Analyse und Bereitstellung von positionsbasierten Mobilfunkmessungen zur trace-basierten V2X-Simulation

Masterarbeit

von

Adrian Skuballa

aus

Solingen

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

2015

Betreuer:
Norbert Goebel M. Sc.

Abstract

Die vorliegende Masterarbeit befasst sich mit der Aufarbeitung von positionsbasierten Mobilfunkmessungen zur trace-basierten V2X-Simulation. Um die Testung und Qualitätskontrolle von V2X-Applikationen, die in neueren und digitalisierten Fahrzeugen immer häufiger vorzufinden sind, kostengünstig und effizient zu ermöglichen, sind Simulationen erforderlich. Für die simulierten Mobilfunknetzwerke werden naturgetreue Netzwerkcharakteristiken benötigt, um die Beweiskraft und Güte der Resultate von Simulationen zu verifizieren. Im Rahmen dieser Masterarbeit wurden vier Anwendungen umgesetzt, die die vom Rate Measurement Framework (RMF) gemessene Netzwerkcharakteristiken, in mehreren Schritten analysieren und aufarbeiten, damit diese anschließend einem Simulator zur Verfügung gestellt werden können.

Eine zentrale Aufgabe besteht in der Verteilung der Messwerte des RMFs auf den OpenStreetMap (OSM) Graphen, da die Simulationen auf dem gleichen Kartenmaterial aufbauen. Für diese Umsetzung wird ein vorhandener Offline MapMatching Algorithmus angewendet, der für die Aufgabenstellung dieser Masterarbeit angepasst und verbessert wurde. Anschließend wird jeder Messwert durch mehrere zeit- und positionsbasierte Abbildungen und vorherige umfangreiche Umstrukturierungen der OSM Datei auf eine eindeutige Position einer Kante des Simulator Graphen positioniert.
Des Weiteren können die Messwerte vom Anwender auf verschiedene Weisen aggregiert werden, so dass hierdurch für den Anwender eine weitere Möglichkeit der Einflussnahme auf die Konfiguration der simulierten Mobilfunknetzwerke besteht. Hierdurch behalten die Messdaten und die Simulationen und ihre Güte, in dem die Messwerte nicht berechnet, geglättet oder verfälscht werden.

Die Ergebnisse der vorliegenden Abschlussarbeit leisten einen Beitrag zur Aussagekraft und Qualität des am Lehrstuhl für Rechnernetze und Kommunikationssysteme entworfenen Simulators für V2X-Applikationen.

Danksagung

An dieser Stelle möchte ich die Gelegenheit nutzen, um denen, die mir bei der Erstellung meiner Masterarbeit zur Seite standen, meinen besten Dank auszusprechen.

Zunächst danke ich Prof. Dr. Martin Mauve für die Herstellung der Rahmenbedingen, die die vorliegende Arbeit erst möglich machten.

Weiterhin möchte ich mich herzlich bei meinem Betreuer Norbert Goebel und meinem Leidensgenossen Raphael Bialon für die vielen und regelmäßigen Diskussionen sowie hilfreichen Anregungen und Tipps bedanken. Besonders hervorheben möchte ich, dass Herr Goebel viele Messfahrten durchgeführt hat, deren Messwerte die erforderliche, praxisnahe Umsetzung der entwickelten Tools möglich machten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kapitel 1.

Einleitung

1.1. Motivation

Alle namhaften und großen Autohersteller, Zulieferer und IT Unternehmen der Welt forschen aktuell im Bereich der Digitalisierung von Fahrzeugen. Die Computer in den Fahrzeugen und der Infrastruktur sollen dem Fahrer zunehmend mehr Aufgaben abnehmen und zugleich die Fahrsicherheit erhöhen. So hat zum Beispiel die Daimlar AG auf der Internationalen Automobil-Ausstellung (IAA) 2014 für Nutzfahrzeuge in Las Vegas den *Future Truck 2025* - einen selbstfahrenden Lastkraftwagen - der Öffentlichkeit vorgestellt ([au14], [ne14]). Für eine optimale Funktionsvielfalt und maximale Sicherheit genügt es jedoch nicht die Sensorarten und ihre Anzahl zu erhöhen und im Vorfeld die Verkehrsfahrzeuge mit intelligenterer Software auszustatten. Entscheidend ist, wie ein Fahrzeug während der Fahrt - also wenn es sich bereits im Straßenverkehr befindet - die ihm zur Verfügung stehenden Informationen optimal nutzen kann, um so adaptiv auf die sich verändernde Realität zu reagieren. Hier stellt die Kommunikation der Fahrzeuge untereinander sowie mit der Infrastruktur (zum Beispiel mit Straßen, Ampeln, Rechenzentren) einen relevanten Faktor dar. Diese Kommunikation kann unter anderem über W-Lan oder Mobilfunk umgesetzt werden.
Das Entwickeln, Testen und Kontrollieren dieser digitalen Fahrzeuge und ihrer Systeme stellt Forschung und Industrie vor große Herausforderungen. Bislang wurden Effektivität und Effizienz dieser Fahrzeuge vorwiegend auf speziellen und abgesperrten Ver-

kehrstestgebieten untersucht. Im Sommer 2015 berichteten mehrere Medien, dass Apple Inc. einen Antrag auf den Zugang zum *GoMentum Station* eingereicht hat ([he15]). Das GoMentum Station ist ein 8,5 km^2 großes Verkehrstestgebiet in Concord (Kalifornien, USA), auf dem bereits Google Inc. seine selbstfahrenden Fahrzeuge testet. Daimler hingegen hat erst im Mai 2015 eine Genehmigung für den US-Bundesstaat Nevada erhalten, um zwei seiner Future Trucks auf öffentlichen Straßen zu erproben ([sp15], [ze15]). Der Automobilzulieferer Delphi Automotive bestätigte im August 2015, dass von 2016 an auf der Landstraße L418 bei Wuppertal Tests von selbstfahrenden Fahrzeugen unter deutschen Alltagsbedingungen durchgeführt werden dürfen ([wd15]).
Bei all diesen aufgelisteten Beispielen fahren die Testfahrzeuge nicht eigenständig, sondern werden durchgängig von Testfahrern und Ingenieuren überwacht, die bei Softwarefehlern und/oder Notfallsituationen jederzeit eingreifen können. Das Testen der Kommunikation der Fahrzeuge untereinander ist komplexer und aufwendiger, da mehrere bis viele Testfahrzeuge benötigt werden und sich die Kommunikationseigenschaften (Netzwerkcharakteristiken) durch die Anzahl der Teilnehmer ändert. Hier wäre eine realistische Simulationslösung von großem Vorteil. Eine solche Lösung wurde am Lehrstuhl für Rechnernetze und Kommunikation an der Universität Düsseldorf im Rahmen einer vorangegangenen Masterarbeit untersucht und umgesetzt ([bi15]). Diese Arbeit vereinigte mehrere bekannte und bewerte Simulationskomponenten zu einer Lösung, so dass die V2X-Appliaktionen per Mobilfunkkommunikation auf Basis von Traces sehr detailliert simuliert werden konnten. Damit diese Simulationen möglichst realitätsnah abgewickelt werden können, werden echte, unter natürlichen Bedingungen beobachtete Netzwerkcharakteristiken benötigt. Diese positionsbasierten Messwerte werden bei Messfahrten mit einem Fahrzeug auf Verkehrsstraßen von der Messsoftware RMF erfasst und protokolliert.
In der vorliegenden Masterarbeit werden die protokollierten Messwerte des RMF, die während einer Messfahrt entstanden sind, soweit aufgearbeitet, dass diese dem Simulator als Grundlage für die realitätsnahe Simulation der Mobilfunkverbindung dienen. Um dies zu ermöglichen wurden im Rahmen der vorliegenden Masterarbeit mehrere Aufgaben und Probleme gelöst. Hierbei werden die vom Messfahrzeug gemessenen und protokollierten Netzwerkcharakteristiken auf den Straßenkartengraph der Simulation verteilt. Hierzu sind mehrere positions- und zeitbezogene Abbildungsschritte nötig. Damit diese Abbildungen möglich sind, müssen unter anderem der OpenStreetMap Straßengraph angepasst, ein mathematisch optimaler Kantenpfad im OpenStreetMap Straßengraph er-

mittelt und mehrere Abbildungsschritt der GPS Positionen durchgeführt werden. Als Resultat dieser Masterarbeit, können die V2X-Appliaktionen per Mobilfunkkommunikation mit realistischen Netzwerkcharakteristiken auf echten Straßenkarten erfolgreich simuliert werden. Folglich stellt diese Masterarbeit mit ihren Ergebnissen nicht nur einen Machbarkeitsnachweis eines Gedankenexperiments dar, sondern liefert einen Mehrwert, indem es einer praktischen Umsetzung und Validierung unterzogen wurde.

1.2. Aufbau der Arbeit

Kapitel 1 erläutert die Arbeiten, auf denen die vorliegende Abschlussarbeit aufbaut und betont die Notwendigkeit ihrer Ergebnisse für die praktische Anwendung.
In Kapitel 3 werden die Algorithmen sowie ihre Probleme und Lösungen, die für das Bereitstellen der gemessenen Netzwerkcharakteristiken für Simulationen umgesetzt wurden, vorgestellt.
Im Rahmen dieser Arbeit wurden vier Anwendungen entworfen und erweitert. Wichtige Aspekte der Implementierung werden in Kapitel 4 ausgeführt.
Eine schrittweise Anleitung der Anwendung der implementierten Programme ist in Kapitel 5 zu finden und soll dem Anwender an einem konkreten Beispiel den Einstieg in die Anwendung der Programme erleichtern.
Die Ergebnisse des MapMatchings werden in Kapitel 6 diskutiert.
Die Möglichkeiten der Daten-Aggregation werden in Kapitel 7 erklärt.
Zum Schluss wird in Kapitel 8 die vorliegende Masterarbeit rekapituliert. Darüber hinaus wird ein Ausblick in weitere Aufgaben und Themen gegeben.

Kapitel 2.

Verwandte Arbeiten

Die vorliegende Masterarbeit verfolgt das praktische Ziel, gemessene Netzwerkcharakteristiken von Mobilfunkverbindungen auf den Straßengraphen eines Simulators abzubilden.

Die Netzwerkmessungen werden mit dem *Rate Measurement Framework* (RMF) vorgenommen ([go14]). Dieses Framework wurde am Lehrstuhl für Rechnernetze und Kommunikation der Heinrich-Hein-Universität unter der Projektleitung von Norbert Goebel entwickelt (weitere Mitwirkende: Sebastian Wilken, Christian Lange, Franz Kary und Tobias Krauthoff). Es misst die Datenrate, Latenz und Paketverlustrate beider Verbindungsrichtungen zwischen einer stationären Serverkomponente und einem mobilen Client, der auf einem Messrechner (zum Beispiel Notebook) im Fahrzeug läuft. Der Server und Client des RMFs senden mit einer Frequenz von ca. 5 Hz eine Reihe von Datenpaketen an den anderen Teilnehmer. Dabei berücksichtigen beide Seiten die sich im Mobilfunk sehr schnell ändernden Netzwerkeigenschaften und passen ihr Sendeverhalten entsprechend an. Sogar das Problem, dass es zum *self-induced congestion* beim Mobilfunkteilnehmer kommen kann und durch zu vieles sowie schnelles Senden entsteht, findet durch das RMF Berücksichtigung ([go14]).

Eine der zentralen Aufgaben der vorliegenden Masterarbeit besteht darin, eine während einer Messfahrt erstellte GPS-Protokollierung, Modeminformationen zum Verbindungsstatus und die dazugehörigen Messwerte des RMFs auf einen Kantenpfad des OpenStreetMap Graphen abzubilden. Im Kontext dieser Masterarbeit wurde diese Aufgabe

mit dem Programm JXMapMatchVer3 gelöst. Bei dieser Lösung handelt es sich um eine Fortsetzung und Erweiterung des Programms JXMapMatchVer2, an dem Daniel Sathees Elmo und zuvor Tobias Amft gearbeitet haben (Herr Elmo und Herr Amft waren zur damaligen Zeit am Lehrstuhl für Rechnernetze und Kommunikation der Heinrich-Heine-Universität beschäftigt). Die Ausgangslage für den MapMatching Algorithmus ist das Paper *Efficient map-matching of large GPS data sets-Tests on a speed monitoring experiment in Zurich* (2014) von Fabrice Marchel und Kay W. Axhausen ([fk14]). Angaben zu wichtigen Änderungen zwischen JXMapMatchVer2 und JXMapMatchVer3 werden in Kapitel 4 (Implementierung) aufgeführt.

