeit[zeilencount][3], ",",
      writeit[zeilencount][4], ",",
      writeit[zeilencount][5], ",",
      writeit[zeilencount][6], ",",
      writeit[zeilencount][7]);
264     zeilencount++;
265   }
266   file.close();
267   return zeilencount;
268 }
269
270 measuredata[] erkennen(measuredata[] inputdata){
271   measuredata[] erkannt;
272   float[] uberhol;
273   float amindest=strasse+amind;
274   int flaguber;
275   int flaguber2;
276   int flagunter;
277   float lastuberholtime=0;
278   float begin1;
279   float end1;
280   foreach(measuredata input; inputdata){
281     float vorne=input.vorne;
282     float hinten=input.hinten;
283
284     if((input.time)>20.0){//alles innerhalb der ersten 20
      Sekunden wird nicht berücksichtigt
285       if( (vorne*higher<hinten) && vorne<amindest ){ ////
      vorderer Wert ist um über 40 Prozent kleiner als
      hinterer, überholvorgang erst ab 2,5m erkennen
286         if(flaguber==0){
287           begin1=input.time;
288         }
289         flaguber++; //Marker setzen

```

```

290         flagunter=0;
291     } else if( (hinten<amindest) && (vorne<amindest) && (
292         flaguber>empfindlichkeitstart) && (hinten*lower)<
293         vorne && (hinten*higher)>vorne ){
294         if (flaguber2==0){
295             end1=input.time;
296         }
297         uberhol~=(vorne+hinten)/2;
298         flaguber2++;
299         flagunter=0;
300     } else if( (vorne*lower)>hinten && (vorne>amindest)
301         && (flaguber2>empfindlichkeit) ){
302         int numberarr=uberhol.count();
303         float gesamtuberhol=0;
304         for(int zeilencount ; zeilencount<numberarr ;
305             zeilencount++){
306             gesamtuberhol=gesamtuberhol+uberhol[
307                 zeilencount];
308         }
309         float timeforspeed=end1-begin1;
310         float timeforlength=input.time-end1;
311         float overtaking=gesamtuberhol/numberarr;
312         float distanceforspeed=overtaking/20;
313         float speed=(distanceforspeed)/timeforspeed;
314         float length=(speed*timeforlength)/25;
315
316         measuredata.erkannt1;
317         erkannt1=input;
318         erkannt1.speedKFZ=input.speed+speed;
319         erkannt1.lengthKFZ=length;
320         erkannt1.overtaking=overtaking;
321         //erkannt1.speedKFZ=flaguber;
322         //erkannt1.lengthKFZ=bearbeitet[i][3];
323         float timediff=input.time-lastuberholtme;
324         if (numberarr>3 && input.time-lastuberholtme>1){
325             erkannt~=erkannt1;
326         }
327         uberhol.destroy();
328         lastuberholtme=input.time;

```

```

324         //erkannt~=bearbeitet [ i-flaguber2 /2];
325         flaguber=0;
326         flaguber2=0;
327     } else {
328         flagunter++;
329     }
330     if (flagunter >50){
331         flaguber=0;
332         flaguber2=0;
333         flagunter=0;
334     }
335 }
336
337 }
338 return erkannt;
339 }
340
341 float calcualtedistance(geodata[] geodatafordistance){
342     float drivendistance=0;
343     int numberofdata=count(geodatafordistance);
344     //writeln(numberofdata);
345     for (int i=0; i<numberofdata-1;i++){
346         float timedif=geodatafordistance[i+1].utctime-
347             geodatafordistance[i].utctime;
348         //writeln(timedif);
349         if (timedif>0&&timedif<50){
350             float eastdif=geodatafordistance[i+1].east-
351                 geodatafordistance[i].east;
352             float northdif=geodatafordistance[i+1].north-
353                 geodatafordistance[i].north;
354             float eastdifcos=eastdif*cos(geodatafordistance[i].
355                 north);
356             float dist=111.3*sqrt(northdif*northdif+eastdifcos*
357                 eastdifcos);
358             //writeln(dist);
359             drivendistance=drivendistance+dist;
360         }
361     }
362     return drivendistance;

```