Die Ergebnisse dieser Masterarbeit sind die Verarbeitung, Abbildung und Aggregation der Messdaten des RMFs sowie der dazugehörigen Anpassungen des OpenStreetMap Graphen.
Diese Daten werden für anschließende Simulationen von V2X-Applikationen benötigt. Die Simulationssoftware wurde in der Masterarbeit von Raphael Bialon erarbeitet ([bi15]). Sie besteht aus mehreren bekannten und bewerten Simulationskomponenten, die zu einer Gesamtlösung vereint wurden. In einer Simulation werden auf einem Straßengraphen, der auf OpenStreetMap Daten aufbaut, fahrende Fahrzeuge, die V2X-Applikationen und die Mobilfunkverbindungen der Fahrzeuge detailliert simuliert. Während der Bearbeitung der beiden Masterarbeiten (die vorliegende Masterarbeit und die Arbeit von Herrn Bialon) fanden regelmäßige Besprechungen statt, bei denen technische Überschneidungen und Schnittstellen für die Datenübergabe abgestimmt wurden. Die Zusammenarbeit beider Masterarbeiten führte bereits während ihrer Bearbeitungsphase zu ersten erfolgreichen Simulationen, die in der Natur beobachteten Mobilfunkeigenschaften berücksichtigten.

Somit liefert die vorliegende Abschlussarbeit eine entscheidende Vorarbeit für das Simulieren von V2X-Applikationen im Kontext der Mobilfunkkommunikation.

Kapitel 3.

Funktionsweise und Algorithmen

Im Rahmen dieser Masterarbeit wurden mehrere Programme konzipiert und implementiert. Jedes Programm erhält eine oder mehrere Dateien als Eingabe, verarbeitet diese Daten und speichert seine Ergebnisse als Ausgabe in eine oder mehrere neue Dateien. Die Ausgabedateien dienen der a.) Datenanalyse und -kontrolle für den Anwender, b.) als Eingabe für das nächste Programm oder c.) beides. Es wäre auch möglich einige oder alle Programme in einem größeren Programm zu vereinen. Auch wenn dies theoretisch und praktisch möglich wäre, wurde dies aber nicht umgesetzt, um eine maximale Trennung von einzelnen Aufgaben und ihren Lösungen zu erlauben. Des Weiteren ist so auch ein manueller Eingriff in die zwischen Lösungen und Dateien möglich.

3.1. Ausgangslage und Ziel

Ziel dieser Masterarbeit ist es einer Simulationssoftware möglichst realitätsnahe Netzwerkcharakteristiken von Mobilfunkkommunikation zur Verfügung zu stellen. Ausgangslage dieser Masterarbeit waren verschiedene gemessene und protokollierte Netzwerkcharakteristiken. Diese Daten sind in drei Arten unterteilt:

- Mit Hilfe des RMF werden Netzwerkcharakteristiken und die dazugehörigen Zeitstempel protokolliert. Bei den Netzwerkcharakteristiken werden die Datenrate (*datarate*), die Latenz (*delay*) und die Verlustrate (*loss_rate*) gemessen. Im Folgendem werden diese drei Werte, die zu einem genauen Zeitpunkt (Zeitstempel) gemessen wurden, **Messwert** genannt.

- Während einer Messfahrt werden auch die Verbindungseigenschaften des Mobilfunkmodems abgefragt und zusammen mit Zeitstempeln gespeichert. In diesen Daten sind Informationen zu der Verbindungsart (*2G* oder *3G*) und der verbundenen Basisstation enthalten. Diese Informationen werden **Modemdaten** genannt.

- Des Weiteren wird während einer Messfahrt die GPS Position inklusive ihres Zeitstempels im Sekundentakt gespeichert.

Die Messwerte sollen aufgearbeitet und dann der Simulationssoftware zur Verfügung gestellt werden, so dass am Ende die Simulationssoftware mit Hilfe der positionsbasierten Netzwerkcharakteristiken die Kommunikation von V2X-Anwendungen über Mobilfunknetze möglichst realitätsnah simulieren kann. Die Simulationssoftware besteht aus mehreren Komponenten und Modulen:

- Simulation of Urban Mobility (SUMO)

- VSimRTI_App (V2X-App Simulator)

- OMNeT++

  - Trace Based UMTS Simulation (TBUS) innerhalb von OMNeT++

Wie diese Komponenten genau zusammenarbeiten und benutzt werden, wird in der Masterarbeit von Raphael Bialon (*Coupled simulation of road traffic, V2X-applications and V2X-communication via mobile cellular networks*) dargestellt. Nachfolgend werden diese Simulationssoftware und alle ihre Komponenten **Simulator** genannt.

Während einer Messfahrt werden die Eigenschaften der Mobilfunkdatenverbindung sowie die Position gemessen und protokolliert. Das Ergebnis wird in vier Dateien ab-

gespeichert. Alle vier Dateien sind vom Inhalt Textdateien (*gpsd.log* und *cellinfo.txt*) oder Character-separated values (*downstream-data.csv* und *upstream-data.csv*) und somit auch menschenlesbar. An dieser Stelle werden die vier Dateien und ihr Inhalt genauer betrachtet:

- **gpsd.log**
  In dieser **GPS-Log-Datei** sind die genauen Ort- und Zeitangaben des Messfahrzeugs im Sekundentakt gespeichert. Hierbei handelt es sich primär um die Ausgabe des Linux *gpsd* Dienstes beziehungsweise des *gpspipe*. Am Anfang jeder Zeile wird zusätzlich ein Zeitstempel hinzugefügt. Anschließend folgt die gpspipe JSON Ausgabe. In den JSON Objekten der Klasse *TPV* ist die *Global Positioning System* (GPS) Ortsangabe in Längen- und Breitengraden angegeben. Die Liste der GPS Positionen inklusive ihrer Zeitstempel wird auch **GPS-Trace** genannt.

- **cellinfo.txt**
  In dieser Datei sind technische Statusinformationen des verwendeten Mobilfunkmodems hinterlegt. Diese Informationen wurden mit Hilfe von AT Kommandos aus dem Modem abgefragt und zusammen mit einem Zeitstempel gespeichert. AT Kommandos erlauben die Auswahl verschiedenster Modemeinstellungen und das Auslesen von Informationen. Im Rahmen dieser Masterarbeit sind nur die Verbindungsart (2G oder 3G) und die Identifikation der verbundenen Zelle relevant. Um die Identifikation von Zellen zu ermöglichen, werden bei einer 2G Verbindung der *Location Area Code* und die *CellID* abgefragt und abgespeichert; bei einer 3G Verbindung werden der *Channel* und der *Primary Scramblingcode* abgefragt und abgespeichert.

- **downstream-data.csv** und **upstream-data.csv**
  In diesen zwei Dateien werden die eigentlichen gemessenen Netzwerkcharakteristiken protokolliert. Die drei wichtigsten Spalten und Werte (Messwert) lauten:

  loss rate, **data rate [Byte/s]** und **delay [s]**

  Eine der wichtigsten Aufgaben dieser Masterarbeit ist es diese gemessenen Werte auf eine eindeutige und genaue Position einer Straßen-Kante des Simulators präzise zu positionieren. Auch in diesen beiden Dateien existiert zu jedem Datensatz

ein Zeitstempel. Für jede Verbindungsrichtung (*downstream* und *upstream*) wurden die Messwerte in einer separaten Datei abgelegt.
Die Mobilfunkmesssoftware (RMF) wurde und wird von Norbert Goebel, seinen Mitarbeitern und Studenten entwickelt und betreut.

3.1.1. OpenStreetMap

OpenStreetMap (OSM) ist ein öffentliches Projekt, bei dem Kartenmaterial nach dem OpenSource Prinzip entsteht. Jede Person darf neues Kartenmaterial einreichen, altes Kartenmaterial aktualisieren oder verbessern. Benutzer können die OSM Karten im Browser abrufen, anschauen oder als Dateien herunterladen. Eine OSM Datei enthält das Kartenmaterial im *Extensible Markup Language* (XML) Format. Um eine Straße in OSM zu beschreiben sind nodes und ways nötig.

- **node**
  Ein node wird in ein XML-Tag mit mehreren Attributen angegeben. Die drei wichtigsten Attribute sind die *id*, *lat* und *lon*. Mit der id wird die node identifiziert. lat und lon ist die GPS Position in Breiten- und Längengraden.

- **way**
  way ist ebenfalls ein XML Tag mit mehreren Attributen. Das wichtigste Attribut ist das *id* Attribut für die Identifizierung. Der way hat aber mehrere Kinder-Tags, in denen alle anderen wichtigen Informationen gespeichert werden. So geben diese Kinder-Tags und ihre Attribute an, ob dieser way eine Straße ist und welche Eigenschaften diese hat (z.B. Geschwindigkeitsbegrenzung, Fahrspurenanzahl, Fahrtrichtung). Ein way beinhaltet aber auch zwei oder mehrere *nd*-Tags. Jeder nd-Tag hat ein *ref*-Attribut, das die node id als Wert enthält. Diese Liste von nodes mit ihren GPS Positionen bilden den Verlauf der Straße nach.

Die Verbindung zwischen zwei nodes wird nachfolgend als **wayPart** bezeichnet. Wenn ein way n viele nd-Tags beinhaltet, so besteht dieser way entsprechend aus n-1 vielen wayParts.

Das Listing 3.1 zeigt ein Beispiel mit zwei nodes und einem way Tag.

Listing 3.1: Beispiel OpenSteetMap XML Tag

```
1  <node id="2409285517" lat="51.1622775" lon="6.8725373" version="1" timestamp="2013-08-06T21:13:27Z" changeset
       ="17247138" uid="128720" user="EinKonstanzer"/>
2  <node id="296343732" lat="51.1622536" lon="6.8726286" version="6" timestamp="2015-05-06T15:07:00Z" changeset
       ="30845320" uid="24644" user="Athemis"/>
3
4  ...
5
6          <way id="1011096" version="20" timestamp="2015-05-06T15:21:40Z" changeset="30845785" uid="24644" user
               ="Athemis">
7                  <nd ref="2409285517"/>
8                  <nd ref="296343732"/>
9                  <tag k="highway" v="secondary"/>
10                 <tag k="lanes" v="2"/>
11                 <tag k="lit" v="yes"/>
12                 <tag k="lit_by_gaslight" v="no"/>
13                 <tag k="maxspeed" v="50"/>
14                 <tag k="name" v="Benrather Schlossallee"/>
15                 <tag k="postal_code" v="40597"/>
16         </way>
```

3.1.2. Simulator Graph

Der Simulator erstellt aus einem OSM Graphen einen eigenen Straßengraphen. Dabei übernimmt der Simulator nicht alle Informationen eins zu eins aus der OSM Datei. Nach mehreren internen Aufarbeitungs- und Optimierungsschritten erstellt der Simulator aus einem OSM way mit Hilfe des Tools *netconvert* einen eigenen Graphen. Dieser Graph beinhaltet nun keine ways mehr, sondern **edges** mit **lanes**. Diese edges, lanes und weitere Informationen speichert der Simulator je nach Simulationskonfiguration in einer internen Datenbank und / oder in einer XML Ausgabedatei. Das Listing 3.2 zeigt ein Beispiel einer edge aus der XML Ausgabe. Diese edge ist aus dem way von Listing 3.1 entstanden.

Listing 3.2: Beispiel edge des Simulators

```
1      <edge id="1011096_2409285517_296343732_2409285517" from="2409285517" to="296343732" priority="-1">
2          <lane id="1011096_2409285517_296343732_2409285517_0" index="0" speed="13.89" length="6.96" shape
               ="4590.87,7458.56 4597.25,7455.68"/>
3          <lane id="1011096_2409285517_296343732_2409285517_1" index="1" speed="13.89" length="6.96" shape
               ="4592.23,7461.57 4598.54,7458.72"/>
4      </edge>
```

3.2. Aufgaben, Probleme, Lösungen und Grundkonzeption

Um die drei wichtigen gemessenen Netzwerkcharakteristiken (datarate, delay und loss_rate) aus den csv-Dateien dem Simulator zur Verfügung zu stellen, sind mehrere Abbildungsschritte nötig, da ein Messwert zwar mit einem Zeitstempel versehen ist, aber keine GPS Position enthält. Eine GPS Position hat nur der GPS-Datensatz. Ein weiteres Problem besteht darin, dass die GPS Position zum Teil nicht auf den Zentimeter oder Meter genau erfasst wird. Dieses *GPS-Rauschen* ist immer vorhanden und wird besonders bei den gemessenen GPS Positionen von einem stehenden Fahrzeug evident. Hier kann sich die GPS Position je nach Verbindungsqualität des GPS Empfängers um Zentimeter bis hin zu einigen Metern verändern. Als weiteres GPS Problem kommt hinzu, dass die OSM Daten von der Wirklichkeit zum Teil um mehrere Meter abweichen können, weil beispielsweise eine vierspurige Straße in OSM als ein way gespeichert sein kann. Laut OSM haben dann alle vier Fahrspuren dieselben GPS Positionen. Das Abbilden der während einer Messfahrt aufgezeichneten GPS Positionen auf die passenden ways des OSM Graphen wird **MapMatching** genannt.
Die gemessenen und auf den OSM Graphen abgebildeten GPS Punkte müssen anschließend auf den Graphen des Simulators abgebildet werden, da während der Simulation die originalen OSM ways und ids nicht mehr vorhanden sind.
Die Abbildung von OSM Graph auf den netconvert/Simulator Graph ist surjektiv. Durch diese surjektive - nicht bijektive - Abbildung vom OSM Graphen zum Simulator Graphen entsteht ein neues und ernstes Problem, denn das ursprüngliche Ziel war es die ermittelten Messwerte aus der Natur möglichst präzise und ohne Verluste beziehungsweise nur mit minimalen Änderungen als Basis für die Simulation zu verwenden.

Für die diversen Abbildungen wurden mehrere Programme entworfen und umgesetzt. Ihre Aufgaben und die mit ihnen umgesetzten Lösungen werden in den folgenden Unterkapiteln beschrieben.
Der Ablauf der vier Programme, des Simulators (SUMO) sowie der Ein- und Ausgabedateien ist in Abbildung 3.1 dargestellt.

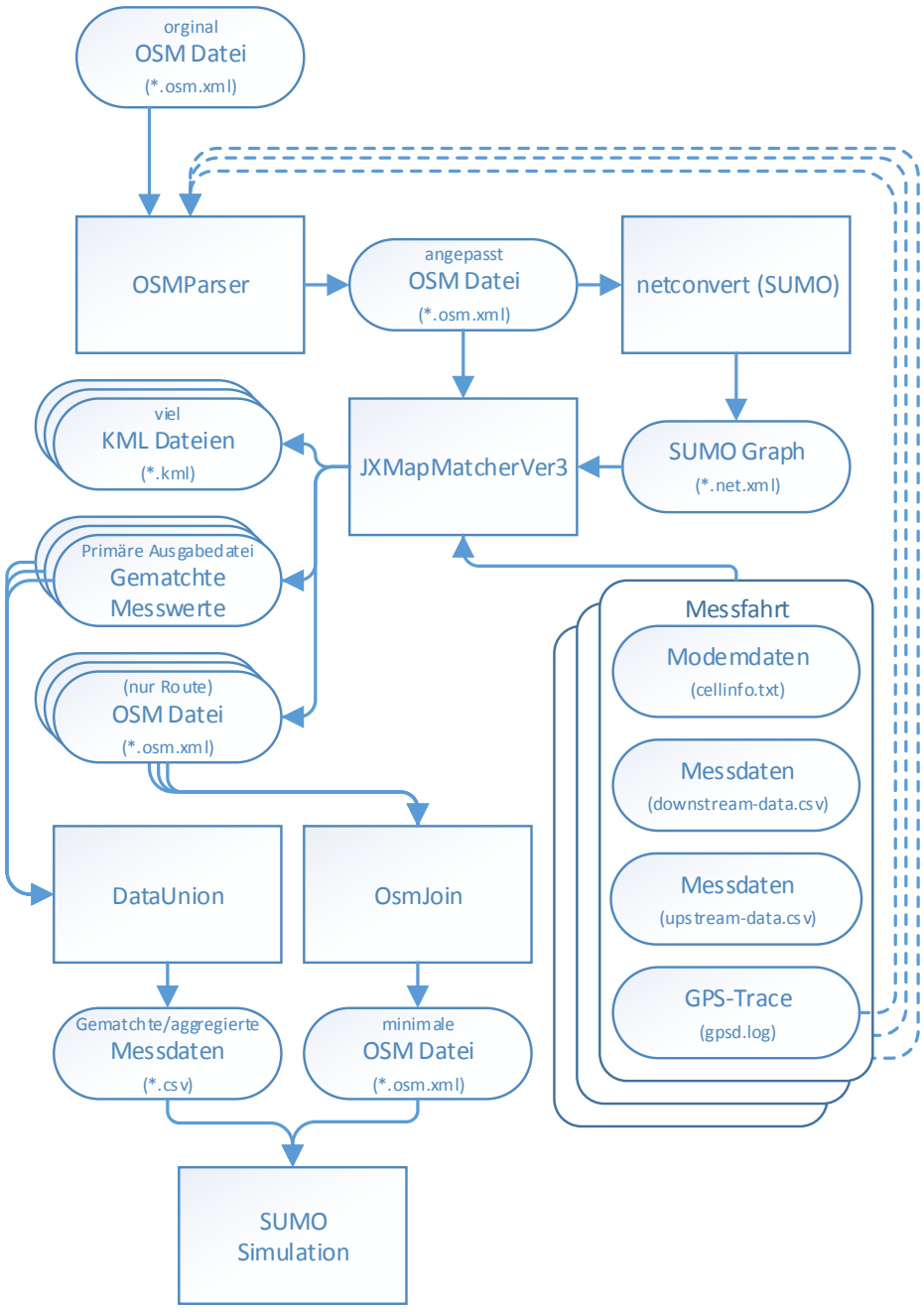

Abbildung 3.1.: Flussdiagramm: Programme mit Ein-/Ausgabe

3.2.1. OsmParser

Der OsmParser liest eine OSM Datei sowie keine, eine oder mehrere GPS-Log-Dateien ein und erstellt eine neue OSM Datei. Der OsmParser hat mehrere Aufgaben:

- Der OSMParser ermittelt die minimalen und maximalen GPS Positionen der GPS-Log-Dateien, so dass die neu erstelle OSM Karte nur Informationen in diesem Gebiet beinhaltet. Dies soll die OSM Datei verkleinern, um die Ladezeiten dieser Datei für die nächsten Programme zu reduzieren und Timeouts zu vermeiden. Um an den Rändern der Karte keine wichtigen Straßen abzuschneiden und somit wichtige Informationen zu verlieren, werden die ermittelten minimalen und maximalen GPS Positionen um einen parametrisierten Offset vergrößert.
  Die Position, Größe und Ränder der neuen OSM Karte können nicht nur durch die GPS-Log-Dateien bestimmen werden, sondern auch durch vom Anwender definierte GPS Koordinaten. Hierfür können dem OsmParser zwei GPS Positionen als Parameter übergeben werden, die zwei gegenüberliegende Eckpunkte in einem Rechteck bilden.
  Falls der OsmParser keine der beiden Informationen erhält, wird die Größe des Gebietes, das der OSM Graphen abdeckt, nicht verkleinert.

- Des Weiteren werden nur solche Karteninformationen übernommen, die für den Autoverkehr und somit für die Simulationen relevant sind. Diese Eingrenzung ist wichtig, da in den OSM Dateien sehr viele Informationen existieren, die für den Autoverkehr nicht von Bedeutung sind (zum Beispiel Gebäude, Parks oder Stromleitungen). Auch dieser Schritt verkleinert die ausgegebene OSM Datei häufig um deutlich über 50 Prozent.

- Der Graph des Simulators ist optimiert und somit nicht mehr identisch mit dem OSM Graphen. Nodes mit identischen GPS Koordinaten werden zusammengefasst. Dadurch entstehen neue Straßenverbindungen (edges/ways) und neue nodes. Dieses Verhalten ist laut OSM Definition nicht vorgesehen. Diese neuen ways und nodes gibt es im OSM Graph nicht. Somit ist eine bijektive Abbildung zwischen diesen zwei Kantenmengen der Graphen nicht gegeben. Eine Abbildung der Messwerte auf die ways und im nächsten Schritt auf die edges wäre deutlich komplexer.

Um dieses Problem zu lösen, prüft der OsmParser, ob alle erforderlichen nodes eindeutige GPS Koordinaten haben. Falls zwei nodes identische GPS Koordinaten haben, wird der Breitengrad des zweiten nodes um 0.00000001 Grad verschoben. Dies entspricht einer Verschiebung um ca. einen Millimeter und führt nur direkt am Nordpol zu einem Fehler (Breitengrad über 90°).

- Im OSM Graph kann es vorkommen, dass eine Straßenverbindung zwischen zwei nodes in mehreren ways vorkommt und somit mehrfach vorhanden ist. Diese Verbindungen werden im netconvert Graph zusammengefasst, so dass neue ways beziehungsweise edges entstehen, wodurch wiederum eine bijektive Abbildung verhindert wird. Der OsmParser filtert deshalb alle doppelten Straßenverbindungen heraus.

- Ein OSM way besteht aus einem Pfad, der durch zwei oder mehr nodes definiert wird. Wenn ein node nur in einem way benutzt wird und somit keine Verbindung zu einem anderem way bildet, optimiert der Simulator (netconvert) diesen node weg. Da die edges des Simulators eine Längenangabe in Metern haben, wäre hier eine bijektive Abbildung zwischen einem OSM way beziehungsweise einem Kantenpfad und einer netconvert edge möglich, wenn die Länge bei der Abbildung Berücksichtigung findet. Jedoch wurde festgestellt, dass die Längenangaben des netconverts nicht in allen edges der Wirklichkeit entsprechen. Diese Problematik äußert sich darin, dass ein Punkt auf der optimierten edge eins zu eins auf den ursprünglichen OSM way abgebildet werden kann, da der OSM way aus mehreren wayParts besteht. Diese Kanten wurden in der edge zusammengefasst. Durch die fehlerhafte Längenberechnung einer edge ist eine Rückabbildung nicht mehr möglich. Dieses Problem wurde gelöst, indem jeder way in der OSM Ausgabedatei des OsmParsers nur aus zwei nodes besteht. Auf diese Weise fasst der Simulator wayParts nicht zusammen, wodurch wiederum eine lineare und bijektive Abbildung von einem way auf eine edge ermöglich wird.
  Zu beachten ist, dass die Begriffe way und wayPart im Folgenden für die OSM Ausgabedatei des OsmParsers **gleichbedeutend** sind, weil in der OSM Ausgabedatei und somit auch in allen folgenden Anwendungen (JXMapMatcher, DataUnion, OsmJoin) und Algorithmen (N Route MapMatching) jeder way nur einen wayPart hat.

Beispiel: Ein OSM way besteht aus n nodes und somit aus n-1 wayParts. Dieser OSM way wird mit allen seinen Eigenschaften n-1 mal vervielfacht, so dass in der Ausgabe des OsmParsers n-1 viele OSM ways resultieren. Natürlich erhalten alle ways neue eindeutige ids. Die alte id wird aber in einem neu erstellten Kinder-Tag abgespeichert (<*tag k="old_way_id" v="197461114"/*>).

- In einem way-Tag wird die Fahrtrichtung in einem Unter-Tag gespeichert. Wenn diese Angabe nicht vorhanden ist, sind laut OSM Definition beide Richtungen befahrbar. Jedoch verfährt der Simulator in seltenen Fällen nicht konform zu dieser Regel. Deshalb fügt der OsmParser die Information der Fahrtrichtung, falls nicht vorhanden, zu jedem way hinzu.