```

358 }
359
360 void geojsononline(string filename, float [[[ writeit ){
361     int zeilencount;
362     File file=File(filename, "w");
363     file.writeln("{");
364     file.writeln("\"type\": \"FeatureCollection\",");
365     file.writeln("\"features\": [");
366     file.writeln("{");
367     file.writeln("\"type\": \"Feature\",");
368     file.writeln("\"properties\":{},");
369     file.writeln("\"geometry\":{");
370     file.writeln("\"type\": \"LineString\",");
371     file.writeln("\"coordinates\":[" );
372     foreach(i; writeit){
373         float north=writeit[zeilencount][3];
374         float east=writeit[zeilencount][4];
375         float abstand=writeit[zeilencount][0];
376         if(north<90.0 && east<180.0 && abstand<amaxuber &&
377             abstand>amind/*&& writeit[zeilencount][2]<amaxuber*/)
378         {
379             file.writeln("[" );
380             file.writeln(east, ", ");
381             file.writeln(north);
382             file.writeln("], ");
383         }
384         file.writeln("] ");
385         file.writeln("} ");
386         file.writeln("} ");
387         file.writeln("] ");
388         file.writeln("} ");
389     }
390
391     int writegeojson(string filename, measuredata[] writedata){
392         int zeilencount;
393         File file=File(filename, "w");
394         file.writeln("{");

```

```

395     file . writeln( "\\" type \": \" FeatureCollection \", " );
396     file . writeln( "\\" features \": [ " );
397     foreach( measuredata  writedata ; writedata ){
398         float  north=writedata . north ;
399         float  east=writedata . east ;
400         float  abstand=writedata . overtaking ;
401         if ( north < 90.0 && east < 180.0 && abstand < amaxuber &&
402             abstand > amind /*&& writeit [ zeilencount ][2] < amaxuber */ )
403         {
404             file . writeln( " { " );
405             file . writeln( "\\" type \": \" Feature \", " );
406             file . writeln( "\\" properties \": { " );
407             // writeln( abstand );
408             if ( abstand < (amind+50) ){
409                 file . writeln( "\\" marker-color \": \" #930000 \\" );
410             }
411             if ( abstand < (amind+100) && abstand > (amind+50) ){
412                 file . writeln( "\\" marker-color \": \" #e2001a \\" );
413             }
414             if ( abstand < (amind+150) && abstand > (amind+100) ){
415                 file . writeln( "\\" marker-color \": \" #ffff00 \\" );
416             }
417             if ( abstand > (amind+150) ){
418                 file . writeln( "\\" marker-color \": \" #46812b \\" );
419             }
420             file . writeln( " } , " );
421             file . writeln( "\\" geometry \": { " );
422             file . writeln( "\\" type \": \" Point \", " );
423             file . writeln( "\\" coordinates \": [ " );
424             file . writeln( east , " , " );
425             file . writeln( north );
426             file . writeln( " ] " );
427             file . writeln( " } " );
428             file . writeln( " } , " );
429         }
430         zeilencount++;
431     }
        file . writeln( " ] " );
        file . writeln( " } " );

```

```

432     return zeilencount;
433 }
434
435 int writegeojson einfach( string filename , measuredata[] writedata
) {
436     float [][] green;
437     float [][] yellow;
438     float [][] red;
439     float [][] darkred;
440     foreach( measuredata writedat ; writedata ) {
441         float north=writedat .north ;
442         float east=writedat .east ;
443         float abstand=writedat .overtaking ;
444         if ( north < 90.0 && east < 180.0 && abstand < amaxuber &&
abstand > amind /*&& writeit [zeilencount ][2] < amaxuber */ )
{
445             if ( abstand < (amind+50)) {
446                 darkred ~=[north , east ];
447             } else if ( abstand < (amind+100) && abstand > (amind+50) )
{
448                 red ~=[north , east ];
449             } else if ( abstand < (amind+150) && abstand > (amind+100) )
{
450                 yellow ~=[north , east ];
451             } else if ( abstand > (amind+150) ) {
452                 green ~=[north , east ];
453             }
454         }
455     }
456
457     File file=File (filename , "w" );
458     file .writeln ("{"); //Gesamtklammer
459     file .writeln ("\\" type \": \"FeatureCollection\",");
460     file .writeln ("\\" features \": ["); //allFeatures
461
462     file .writeln ("{"); //erstes Feature
463     file .writeln ("\\" type \": \"Feature\",");
464     file .writeln ("\\" properties \": {\"marker-color \":
\">#930000\"},");

```