3.2.2. N Route Algorithmus - die optimale OSM Route

In diesem Kapitel wird beschrieben, wie das Offline-MapMatching der GPS-Datensätze auf die OSM wayParts funktioniert, um die optimale Route im OSM Graph zu finden. Diese Route soll den tatsächlich gefahrenen Weg im OSM Graphen nachbilden.

Hierfür benötigt JXMapMatchVer3 die Liste mit den GPS Positionen (GPS-Trace) und den OSM Graphen (OSM Ausgabedatei des OsmParsers). Der OSM Graph besteht aus ways. Ein way hat nur einen wayPart und ein wayPart besteht aus zwei GPS Positionen (nodes). Ein normaler OSM way kann auch aus mehreren wayParts bestehen. Da ein way des OsmParsers immer genau nur einen wayPart hat, sind diese Begriffe in dem Ausgabe OSM Graph des OsmParsers und somit dem Eingabe OSM Graph des JXMapMatchVer3 gleichbedeutend. Straßenverbindungen und Kreuzungen werden in einem OSM Graphen nachgebildet, indem ein node von mehreren ways genutzt wird.

Der MapMatching Algorithmus sucht die optimale Route im OSM Graphen, die den minimalen Abstand zu allen GPS Punkten aufweist.
Am Anfang wird die erste GPS Position betrachtet und die zu ihr n vielen am nächsten liegenden wayPart gesucht. Jeder dieser wayParts bildet eine Route, die aus einem Abschnitt (wayPart) besteht. Jede Route hat einen *Entfernungswert*, in dem die Entfernung vom GPS Punkt zu dem gematchten wayPart gespeichert wird. Sobald ein GPS Punkt auf

einen wayPart einer Route gematcht wird, wird diese Entfernung zwischen GPS Punkt und wayPart zu dem *Entfernungswert* der Route hinzuaddiert.
Im nächsten Schritt wird der nächste (zweite) GPS Punkt benutzt. Jede der n vielen Routen wird entsprechend ihrer Häufigkeit an ausgehenden wayParts kopiert. Jede dieser Kopien erhält einen ausgehenden wayPart als nächsten (aktuellsten) Routenabschnitt. Sodann wird der Abstand vom neuen GPS Punkt zu den aktuellsten wayParts aller Routen berechnet. Falls der Abstand von einer neuen *Kopie-Route* kleiner ist, als der der Ursprungsroute, so werden von dieser neuen Kopie-Route erneut alle ausgehenden wayParts untersucht. Dieser Vorgang wird so oft rekursiv wiederholt bis die Entfernung des aktuellen GPS Punktes zu allen ausgehenden wayParts größer ist als die Entfernung zum vorherigen wayPart. Diese Rekursion ist erforderlich, wenn es einen oder mehrere wayParts auf der Route gibt, auf die kein GPS Punkt gematcht wird.
Als nächstes werden die n vielen Routen gesucht, die den kleinsten *Entfernungswert* haben. Wenn zwei Routen den gleichen *Entfernungswert* aufweisen, dann werden ihre unterschiedlichen aktuellsten wayParts zur Bestimmung herangezogen. Falls auch die aktuellsten wayParts gleich sind, wird nur die kürzeste Route weiter verwendet. Alle anderen Routen werden nicht weiter betrachtet.
Abbildung 3.2 zeigt zwei Routen mit gleichem Entfernungswert, gleichen aktuellsten wayParts aber unterschiedlichen Längen. In dem dargestellten Beispiel wird *Route 2* nicht weiter berücksichtigt.

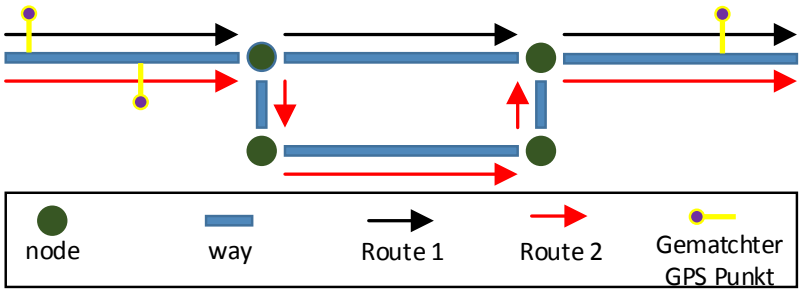

Abbildung 3.2.: Zwei Routen, gleiche Entfernungswerte, gleiche aktuellste wayParts, unterschiedliche Längen

Im nächsten Schritt wird der nächste – also der dritte – GPS Punkt betrachtet und alle n vielen Routen sowie ihre ausgehenden wayParts werden auf diesen GPS Punkt hin untersucht. Dieses Vorgehen wiederholt sich bis alle GPS Punkte betrachtet wurden.

Bei diesem Algorithmus werden immer n viele Routen und alle ihre nächsten möglichen Ausgangsrichtungen (wayParts) betrachtet und untersucht. Das n kann in der Benutzeroberfläche des JXMapMatchVer3 eingestellt werden. Während der Entwicklung und des Testens hat sich ein n der Größe 10 als effektiv und performant erwiesen. Wenn das n zu klein gewählt wird, kann es passieren, dass der N Route Algorithmus nicht die optimale Route findet. Falls die GPS Punkte an einer Kreuzung kurzzeitig eine andere Straße (wayPart) bevorzugen, wird die richtige Ausgangsrichtung gegebenenfalls verworfen. Wenn das n aber zu groß gewählt wird, steigt der Rechenaufwand mit einer Laufzeit von *n * log(n)* an. Dies ist auf den Umstand zurückzuführen, dass n viele Routen und alle ihre ausgehenden nächsten wayParts (inkl. Rekursion) geprüft werden. Des Weiteren werden dann auch noch alle diese Routen sortiert und bei gleichen *Entfernungswerten* im Detail verglichen.
Daher empfiehlt es sich mit einem n der Größe 10 anzufangen und diesen, wenn nötig, bei einem nächsten Durchlauf des N Route Algorithmus um den Faktor 1,5 zu vergrößern. Auf der Benutzeroberfläche des JXMapMatchVer3 ist nach einem Durchlauf schnell erkennbar, ob die gefundene optimale Route den tatsächlich gefahrenen Weg abbildet.

Wenn JXMapMatchVer3 eine Route gefunden hat und diese nur minimale Fehler beinhaltet, können diese anschließend manuell verbessert werden. Mit der rechten Maustaste kann ein wayParts aus der Route entfernt werden und mit der linken Maustaste kann ein wayPart an ein Ende der Route hinzugefügt werden. Beim Entfernen ist zu beachten, dass es nur einmal erlaubt ist eine Route zu *zerschneiden*, so dass *eine* Route maximal aus zwei Teilen bestehen darf. Erst wenn diese zwei Teile durch das Hinzufügen von neuen wayParts (linke Maustaste) vereinigt wurde, kann an einer anderen Stelle die Route *zerschnitten* werden.

Der N Route Algorithmus untersucht bei jedem Schritt nicht alle möglichen Routen, sondern nur die n besten. Falls unter den n besten nächsten Wegen die richtige Route nicht dabei ist, weil zum Beispiel die GPS Positionen zu ungenau oder das OSM Kartenmaterial die tatsächliche gefahrene Route nicht enthält, kann es passieren, dass alle n Routen des N Route Algorithmus in eine Sackgasse laufen und gegebenenfalls nicht mehr herausfinden. Dieses Problem kann umgangen werden, wenn nicht nach jedem Schritt nur die n besten Routen, sondern alle Route weiter untersucht werden. Dieses Vorgehen zieht aber den Nachteil einer exponentiellen Laufzeit nach sich, da sich bei jeder Abzweigung

und Kreuzung die Routenanzahl um die Anzahl der ausgehenden wayParts als Faktor erhöht.

3.2.3. JXMapMatchVer3

Mit dem JXMapMatchVer3 werden die Messwerte auf die edges des Simulators übertragen. Dieses Programm benötigt sechs Eingabedateien und erstellt viele neue Ausgabedateien. Folgend werden die wichtigsten Ein- und Ausgabedateien sowie ihre Funktionen erläutert. Ausführlichere Informationen zu sekundären Ein- und Ausgabedateien sind in Kapitel 5 (Anwendung) aufgeführt.

Eingabedateien:

- OSM Graph (*.osm.xml)
  Bei dieser Datei handelt es sich um die OSM Ausgabedatei des OsmParsers. (siehe Kapitel 3.2.1 - OsmParser)

- Simulator / netconvert Graph (*.net.xml)
  Dies ist die XML Datei, die den Graphen des Simulators enthält. netconvert ist eine Simulatorkomponente und speichert in dieser Datei seinen eigenen Graphen mit edges, den es aus der OSM Ausgabedatei des OsmParsers erstellt. Die Dateien OSM Graph (*.osm.xml) und netconvert Graph (*.net.xml) bilden ein festes Paar, da der netconvert Graph aus dem OSM Graph entsteht.

- GPS-Log-Datei (*.log)
  In dieser Datei werden die GPS-Datensätze des Fahrzeugs während einer Messfahrt gespeichert.

- Modem- und Verbindungsinformationen (cellinfo.txt)
  In dieser Datei sind technische Status- und Verbindungsinformationen des Mobilfunkmodems gespeichert, mit deren Hilfe Mobilfunkzellen identifizieren werden können.

- Netzwerkcharakteristiken (downstream-data.csv / upstream-data.csv)
  In diesen zwei Dateien sind pro Verbindungsrichtung (Up- und Downstream) die Messwerte, ihre Zeitstempel und weitere Protokollwerte des RMF gespeichert.

Ausgabedateien:

- **Primäre Ausgabedatei (*.csv)**
  Bei dieser Datei handelt es sich um eine csv-Tabellen-Datei. Die wichtigsten Spalten sind die drei gemessenen Messwerte (datarate, delay und loss_rate), edge_id_str, length_in_edge down_up und type. Die Spalte *edge_id_str* gibt die edge-Bezeichnung an, wie sie während der Simulation benutzt wird. Die Spalte *length_in_edge* gibt die genaue Position auf einer edge an (*Längenposition*). *down_up* gibt die Verbindungsrichtung der Messwerte an: *down* für das Empfangen und *up* für das Senden. Die Spalte *type* gibt folgende Informationen an:

  - **Real:** Der Datensatz und seine GPS Position wurden direkt auf die edge abgebildet.

  - **Start:** Falls es für diese edge keinen Messwert gibt der genau auf den Startpunkt der edge abgebildet werden kann (*length_in_edge = 0*), wird der letzte Messwert der vorherigen abgefahrenen edge übernommen. Der Wert für length_in_edge ist dann negativ.

  - **End:** Dieser Wert ist mit Start vergleichbar, nur setzt er - falls nötig - einen Messwert an das Ende der edge. Hierfür benutzt er den ersten Messwert der nächsten edge.

  - Die meisten Straßen sind in beide Richtungen befahrbar. Der Simulator hat in seinem Graph für jede Richtung eine eigene edge. Damit die Messwerte für beide Richtungen, also für beide edges, benutzt werden können, gibt es für die Spalte type auch die Werte **BackDirektion**, **StartBackDirektion** und **EndBackDirektion**. Hierbei handelt es sich um die Pendants zu Real, Start und End.

- ***.kml Dateien**
  Der JXMapMatchVer3 erstellt und speichert auch viele *Keyhole Markup Language* (KML) Dateien ab. In KML Dateien werden Geodaten und -informationen im XML Format gespeichert. Die Unterstützung des KML Formates von *Google Earth* hat die Bekanntheit von KML sehr verbreitet. Mit ihnen ist es möglich auf den Google Earth Karten weitere eigene Punkte, Linien und Abbildungen visuell darzustellen.
  Diese Möglichkeit wird auch hier genutzt, um verschiedene MapMatching- und Abbildungsschritte zu visualisieren.

  Erst diese visuelle Darstellung der KML Dateien in Google Earth ermöglicht es einen schnellen Überblick zu gewinnen und zu bestimmen, wie erfolgreich bestimmte MapMatching- und Abbildungsschritte durchgeführt wurden. So hat erst diese grafische Kontrolle der Daten gezeigt, dass die ursprüngliche Version (JXMapMatchVer2) und viele Zwischenversionen zum Teil sehr ungenau die gemessenen GPS Positionen auf die ways gematcht haben. In einigen Einzelfälle konnte sogar eine Verschlechterung beobachtet werden: Der gematchte GPS Punkt war weiter von dem way entfernt, als der ungematchte GPS Punkt. Die Qualitätskontrolle des MapMatching konnte in den Vorgängerversionen des JXMapMatchVer3 kaum praktisch geprüft werden, da eine Prüfung nur durch einen manuellen Vergleich der Vorher / Nachher GPS Koordinaten durchgeführt wurde. In Kapitel 5.3 (JXMapMatchVer3) werden die unterschiedlichen KML Dateien genauer erläutert.

- ***.route.osm.xml**
  Bei dieser Datei handelt es sich um eine OSM Datei. Sie beinhaltet nur die nodes und ways, die für den gematchten OSM way Pfad (Route) nötig sind.

3.2.4. Abbildung von GPS Punkten auf die Route

Wenn eine optimale Route (Pfad von wayParts) gefunden wurde, die gegebenenfalls manuell korrigiert wurde und aus einem Teil besteht, können im nächsten Schritt die GPS Punkte auf diese Route abgebildet werden. Dieser Schritt passiert nicht automatisch direkt nach dem N Route Algorithmus, damit der Anwender die Möglichkeit erhält vorab

die Route zu kontrollieren und Korrekturen vorzunehmen.
Auch für diese Aufgabe gibt es einen Algorithmus, um eine gute bis optimale Abbildung zu finden.
Der erste GPS Punkt wird auf den ersten wayPart der Route abgebildet. Dabei wird er auf die Position im wayPart abgebildet (*Längenposition*), die dem GPS Punkt am nächsten ist. Der nächste GPS Punkt betrachtet den aktuellsten wayPart und die nächsten h vielen wayParts auf der Route. Dieser GPS Punkt wird nun auf den wayPart abgebildet, zu dem die Entfernung minimal ist. Diese Schritte wiederholen sich bis alle GPS Punkte auf wayParts abgebildet sind. Nun wird diese Summe aller Entfernungen der GPS Punkte zu ihren wayParts abgespeichert.
Dieses Vorgehen wird für h von 1 bis 10 wiederholt. Es wird nun die Abbildung von GPS Punkten auf wayParts ausgewählt, bei der die Summe aller Entfernungen minimal ist. Dieses Vorgehen stellt sicher, dass es auch wayParts ohne GPS Punkte auf der Route geben kann und diese von GPS Punkten übersprungen werden können. Die Auswahl der minimalen Summe der Entfernungen stellt sicher, dass am Anfang nicht zu viele wayParts ausgelassen werden, weil dies für spätere GPS Punkte zu deutlich schlechteren Abbildungen führen kann.

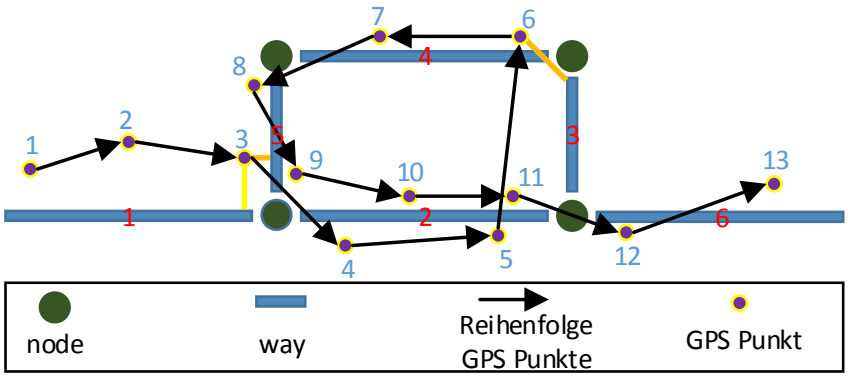

Abbildung 3.3.: Beispiel für GPS Punkte auf Route abbilden

Abbildung 3.3 zeigt einen kleinen OSM Graphen und soll die Notwendigkeit der Auswahl der minimalen Summe der Entfernungen für h von 1 bis 10 verdeutlichen. Die gefundene Route auf dieser Abbildung besteht aus dem way-Pfad 1, 2, 3, 4, 5, 2 und 6. Bei h = 1 wird nur der aktuelle und nächste way (wayPart) für das Abbilden eines GPS Punktes betrachtet. Für h = 1 wird der 6. GPS Punkt nicht auf way 4, sondern auf way 3

abgebildet. Erst für h = 2 ist, wird der 6. GPS Punkt auf den aktuellen way (2) und auf die zwei nächsten ways (3 und 4) untersucht und somit auf den richtigen way (4) abgebildet. Mit h = 4 werden für die Abbildung vom dritten GPS Punkt der aktuelle way (1) und die vier nächsten ways, also way 2, 3, 4 und 5, untersucht. Da sich way 5 am nähsten an dem dritten GPS Punkt befindet, wird der dritte GPS Punkt auf way 5 abgebildet und way 5 ist der aktuelle way. Mit h >= 4 wird gezeigt, dass zu viele ways übersprungen wurden, denn alle folgenden GPS Punkte, und somit auch GPS Punkte 6, 7, 8 und 9, müssen auf die nächsten ways der Route (way 2 und 6) abgebildet werden. In diesem Beispiel liefert h = 2 und h = 3 die minimale Summe der Entfernungen aller GPS Punkte.

3.2.5. Abbildung von Messwerten auf die Route

Als Nächstes müssen die Messwerte des RMF auf die Route abgebildet werden.
Die Messwerte werden mit Hilfe ihres Zeitstempels und der Zeitstempel der GPS Punkte auf die Route linear interpoliert. Dies soll mit einem kleinen und einfachen Beispiel erklärt werden:
Der Zeitstempel vom Messwert liegt zeitlich zwischen zwei GPS Zeitstempeln.
GPS Punkt 1 hat den Zeitstempel t_1 = 17 und Längenposition auf der Route l_1 = 100.
GPS Punkt 2 hat den Zeitstempel t_2 = 19 und Längenposition auf der Route l_2 = 115.
Der Messwert hat den Zeitstempel t_3 = 17,4.
Gesucht ist die Längenposition (l_3) des Messwertes auf der Route. Diese wird mit folgender Formel ermittelt:

$$l_3 = l_1 + ((l_2 - l_1) * \frac{(t_3 - t_1)}{(t_2 - t_1)})$$

$$l_3 = 100 + ((115 - 100) * \frac{(17{,}4 - 17)}{(19 - 17)})$$
$$l_3 = 100 + (15 * \frac{0{,}4}{2})$$
$$l_3 = 100 + (15 * 0{,}2)$$
$$l_3 = 100 + 3$$
$$l_3 = 103$$

Das Ergebnis dieser Abbildung beziehungsweise des MapMatchings wird in Abbildung 5.4 dargestellt.

3.2.6. Abbildung von Messwerten auf edges

Für die Simulation müssen die Messwerte auf die edges abgebildet werden. Hierfür muss jeder wayPart mit der edge, die netconvert aus ihm gebildet hat, referenziert werden. Für das Bilden dieser *way-edge-Paare* ist kein komplexer Algorithmus nötig. Denn jeder way besteht genau aus einem wayPart und kann somit mit genau einer edge verknüpft werden. Dies wurde durch die Aufbereitung des OSM Graphen vom OsmParser vorbereitet. Der OsmParser und die Aufbereitung des OSM Graphen sind in Kapitel 3.2.1 (OsmParser) beschrieben. Die meisten Aufgabenschritte des OsmParsers dienen einer bijektiven Abbildung zwischen den Kantenmengen des Simulators (edges) und des OSM Graphen (ways / wayParts). Da die id einer edge aus der id des way und den ids der zwei nodes gebildet wird, kann JXMapMatchVer3 zu jedem way die passende edge verknüpfen. Listing 3.3 zeigt einen way (id: 1000596) der aus einem Start-node (id: 1573051002) und einem End-node (id: 1836570477) besteht. Die id der passenden edge wird nun wie folgt vom Simulator (netconvert) gebildet:

way-ID _ Start-node-ID _ End-node-ID _ Start-node-ID
1000596_1573051002_1836570477_ 1573051002

Listing 3.3: Erstellung der edge ID

```
1  OSM nodes und way:
2
3  <node id="1573051002" lat="51.1227132" lon="6.7750606" version="1" timestamp="2012-01-01T19:21:12Z" changeset
       ="10264737" uid="8587" user="innuendo"/>
4  <node id="1836570477" lat="51.1227595" lon="6.7750414" version="1" timestamp="2012-07-24T13:40:01Z" changeset
       ="12466871" uid="141931" user="Sharlin"/>
5
6  <way id="1000596" version="23" timestamp="2013-05-17T20:26:21Z" changeset="16172600" uid="510184" user="XaMi
       ">
7          <nd ref="1573051002"/>
8          <nd ref="1836570477"/>
9          <tag k="highway" v="residential"/>
10         <tag k="maxspeed" v="30"/>
11         <tag k="name" v="Am Schwimmbad"/>
12         <tag k="old_way_id" v="23645367"/>
13         <tag k="oneway" v="yes"/>
14         <tag k="old_oneway" v="yes"/>
15 </way>
16
17 ____________________________________________________________________________________________________
18 Simulator/netconvert edge:
19
20 <edge id="1000596_1573051002_1836570477_1573051002" from="1573051002" to="1836570477" priority="-1">
21         <lane id="1000596_1573051002_1836570477_1573051002_0" index="0" speed="8.33" length="4.02" shape
               ="2843.76,42.01 2842.87,45.93"/>
22         <lane id="1000596_1573051002_1836570477_1573051002_1" index="1" speed="8.33" length="4.02" shape
               ="2840.55,41.27 2839.65,45.20"/>
23 </edge>
```

Somit wird deutlich, dass der OsmParser und seine Bearbeitung des Ursprungs-OSM-Graphen von entscheidender Bedeutung sind. Denn wir können nur den OSM Graphen bearbeiten, der dem Simulator vorgegeben wird. Wir haben weder Einfluss darauf, wie dieser die OSM Informationen aufarbeitet noch darauf, wie und welche edges er für seinen Graphen erstellt.

Die Messwerte sind - wie in Kapitel 3.2.5 (Abbildung von Messwerten auf die Route) beschrieben - auf die ways (wayParts) abgebildet. Ihre genaue Position auf dem way wird durch ihre Längenposition (Offset auf dem way) angegeben. Diese Längenposition auf dem way wird linear auf die Längenposition der edge interpoliert. Somit kann jeder Messwert vom way auf seine edge abgebildet werden und hat eine eindeutige und genaue Position auf der edge.

3.2.7. Abbildung von Modemdaten

In der Datei *cellinfo.txt* ist hinterlegt, wann das Mobilfunkmodem mit welcher Mobilfunkbasisstation verbunden war. Diese notwendigen vier Daten (bei 2G *Location Area Code*, *CellID* und bei 3G *Channel*, *Primary Scramblingcode*) sind mit einem Zeitstempel versehen und werden jedem Messwert hinzugefügt. Für diese Zuweisung wird der Zeitstempel des Messwerts und der Modemdaten herangezogen. Jeder Messwert erhält die Modemdaten, die in der Vergangenheit liegen, aber am aktuellsten sind. Diese Modeminformationen werden im nächsten Schritt vom DataUnion abgerufen und verarbeitet.

3.2.8. DataUnion

Das DataUnion verarbeitet und vereint die primäre Ausgabedatei des JXMapMatchVer3. Es hat folgende Aufgaben und Funktionen:

- Verschiedene Mobilfunkzellen erkennen
- Messwerte aggregieren/filtern

- Mehrere Ausgabedateien des JXMapMatchVer3 zusammenfassen
- matchtedLinkNr vereinen

Jede Überfahrt des Messfahrzeugs über einen Straßenabschnitt eines wayParts erzeugt einen *matchtedLinkNr*. Dieser Begriff und seine Funktion werden in Kapitel 7 (Daten-Aggregation) ausführlich erklärt. DataUnion stellt sicher, dass der matchedLinkNr, der in zwei Messfahrten unabhängig vergeben wurde, aber identisch ist, sich in der gemeinsamen Ausgabe unterscheidet. Dies ist wichtig, damit die Messdaten aus zwei Messfahrten nicht vermischt werden.