```

465   file .writeln( "\\"geometry\":{ " ); //geometry klammer
466   file .writeln( "\\"type\": \"MultiPoint\", " );
467   file .writeln( "\\"coordinates\": [ " ); //Koordinaten
468   foreach( int i , float [] coordinates ; darkred ){
469     if( i<count( darkred )-1){
470       file .writeln( " [ " , coordinates [ 1 ] , " , " , coordinates [ 0 ] , "
471           ] , " );
472     } else {
473       file .writeln( " [ " , coordinates [ 1 ] , " , " , coordinates [ 0 ] , "
474           ] " );
475     }
476     file .writeln( " ] " ); //Koordinaten
477     file .writeln( " } " ); //geometry
478     file .writeln( " } , " ); //erstes Feature
479
480     file .writeln( " { " ); //zweites Feature
481     file .writeln( "\\"type\": \"Feature\", " );
482     file .writeln( "\\"properties\": {\"marker-color\": \"#e2001a
483           \"}, " );
484     file .writeln( "\\"geometry\":{ " ); //geometry klammer
485     file .writeln( "\\"type\": \"MultiPoint\", " );
486     file .writeln( "\\"coordinates\": [ " ); //Koordinaten
487     foreach( int i , float [] coordinates ; red ){
488       if( i<count( red )-1{
489         file .writeln( " [ " , coordinates [ 1 ] , " , " , coordinates [ 0 ] , "
490             ] , " );
491       } else {
492         file .writeln( " [ " , coordinates [ 1 ] , " , " , coordinates [ 0 ] , "
493             ] " );
494       }
495     }
496     file .writeln( " ] " ); //Koordinaten
497     file .writeln( " } " ); //geometry
498     file .writeln( " } , " ); //zweites Feature
499
500     file .writeln( " { " ); //dritttes Feature
501     file .writeln( "\\"type\": \"Feature\", " );

```

```

498     file .writeln( "\\" properties \": {\"marker-color \": \"#ffff00
      \"}, " );
499     file .writeln( "\\" geometry \":{ " ); //geometry klammer
500     file .writeln( "\\" type \": \"MultiPoint\", " );
501     file .writeln( "\\" coordinates \": [ " ); //Koordinaten
502     foreach( int i , float [] coordinates ; yellow ){
503         if( i<count( yellow )-1){
504             file .writeln( "[ " , coordinates [ 1 ] , " , " , coordinates [ 0 ] ,
505                           " ] , " );
506         } else {
507             file .writeln( "[ " , coordinates [ 1 ] , " , " , coordinates [ 0 ] ,
508                           " ] " );
509         }
510         file .writeln( " ] " ); //Koordinaten
511         file .writeln( " } " ); //geometry
512         file .writeln( " } , " ); //dritt es Feature
513
514         file .writeln( " { " ); //vier tes Feature
515         file .writeln( "\\" type \": \"Feature\", " );
516         file .writeln( "\\" properties \": {\"marker-color \": \"#46812b
      \"}, " );
517         file .writeln( "\\" geometry \":{ " ); //geometry klammer
518         file .writeln( "\\" type \": \"MultiPoint\", " );
519         file .writeln( "\\" coordinates \": [ " ); //Koordinaten
520         foreach( int i , float [] coordinates ; green ){
521             if( i<count( green )-1){
522                 file .writeln( "[ " , coordinates [ 1 ] , " , " , coordinates [ 0 ] ,
523                               " ] , " );
524             } else {
525                 file .writeln( "[ " , coordinates [ 1 ] , " , " , coordinates [ 0 ] ,
526                               " ] " );
527             }
528             file .writeln( " ] " ); //Koordinaten
529             file .writeln( " } " ); //geometry
530             file .writeln( " } , " ); //vier tes Feature
531
532             file .writeln( " ] " ); //allFeatures schliessen

```

```

531     file . writeln ( " } " ) ; //Gesamtklammer schliessen
532     return 100 ;
533 }
534
535 int main( string [] args )
536 {
537     int x=0;
538
539     string filenumber=to ! string (x) ;
540     string filename=join ( [ " messdaten " ,filenumber ,".txt" ] ) ;
541     while( exists (filename) ){
542         //write (filename) ;
543         auto inputdata=readcsv (filename) ;
544         x++;
545         filenumber=to ! string (x) ;
546         filename=join ( [ " messdaten " ,filenumber ,".txt" ] ) ;
547         //write (filename) ;
548     }
549     if ( count ( current _ drive .inputdata )!=0 ){
550         current _ drive=automatisch _ erkennen (current _ drive) ;
551         entiredata ~ =current _ drive ;
552         destroy (current _ drive) ;
553     }
554     //int intret=writecsvfromstruct (" messdaten000.csv " ,
555     //inputdata , true ) ;
556     //writeln (" Es wurden " ,intret , " Zeilen geschrieben " );
557
558     //measuredata [] overtakedata=erkennen (inputdata) ;
559     measuredata [] overtakedata ;
560     float gesamtdistanz=0;
561     //writeln (count (entiredata)) ;
562     foreach ( int i ,fahrt aktuelle _ fahrt ;entiredata ){
563         overtakedata ~ =aktuelle _ fahrt .overtaking ;
564         gesamtdistanz=gesamtdistanz+aktuelle _ fahrt .gesamtstrecke
565         ;
566         destroy (entiredata [ i ]) ;
567     }
568     writeln (gesamtdistanz) ;

```

```

567     int intret2=writecsvfromstruct("erkannte.csv", overtakedata,
568                                     false);
569     int intret4=writegeojson("Geodata_overtake.geojson",
570                               overtakedata);
571     int intret6=writegeojsoneinfach("Geodata_overtake_simple.
572                                     geojson", overtakedata);
573     writeln(intret2, "      ", intret4);
574     //geojsononline("line.geojson", ret);
575
576     return 0;
577 }
```

in runden Klammern werden die Zeilenzahlen angegeben

- Pakete einbinden, Variablen definieren und Strukturen festlegen (1-62)
- UP (*coordinateconvert*) zur Konvertierung von Dezimalminuten in Dezimalgrade (Vgl. Abschnitt 5.1) (64-70)
- UP (*readcsv*) zum Einlesen aller Daten (72-176) unter
  - Koordinatenumwandlung (102, 105, 108, 111)
  - Knoten in km/h umrechnen (114)
  - Aufteilen in einzelne Fahrten (125)
- UP (*automatisch-erkennen*) zur Behandlung der einzelnen Fahrten, ansonsten können nur Messdaten von bis zu 100 h gleichzeitig ausgewertet werden (178-185)
- UP (*Glattung*) (Vgl. ??) zur Ausbesserung von Messfehlern (187-207)
- UP um CSV-Dateien zu schreiben nicht näher zu erklären
- UP (*erkennen*) Daten einer Fahrt werden übergeben und Überholvorgänge extrahiert (270-339); Vgl. Abschnitt 5.2
  - vorne kleiner als hinten (285)
  - beide gleich groß, aber kleiner Mindestdistanz (291)
  - hinten groß und vorne klein (298)
  - doch kein Überholvorgang, wenn 50 Mal die Messwerte nicht den Anforderungen entsprechen (330)
- UP (*calculatedistance*) zur Berechnung der gefahrenen Strecke (341-358); Vgl. Abschnitt 5.3
- weitere UP zur Ausgabe von GeoJSON-Dateien
- *main*-Funktion

- alle Dateien finden (541-548)
- Dateien einlesen über UP (543)
- Überholvorgänge der einzelnen Fahrten zusammenfügen (549-553)
- Berechnung Gesamtdistanz mit UP (559-565)
- Ausgabe verschiedener GeoJSON-Dateien (567-569)

## 9.8 Adressabfrage in R

```
1 install.packages("opencage")
2 library("opencage")
3 apiKey='YOURAPIKEY'
4 input <- read.csv(file='D:/GPS.csv')
5 output <- data.frame(Number=c(), Distance=c(), LAT=c(), LON=c(),
6   speed=c(), date=c(), time=c(), country=c(), state=c(),
7   citytown=c(), postcode=c(), roadname=c(), roadtype=c(),
8   oneway=c(), maxspeed=c(), speedin=c(), lanes=c())
9 for(i in 1:5){
10   LAT<-input[i,1]
11   LON<-input[i,2]
12   dist<-0
13   speed<-0
14   date<-0
15   time<-0
16   citytown<-0
17   lanes<-1
18   oneway<-0
19   output1 <- opencage_reverse(latitude=LAT, longitude=LON, key='
20     YOURAPIKEY', language='de')
21   if(output1$rate_info$remaining>0){
22     output2 <- output1$results
23     country <- output2$components.country
24     state <- output2$components.state
25     if('components.city' %in% names(output2)){
26       citytown<-output2$components.city
27     }
28     if('components.town' %in% names(output2)){
29       citytown<-output2$components.town
30     }else if('components.village' %in% names(output2)){
31       citytown<-output2$components.village
32     }
33     postcode <- output2$components.postcode
34     roadname <- output2$annotations.roadinfo.road
35     roadtype <- output2$annotations.roadinfo.road_type
36     maxspeed <- output2$annotations.roadinfo.maxspeed
37     speedin <- output2$annotations.roadinfo.speed_in
```