3.2.8.1. Identifizierung von Mobilfunkzellen

Das Mobilfunkmodem ist während einer Messfahrt mit einer primären Basisstation verbunden, um Daten senden und empfangen zu können. Dieser Bereich wird als Mobilfunkzelle bezeichnet. Ziel ist es eine Basisstation (Mobilfunkzelle) eindeutig zu identifizieren. Jede erkannte Basisstation erhält vom DataUnion eine fortlaufende Nummer. Jeder Messwert erhält eine dieser Nummern, so dass klar unterschieden werden kann, ob das Mobilfunkmodem mit der gleichen oder unterschiedlichen Basisstationen verbunden war als zwei Messwerte gemessen wurden. Zwei benachbarte Mobilfunkzellen dürfen nicht die gleichen Modeminformationen (bei 2G *Location Area Code* und *CellID* und bei 3G *Channel* und *Primary Scramblingcode*) haben, da sie sich gegenseitig stören würden.
Für die Identifikation einer Mobilfunkzelle sortiert DataUnion zuerst alle Messwerte entsprechend ihrer Modeminformationen in Gruppen ein. Messwerte, die unterschiedliche Modeminformationen haben, befinden sich in unterschiedlichen Gruppen und waren auch mit unterschiedlichen Mobilfunkbasisstationen verbunden. Messwerte, die gleiche Modeminformationen haben, befinden sich in gleichen Gruppen und konnten mit der gleichen Basisstation verbunden sein - mussten es aber nicht. Um zu entscheiden, ob zwei Messwerte mit gleichen Modeminformationen mit der gleichen Basisstation verbunden waren, wird mit Hilfe ihrer GPS Positionen ihre Entfernung untersucht. Wenn ihre Entfernung einen parametrisierbaren Wert überschreitet, dann waren sie mit unter-

schiedlichen Basisstationen verbunden. Je nach Verbindungsart (2G oder 3G) können Mobilfunkzellen einen Durchmesser von einigen Kilometern haben. Um zwei Mobilfunkzellen mit den gleichen Modeminformationen als unterschiedliche Mobilfunkzellen zu identifizieren, hat sich eine Entfernung von 5000 Metern bewährt. Dieses Ergebnis wurde aus mehreren Messfahrten während der Anfertigung dieser Masterarbeit und der vorherigen Projektarbeit, die sich ebenfalls mit Größen von Mobilfunkzellen beschäftigte, gewonnen.

3.2.8.2. Zusammenfassen von Messwerten

In der primären Ausgabedatei des JXMapMatchVer3 sind die aufbereiteten und gematchten Messwerte einer Messfahrt gespeichert. DataUnion vereinigt diese Daten von mehreren Messfahrten und speichert sie in eine Datei, ohne diese zu filtern oder zu aggregieren.

3.2.8.3. Aggregation von Messwerten

DataUnion gibt dem Anwender die Möglichkeit die Messdaten zu aggregieren. Dies erlaubt eine direkte Einflussnahme auf die Messdaten, die vom Simulator für die Netzwerkcharakteristiken benutzt werden. So kann der Anwender zum Beispiel Messwerte mit kleiner Latenz oder Messwerte mit niedriger Datenrate bevorzugen.
Messwerte, die während einer Fahrt auf einen Straßenabschnitt, der eine edge darstellt, gemessen werden, werden zu einer **Gruppe** zusammengefasst. DataUnion filtert für jede edge alle bis auf eine Gruppe heraus. Die Auslese der nicht herausgefilterten Gruppe pro edge wird in zwei Schritten durch geführt.

Der Anwender soll mit diesem Programm die Möglichkeit erhalten, eigenständig bestimmen zu dürfen, welche Messwert-Gruppe für eine edge für die Ausgabedatei übernommen werden soll. Je nach Szenario, das im Simulator getestet und untersucht werden soll, kann es erwünscht sein eine Gruppe mit den kleinsten *delay* oder mit einer großen, jedoch nicht der größten, *datarate* auszusuchen.
Die Auswahl der gewünschten Gruppe findet in zwei Schritten statt:

- Für jede Gruppe wird ein Repräsentant aus den Messwerte der Gruppe ausgesucht oder berechnet.

- Für jede edge wird eine Gruppe ausgesucht. Für die Gruppenauswahl wird der Repräsentant der Gruppe zu Hilfe genommen.

Die genauen Möglichkeiten und Einflussnahme des Anwenders auf die Repräsentantberechnung und die anschließende Gruppenauswahl sind in Kapitel 7 (Daten-Aggregation) ausführlich erläutert.

DataUnion erstellt zwei Dateien als Ausgabe. In der ersten Datei sind die zusammengefassten und/oder gefilterten/aggregieren Messwerte enthalten; in der zweiten Datei, die den Zusatz *_Sum* im Namen hat, sind die gruppierten und aggregieren Messdaten zu jeder Gruppe von jeder edge enthalten. Dieser Datei kann entnommen werden, wie viele Messwerte eine Gruppe hat, und welcher Repräsentant sich nach jeder Auswahl- und Berechnungsvariante ergäbe. Mit Hilfe der Daten aus der zweiten Datei (*_Sum*) wurde untersucht, ob ein Muster identifiziert werden kann, welches eine gute automatische Berechnungsauswahl des Repräsentanten erlaubt.

3.2.9. OsmJoin

Dieses Programm vereinigt mehrere OSM Graphen zu einem OSM Graphen. Angewendet wird dieses Programm auf die OSM Graphen der Ausgabe des JXMapMatchVer3. Diese OSM Graphen des JXMapMatchVer3 beinhalten nur die nodes und ways der gematchten Route. Wenn für eine Simulation Messdaten und Routen von mehreren Messfahrten genutzt werden sollen, kann mit diesem Programm ein minimaler OSM Graph, der nur die nötigsten nodes und ways aller gematchten Routen enthält, erstellt werden. Dabei wird sichergestellt, dass alle ids der nodes und ways gleich bleiben und gleichzeitig Doppelungen von nodes und ways vermieden werden. Somit bleibt die bijektive Abbildung von ways (wayParts) und edges erhalten.
Genau wie der OsmParser dient auch OsmJoin dem Ziel dem Simulator eine möglichst kleine OSM Datei zur Verfügung zu stellen, da es zu einem Timeout Fehler kommen

kann, wenn der Simulator zu lange für das Laden und Verarbeiten der OSM Datei benötigt.

OsmJoin hat noch eine zweite Aufgabe. Der Simulator erstellt seinen eigenen Graphen mit eigenem Koordinatensystem. Dabei kann es passieren, dass, wenn eine Straße am Kartenrand liegt, eine ihrer Fahrspuren außerhalb des Koordinatensystem liegt und negative Koordinaten erhält. Abbildung 3.4 skizziert eine solche Situation mit negativen Koordinaten für die untere Fahrspur (lane).

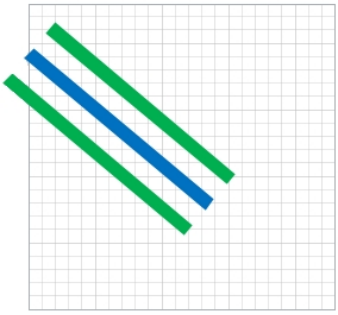

Abbildung 3.4.: Simulator Koordinatensystem, edge (blau), lanes (grün)

Um diesen Fehler zu verhindern, fügt OsmJoin ganz im Norden und Süden zwei einspurige Straßen hinzu. Diese zwei Straßen sind jeweils einspurig, zu keiner anderen Straße verbunden und werden somit vom Simulator nicht für die Fahrzeugbewegung berücksichtigt. Anfang und Ende dieser zwei Straßen liegen in den Ecken des Graphen, der um ein Offset vergrößert wurde. Auf diese Weise ist sichergestellt, dass reguläre Straßen und ihre Fahrspuren während der Simulation keine negativen Koordinaten erhalten und es folglich zu keinem Fehler kommt.

3.3. Reorder

Es wurden bereits verschiedene Ursachen und Gründe für das GPS Rauschen und die GPS Unterschiede zwischen gemessener Wirklichkeit und OSM ways genannt. Dies hat zur Folge, dass gematchte GPS Positionen auf der gefundenen OSM Route vor- und zurückspringen können. Um diese nicht stattgefundene Bewegung auszugleichen und

zu reduzieren wurde eine *reorder Funktion* entworfen und implementiert. Das Vorgehen dieser Lösung lautet wie folgt:
Alle gematchten GPS Punkte werden ihrem Zeitstempel entsprechend der Reihe nach durchgegangen. Wenn ein GPS Punkt gefunden wird, der auf der Route zurückspringt, wurde der Beginn einer Menge von GPS Punkten gefunden, die *reordert* werden müssen. Die Menge dieser Punkte bildet einen Abschnitt auf der Route. Die Grenze von diesem Abschnitt wird durch die GPS Punkte angegeben. Die linke Grenze des Abschnittes wird durch den GPS Punkt angegeben, der am weitesten auf der Route zurückspringt. Das Ende dieser Menge (rechte Grenze) wird durch einen GPS Punkt markiert, der folgende Eigenschaften hat:

- Der GPS Punkt befindet sich nicht mehr im besagtem Abschnitt.
- Nach ihm folgen mindesten 10 GPS Punkte, deren Position auf der OSM Route vorwärts verläuft.
- Die Entfernung zwischen der rechten Grenze und des frühstens 10. nächste GPS Punkt muss größer sein, als die Länge des *reorder-Abschnittes*.

Bei der Suche nach dem GPS Punkt für die rechte Grenze, kann es passieren, dass weitere GPS Punkte nach hinten springen, der Menge hinzugefügt werden, die Grenzen sich verschieben und der reorder-Abschnitt somit vergrößert wird. Nachdem der Abschnitt und die Menge der betroffenen GPS Punkte gefunden wurden, werden diese Punkte auf dem Abschnitt mit Hilfe ihrer Zeitstempel linear interpoliert.
Die festgesetzten numerischen Eigenschaften für die Suche der rechten Grenze wurden experimentell ermittelt und haben sich in der Praxis bewährt.

3.4. Übersicht der Abbildungen

Der Übersicht halber sollen in diesem Kapitel alle Arbeits- und Abbildungsschritte zusammengefasst werden, die erforderlichen sind, um Messwerte auf den Graphen des Simulators zu positionieren:

- OSM Datei vorbereiten (OsmParser)
- Simulator Graph erstellen
- Route mit dem N Route Algorithmus suchen (JXMapMatcherVer3)
- Route prüfen und gegebenenfalls korrigieren (JXMapMatcherVer3)
- GPS Punkte auf Route (way) abbilden (JXMapMatcherVer3)
- gematchte GPS Punkte mit *reorder* optimieren (JXMapMatcherVer3)
- kritische Situationen in Abbiegungen erkennen (JXMapMatcherVer3) (siehe Kapitel 6.1.3 - Projektion: MapMachting in Abbiegungen)
- GPS Punkte auf edge abbilden (JXMapMatcherVer3)
- Modemdaten auf Messwerte abbilden (JXMapMatcherVer3)
- Messwerte auf edge abbilden (JXMapMatcherVer3)
- Mobilfunkzellen identifizieren (DataUnion)
- Messwerte zusammenfassen oder aggregieren (DataUnion)
- OSM Dateien zusammenfassen und aufarbeiten (OsmJoin)

Kapitel 4.

Implementierung

In den vorangegangenen Kapiteln wurden die grundlegenden Techniken sowie die allgemeinen Funktionsweisen, verschiedene Aufgaben und Lösungen vorgestellt und erklärt. Dieses Kapitel widmet sich der Implementierung relevanter Aspekte und verschafft diesbezüglich einen tieferen Einblick. Im Rahmen dieser Masterarbeit wurden die drei Programme OSMParser, DataAggragation und OsmJoin entworfen und implementiert. Zudem wurde das Programm JXMapMatchVer3 angepasst und erweitert. Es stellt somit eine Weiterentwicklung von JXMapMatcherVer2 dar. Folgend werden daher der Aufbau und die Implementierung wichtiger Funktionen, Algorithmen und Programmen erläutert. Alle vier Anwendungen sind in Java programmiert.

4.1. OsmParser

Der OsmParser hat die Aufgabe eine OSM Datei aufzuarbeiten. In der Ausgabe OSM Datei sollen nur noch für den Straßenverkehr benötigte Informationen für ein ausgewähltes Gebiet vorhanden sein. Damit zwischen dem OSM Graphen und dem netconvert Graphen eine bijektive Abbildung möglich ist, werden mehrere OSM Daten überarbeitet. Eine genaue Liste der Aufgaben und Funktionen des OsmParsers kann Kapitel 3.2.1 (OsmParser) entnommen werden.

Die OSM Dateien sind zum Teil sehr groß und haben sehr viele ways und nodes gespeichert. Bei mehreren Aufgaben muss der OsmParser Metainformationen (zum Beispiel GPS Positionen oder ids) von ways und nodes zwischenspeichern. Um sicherzustellen, dass eine Metainformation benötigt, bereits vergeben (zum Beispiel id) oder eindeutig (zum Beispiel GPS Position) ist, werden als Datenstrukturen für das Zwischenspeichern *TreeMap* und *TreeSet* benutzt. Der primäre Grund für diese Entscheidung ist ihre Laufzeit von *n * log(n)*, die sich bei sehr großen OSM Dateien stark bemerkbar macht.

Die OSM Datei wird insgesamt dreimal gelesen:

- Das erste Lesen braucht wenig Zeit und dient lediglich der Ermittlung der Zeilenanzahl. Die Zeilenanzahl wird benötigt, um dem Anwender bei den nächsten Lesedurchläufen einen Status inklusive Fortschrittsanzeige auszugeben. Diese Information ist besonders bei sehr großen OSM Dateien erwünscht.

- Der zweite Lesedurchgang bearbeitet die folgenden zwei Aufgaben:

  - Erfassen und Zwischenspeichern von ids, unabhängig von ihrer Tag-Art (way, node, relation). In der Ausgabe OSM Datei wird aus jedem wayPart ein neuer way erstellt. Um doppelte ids zu vermeiden, werden die Informationen, welche ids bereits vergeben sind, benötigt.

  - Zwischenspeichern der ids von ways, die für den Straßenverkehr relevant sind. Der Entscheidungsalgorithmus, ob ein way für den Straßenverkehr benötigt wird, wurde aus der Bachelorarbeit *Ein Demonstrator für ein Peer-To-Peer-basiertes Verkehrsinformationssystem* (2008) von Norbert Goebel übernommen und nur minimal angepasst ([go08]).

- Beim dritten Lesedurchgang wird gleichzeitig die neue OSM Datei erstellt. Von der Ursprungs-OSM Datei wird jede Zeile gelesen und in die Ausgabedatei geschrieben, falls sie benötigt wird. Beim Schreiben der neuen OSM Datei werden die Anpassungen des OSM Graphen, die in Kapitel 3.2.1 (OsmParser) beschrieben sind, durchgeführt.

Das dreimalige Lesen der Ursprungs-OSM Datei wird zeilenweise durchgeführt. Aus

Performanz- und Laufzeitgründen wurde absichtlich auf Standard Java XML Objekte, wie zum Beispiel *XMLInputFactory* und *XMLStreamReader*, verzichtet. Das zeilenweise Auslesen reduziert die Lesezeit um bis zu 50 Prozent, da die Standard Java XML Objekte die XML Datei nicht nur lesen, sondern auch die ganze XML Datei auf Fehler prüfen und alle Tags, Elemente und Attribute verarbeiten sowie in ihre Datenstruktur abspeichern.
Eine Umsetzung des OsmParsers wäre auch mit nur einem Lesedurchgang der ursprünglichen OSM Datei möglich. Dies hätte aber zur Folge, dass alle Informationen aus der Datei (alle nodes, alle ways) im Arbeitsspeicher vorgehalten werden müssten. Denn erst wenn die Datei vollständig eingelesen wurde, kann eine endgültige Entscheidung, ob eine Information nicht benötigt wird, getroffen werden.

4.2. JXMapMatchVer3

Das JXMapMatchVer3 stellt eine Weiterentwicklung des JXMapMatchVer2 dar. JXMapMatchVer2 wurde am Lehrstuhl für Rechnernetze und Kommunikation von Daniel Sathees Elmo entwickelt, dessen Arbeit auf der Vorarbeit von Tobias Amft basiert. Zu Beginn der vorliegenden Abschlussarbeit wurden JXMapMatchVer2 und seine Arbeitsweise vorgeführt und übergeben. Der MapMatching Algorithmus des JXMapMatchVer2, der den tatsächlich gefahrenen Weg auf dem OSM Graphen finden sollte, liefert nicht immer optimale Ergebnisse. Die Ergebnisse des MapMatchings waren und sind in der JXMapMatchVer2 Benutzeroberfläche zwar zu großen Teilen gut bis sehr gut, jedoch noch nicht ganz zufriedenstellend. Durch das visuelle Ergebnis im JXMapMatchVer2 waren wir uns einig, dass nur noch kleine Verbesserung und Anpassungen notwendig sind. Somit wurde die Entscheidung getroffen kleine Änderungen am vorhandenen Programmiercode des JXMapMatchVer2 vorzunehmen (zum Beispiel Abbilden der Messwerte) und das Programm um fehlende Funktionen zu ergänzen. Für diese Umsetzung sollten die bestehenden Klassen, Algorithmen, Strukturen, Abhängigkeiten sowie Funktionsweise nicht verändert werden. Es sollten hingegen neue Klassen hinzugefügt werden, die an möglichst wenigen Stellen mit der bestehenden Lösung verknüpft (referenziert) werden.
Im Laufe der Einarbeitung in den JXMapMatchVer2 und der Implementierung des JXMapMatchVer3 hat sich gezeigt, dass an bestimmten Stellen größere Anpassungen un-

vermeidbar sind. Vor allem die Ursachensuche (Debugging) für die nicht nachvollziehbaren gefundenen OSM Routen des JXMapMatchVer2 hat sich als erheblich zeitaufwendiger und komplexer herausgestellt. Des Weiteren mussten punktuell wichtige Klassen ausgetauscht werden.

4.2.1. Relevante Klassen

In der aktuellen Version besteht JXMapMatchVer3 aus 12 *packages*, 38 Klassen und vier externen *Referenced Libraries*. Die Bezeichnung der packages, Klassen und Funktionen sind aussagekräftig und geben ihre Funktion an. Die Anzahl der Klassen und Funktionen und ihre Referenzen untereinander sind zum Teil komplex und wurden größtenteils vom JXMapMatchVer2 aus den oben aufgeführten Gründen übernommen und beibehalten. Um eine schnelle Übersicht über die Struktur und Funktionsweise des JXMapMatchVer3 zu gewinnen in der folgenden Aufzählung die wichtigsten Klassen und Funktionen beschrieben:

- Das package *jxmapmatch* beinhaltet die drei Einstiegsklassen *JXMapMatchController*, JXMapMatch und JXMapMatchGUI mit der primäre *main* Funktion.

- Der eingelesene OSM Graph wird in den Klassen *myOSMMap*, *myOSMNode*, *myOSMWay* und *myOSMWayPart* gespeichert und mit der Funktion *loadMapFiles* aus *myOSMMap* eingelesen.
  Die Einlesefunktionsweise wurde aus der Bachelorarbeit *Ein Demonstrator für ein Peer-To-Peerbasiertes Verkehrsinformationssystem* (2008) von Norbert Goebel übernommen. Da aktuell der OSMParser die OSM Datei stark verkleinert und vorfiltert, ist die Ladezeit trotz Standard Java XML Klassen (*XMLInputFactory*, *XMLStreamReader*) sehr kurz. Um die Ladezeit weiter zu verkürzen, wurde die Überprüfung, ob ein way für den Straßenverkehr relevant ist, mit der Variable *useOsmParserFile* in der Funktion *addWay* deaktiviert. Falls OSM Dateien benutzt werden, die nicht vom OsmParser erstellt wurden, sollte diese Überprüfung wieder aktiviert werden.

- Mit der Klasse *myCellInfo* werden die Modemdaten verwaltet. Die Funktion *loadCellInfos* liest die Datei *cellinfo.txt* ein und hat einen Vektor mit allen Modemdaten als Rückgabewert.
  cellinfo.txt ist eine Textdatei und wird zeilenweise in einer Schleife eingelesen. Die einzelnen Modemdaten sind durch eine Zeile mit Rautezeichen (#) getrennt. Nach einer Rautezeile wird immer ein Zeitstempel als numerische Zahl erwartet. Anschließend wird der Anfang der nächsten Zeilen nach einem *Schlüsselwort* durchsucht. Diesen Schlüsselwort gibt an, dass in dieser Zeile die gesuchten Modemwerte gespeichert sind. Die Modemwerte werden anschließen durch Zeichenkettenoperationen extrahiert und abgespeichert. In Abhängigkeit des verwendeten Mobilfunkmodems und seiner Firmware können das Schlüsselwort und die Codierung der gesuchten Modemwerte unterschiedlich sein.
  Dieses einfache, zeilenweise Einlesen und Suchen nach einem Schlüsselwort erlaubt es weitere Mobilfunkmodemmodelle problemlos hinzuzufügen. In Listing 4.1 werden für das Mobilfunkmodem *Ericsson H5321 gw* bei einer 3G Verbindung die Modemwerte *Location Area Code* und *CellID* bei einer 2G Verbindung die Werte *Channel* und der *Primary Scramblingcode* durch die Schlüsselwörter **EWSCI:* und *+CREG:* erkannt und anschließend ausgelesen.

Listing 4.1: Beispiel Erkennen und Einlesen von Modemwerten

```
1  ...
2  else if (line.startsWith("*EWSCI: ")) {
3          line = line.replace("*EWSCI: ", "");
4          String lines[] = line.split(",");
5
6          ci.w1_ch = lines[0].replace("\"", "").trim();
7          ci.w1_sc = lines[1].replace("\"", "").trim();
8  } else if (line.startsWith("+CREG: ")) {
9          line = line.replace("+CREG: ", "");
10         String lines[] = line.split(",");
11
12         ci.g1_lac = lines[2].replace("\"", "").trim();
13         ci.g1_cellid = lines[3].replace("\"", "").trim();
14 }
```

- Mit der Klasse *myDataset* werden die Messwerte der Dateien *downstream-data.csv* und *upstream-data.csv* verwaltet und mit der Funktion *loadDatasets* eingelesen. Da es sich bei diesen Dateien um Tabellen im *Character-separated values* Format handelt, ist das zeilenweise Einlesen trivial. Die Angabe in welcher Spalte die benötigten Werte für einen Messwert vorliegen, wird vorab mit den Funktionen *loadDatasetsUp* und *loadDatasetsDown* ermittelt.

- Mit Hilfe der Klasse *myEdge* wird der netconvert Graph eingelesen und seine edges gespeichert. Da die eingelesene XML Datei eien großen Umfang annehmen kann, wird ein zeilenweises Einlesen bevorzugt. Die Performance-Vorteile und Gründe für diese Entscheidung sind identisch mit denen, die in Kapitel 3.2.1 (OsmParser) vorgestellt werden.

- Die Einstiegsfunktion für den N Route Algorithmus lautet *executeNRouteAlgorithm* und ist in der Klasse *NRouteAlgorithm* implementiert. Die Funktionsweise des Algorithmus ist in Kapitel 3.2.2 (N Route Algorithmus - die optimale OSM Route) beschrieben.

- Nach dem N Route Algorithmus kann die OSM Route vom Anwender mit der Computermaus korrigiert werden. Diese Funktionalität ist in der Klasse *SelectedNRoute* umgesetzt. Die Klasse *JXMapMatchController* fängt die *MouseEvents* ab und startet dann die nötigen Funktionen (setEditableLinks, addLink, deleteLink) von *SelectedNRoute*.

- Die Klasse *MatchGPStoNRouteAlgorithm* beinhaltet die die nötigen Funktionen für das Abbilden der gemessenen GPS Positionen auf die OSM Route.

- Die in Kapitel 3.3 (Reorder) vorgestellte *reorder-Funktion* ist der Funktion *ReorderedMatchedGPSNode.reorderMatchedGPSNodes* umgesetzt.

- Das Abbilden der Messwerte auf die OSM Route aus Kapitel 3.2.5 (Abbildung von Messwerten auf die Route) ist in der statischen Funktion *myDataset.matchMatched GPSNode* implementiert.