```

34 if ('annotations.roadinfo.lanes' %in% names(output2)){
35   lanes<-output2$annotations.roadinfo.lanes
36 }
37 if ('annotations.roadinfo.oneway' %in% names(output2)){
38   oneway<-output2$annotations.roadinfo.oneway
39 }
40 out <- data.frame(Number=i, Distance=dist, LAT=LAT, LON=LON,
41   speed=speed, date=date, time=time, country=country,
42   state=state, citytown=citytown, postcode=postcode,
43   roadname=roadname, roadtype=roadtype, oneway=oneway,
44   maxspeed=maxspeed, speedin=speedin, lanes=lanes)
45 print(out)
46 output <- rbind(output, out)
47 }
48 }
49 write.csv(output, 'D:/apiabfrage.csv')

```

- Es wird eine Variante des Pakets *openrage* benötigt, verfügbar unter *juliusgh/openrage* (1-2)
- Datei mit Überholvorgängen einlesen (4)
- Datenrahmen bzw. Datenstruktur definieren (5)
- in der Beispieldatei sind fünf Koordinaten enthalten (6)
- Eigentliche API-Abfrage (16)
- Je nach Stadtgröße steht entweder die Information über den Stadtnamen in *city*, *town* oder *village*
- Datenstruktur mit der neuen Adresse ergänzen (40)
- Gesamtadressliste ausgeben (45)

## 9.9 Zusätzliche Diagramme

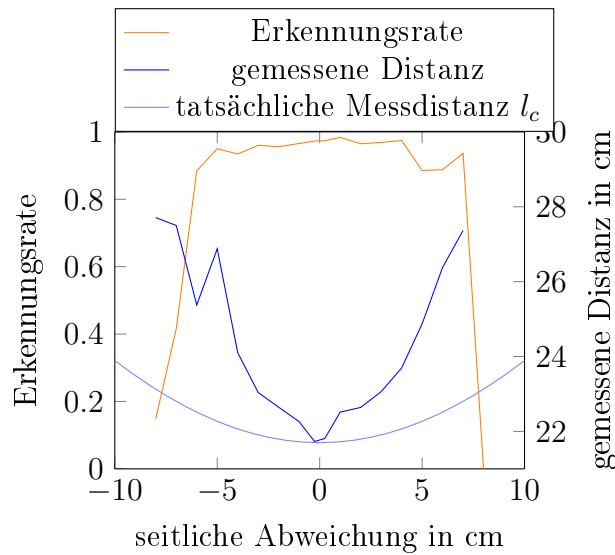


Abbildung 20: Messbereich der Ultraschallsensoren mit  $l_a = 22$  cm

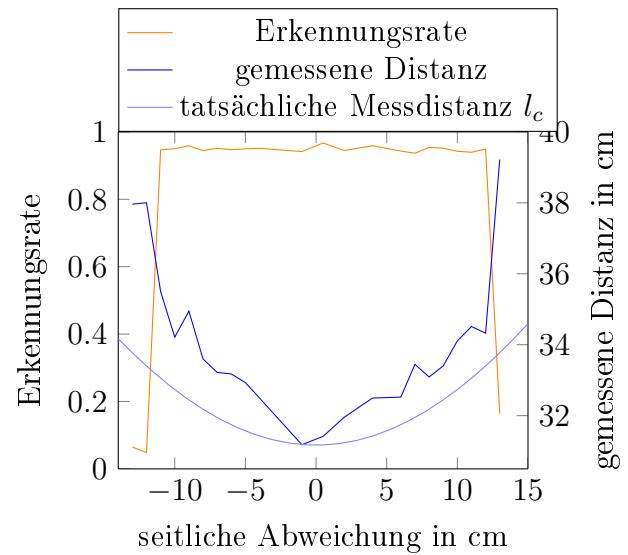


Abbildung 21: Messbereich der Ultraschallsensoren mit  $l_a = 31,2$  cm

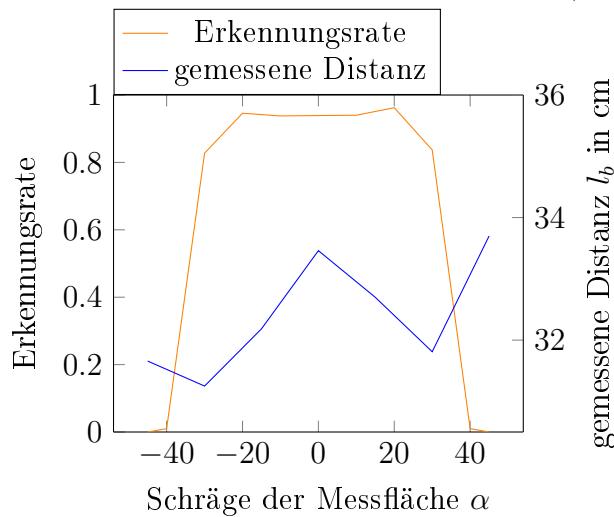


Abbildung 22: Genauigkeit der Messungen mit Ultraschallsensoren bei schrägen Reflexionsflächen