- Das Speichern der Ergebnisse und Erstellen der Ausgabedateien wird vom JXMapMatcherVer3 in der Klasse *mySaveToFile* umgesetzt. In dieser Klasse gibt es mehrere Funktionen mit aussagekräftiger Bezeichnung (zum Beispiel *createOsm* oder *createOsmKml*).
  In dieser Datei befinden sich 12 symbolische Konstanten, die die Farbverteilung der KML, welche die Messwerte farbig darstellen, steuern (zum Beispiel *MIN_DE LAY _UP = 0* oder *MAX_DATARATE_DOWN = 7200000 / 8*.

4.2.2. Erkenntnisse und Änderungen

Das Einarbeiten und Verstehen der Funktionen des JXMapMatchVer2 kann als sehr komplex und zeitaufwendig beurteilt werden. Obwohl die Klassen und Funktionen sprechende Namen haben und diese auch kommentiert sind, ist ihr verketteter Ablauf komplex. Es fehlt eine Übersicht und Zusammenfassung die nur die relevanten Komponenten und ihre Existenzberechtigung im *Big_Picture* beschreiben. Das Einarbeiten wird, durch die Tatsache, dass einige Funktionen und Berechnungen, überflüssig oder fehlerhaft sind, erschwert. Folgende drei Beispiele zeigen wichtige Erkenntnisse, die beim Entwickeln, Debugging und Testen gewonnen wurden. Sie machen zum Teil einfache aber wichtige Unterschiede zwischen dem JXMapMatchVer2 und JXMapMatchVer3 und somit auch zum Teil zum ursprünglichen MapMatching Algorithmus von Fabrice Marchel und Kay W. Axhausen aus.

- Nicht auf jedem way wird ein GPS Punkt gematcht.

- Eine *intersection Postion*[1] existiert nicht. Praxisnahe Tests haben gezeigt, dass selbst wenn auf jede way ein GPS Punkt gematcht wird, sollten und müssen alle nächsten ausgehenden ways untersucht werden.

- Beim JXMapMatchVer2 wird in der Funktion *createChildPathAndAddToSet* ein gematchter GPS Punkt von einer Route entfernt, um diesen dann an allen ausgehenden ways (*Children*) zu untersuchen. Dabei wurde fälschlicherweise angenommen, dass folgende Formeln für alle Zahlen (*a, b, c*) vom Type *Double* gelten.

  $$a := b$$
  $$b := b + c$$
  $$b := b - c$$
  $$a == b \text{ (falsch)}$$

  Diese kleine Ungenauigkeit hat zur Folge, dass in seltenen Fällen die gefundene OSM Route nicht mathematisch optimal und korrekt ist. Beim N Route Algorithmus kann dies dazuführen, dass ein falscher ausgehender way bevorzugt wird.

[1] Position auf einem way, der angibt, ab wann die nächsten ausgehenden ways untersucht werden sollen, wenn ein GPS Punkt auf oder über diese Position gematcht wird.

4.3. DataUnion

DataUnion besteht aus sieben Klassen, die der Aufgabenteil und der Übersicht der Datenstruktur dienen.

4.3.1. Identifikation der Mobilfunkzellen

Wie in Kapitel 3.2.8 (DataUnion) beschrieben itentifiziert DataUnion alle Mobilfunkbasisstationen und gibt jedem Messwert die id der Mobilfunkzelle beziehungsweise der identifizierten Basisstation. Diese Aufgabe wird in der Klasse *CellIdentification* implementiert. Sie hat unter anderem die vier Werte der Modemdaten (*Location Area Code*, *CellID*, *Channel* und *Scramblingcode*) und die minimalen und maximalen Breiten- und Längengrade der Messwerte, mit denen sie referenziert ist, in Variablen gespeichert. Die Identifikation der Mobilfunkzelle funktioniert wie folgt:
In einem Vektor (*CellIds*) werden alle Instanzen von *CellIdentification* gespeichert, welcher am Anfang leer ist. Jeder Messwert wird geprüft, ob er zu einer Instanz von *CellIdentification* aus *CellIds* gehört. Bei dieser Prüfung werden zuerst die vier Modemdaten der *CellIdentification* und des Messwerts verglichen. Wenn diese gleich sind, wird mit Hilfe der minimalen und maximalen Breiten- und Längengrade der *CellIdentification* der geografische Mittelpunkt berechnet. Anschließend wird die Entfernung des Mittelpunktes und des Messwertes untersucht. Falls die Entfernung kleiner oder gleich eines parametrisierten Wertes (*DistanceToNextCell*) ist, wird diese *CellIdentification* dem Messwert zugewiesen und referenziert. Dabei werden gegebenenfalls die minimalen und maximalen Breiten- und Längengrade der *CellIdentification* aktualisiert.
Wenn für einen Messwert alle Mitglieder von *CellIds* untersucht werden und keine passende *CellIdentification* gefunden wird, wird eine neue Instanz von *CellIdentification* erstellt, die Modemwerte und GPS Position des Messwertes übernimmt und in den Vektor *CellIds* hinzufügt.
Nachdem alle Messwerte einer *CellIdentification* zugeordnet wurden, werden die Modemdaten und Entfernungen aller *CellIdentifications* untereinander untersucht. Falls zwei *CellIdentifications* die gleichen Modemdaten aufweisen und ihre jeweilige Entfernung

kleiner als *DistanceToNextCell* ist, werden die Messwerte von der zweiten *CellIdentification* in die erste *CellIdentification* übernommen.

4.3.2. Identifikation der Gruppen

JXMapMatcherVer3 speichert in seiner primären Ausgabedatei zu jedem Messwert auch die id (*matchedLinkNr*) der dazugehörigen Gruppe. Die Messwerte in dieser Datei sind nach ihrer *matchedLinkNr* sortiert. Die Definition des Begriffs *Gruppe* kann in Kapitel 3.2.8 (DataUnion) nachgelesen werden. Bei dieser id handelt es sich um eine aufsteigende Nummer, die für jedes MapMatching bei null beginnt. DataUnion fügt die neue id *matchedLinkNrGlobal* hinzu. Hierbei handelt es sich um eine über alle Messwerte aus allen Eingabedateien eindeutige Gruppen id. Auch dies ist eine fortlaufende Nummer, die bei null anfängt. Sie wird immer um eins inkrementiert, wenn sich *matchedLinkNr* oder die Eingabedatei ändert.

4.3.3. Aggregation von Messwerten

Die primäre Arbeit für die Aggregation der Messwerte übernehmen die Klassen *DataAggregation*, *DA_EdgeGroups* und *DA_DatasetGroup*. Mit diesen drei Klassen werden die Messwerte hierarchisch sortiert und gruppiert.
Dabei ist die Klasse *DatasetGroup* die unterste Ebene. In ihr werden die Messwerte einer Gruppe in einem Vektor (*Datasets*) zusammengefasst. Die Berechnung eines Repräsentanten wird in *DatasetGroup* durchgeführt. Für jede der acht Berechnungsarten und jeden der drei Wertearten (datarate, delay und loss_rate) gibt es eine Variable, in der der Repräsentant gespeichert wird (zum Beispiel *min_datarate* oder *med_delay*). Jede Berechnungsart ist in einer eigenen Funktion implementiert (zum Beispiel *setMin()* oder *setMed()*) und berechnet die Repräsentanten für alle drei Wertearten. Dabei werden alle Messwerte aus dem Vektor *Datasets* berücksichtigt. Der Ausnahme stellt ein Wert von *-1* dar. Der Wert *-1* gibt an, dass das RMF einen Messfehler und/oder einen ungültigen Wert gemessen hat. Neue Berechnungsarten des Repräsentanten könnten in diese Klasse hinzugefügt werden.

Die Klasse *DA_EdgeGroups* verwaltet alle Gruppen (DatasetGroup) einer edge in *TreeMaps.* Der *key* in dieser TreeMaps ist die Gruppen id *matchedLinkNrGlobal* und der *value* ist die *DA_EdgeGroups*. Von diesen *TreeMap* gibt es zwei Stück. Eine für die *downstream* und eine für die *upstream* Messwerte und Gruppen. Die Gruppenauswahl wird in dieser Klasse in der Funktion *SelectGroup()* umgesetzt. Hierfür werden alle Gruppen pro *Streamrichtung* (*down/up*) in eine temporäre *TreeMap* hinzugefügt. Als *key* wird der bereits berechnete Repräsentant jeder Gruppe genutzt. Der temporäre *TreeMap* wird ein neuer *comparator Operator* übergeben, der sicherstellt, dass auch zwei Element den gleichen *key* haben dürfen. In der temporären *TreeMap* sind somit alle Gruppen nach ihrem Repräsentanten sortiert. Anschließend wird nach eingestellter Gruppenauswahlart die Gruppe aus dieser *TreeMap* berechnet und ausgewählt.
In der Klasse *DataAggregation* werden in einer *HashMap* die *DA_EdgeGroups* verwaltet. Der *key* in dieser Map ist die edge id *matchedLinkNrGlobal* und der *value* ist die *DA_EdgeGroups*.

4.4. OsmJoin

Die Aufgaben und Funktionsweise des OsmJoin können, verglichen mit den drei anderen Programmen, als einfach bezeichnet werden. Es mussten keine komplexen Berechnungen oder Algorithmen implementiert werden. OsmJoin vereinigt OSM Graphen aus mehreren OSM Dateien und speichert den neuen OSM Graphen in eine neue OSM Datei. Dabei wird auf zwei Aspekte geachtet:

- Das Kartengebiet wird um einen Offset vergrößert, um ganz im Norden und Süden jeweils eine neue Straße hinzuzufügen. Die Gründe und Erklärung werden in Kapitel 3.2.9 (OsmJoin) ausgeführt.

- Alle nodes und ways und ihre ids müssen eindeutig sein.

Um sicherzustellen, dass die ids der zwei neuen Straßen beziehungsweise der zwei neuen ways eindeutig sind, werden zunächst alle Eingabe OSM Dateien vom OsmJoin eingelesen; sodann werden ihre verwendeten ids in einem *Set* (*allIDs*) abgespeichert.

Anschließend wird die neue OSM Datei erstellt. Zuerst werden alle nodes der Eingabe OSM Dateien eingelesen und in der Ausgabedatei gespeichert. Im Anschluss wird diese Prozedur für alle ways wiederholt. Bei beiden Schritten wird sichergestellt, dass jede node und jeder way nur einmal in die neue Ausgabedatei übernommen werden. Um dies sicherzustellen, wird die id einer bereits übernommenen nodes oder eines ways in ein Set (*nodeIDs* beziehungsweise *wayIDs*) gespeichert. Vor der Übernahme eines ways muss geprüft werden, ob sich die id in dem jeweiligen Set (*nodeIDs* beziehungsweise *wayIDs*) befindet.
Zwischen den übernommenen nodes und ways der Eingabedateien werden die zwei neuen ways und ihre vier nodes hinzugefügt. Für diese sechs neuen OSM Tags wird eine neue id generiert. Da zuerst alle ids eingelesen und in einem Set (*allIDs*) hinterlegt wurden, ist die Sicherstellung der Eindeutigkeit der neuen ids problemlos möglich.

Kapitel 5.

Anwendung

Die vorliegende Arbeit stellt eine Synthese mehrerer Programme dar, die gemeinsam und nacheinander die gemessenen Netzwerkcharakteristiken (Messwerte) aufarbeiten. Die Programme und Ergebnisse dieser Masterarbeit sollen auch in naher Zukunft praktische Anwendung finden, weshalb in diesem Kapitel die Verwendung von allen Programmen detailliert beschrieben wird. Eine ausführliche Beschreibung der Aufgaben und Funktionsweisen der einzelnen Programme wird in Kapitel 3.2 (Aufgaben, Probleme, Lösungen und Grundkonzeption) aufgeführt. Dieses Kapitel widmet sich der direkten Anwendung und den Parametern der Programme. Es werden auch an einzelnen Stellen Hinweise und Vorschläge vorgebracht, um die Programme und ihre einzelnen Zwischenergebnisse möglichst effizient anwenden zu können .

In der Ausgangssituation sind Messwerte vorhanden und befinden sich ebenfalls auf dem Datenträger zu dieser Masterarbeit. Im Folgenden werden beispielhaft die Daten und Messwerte aus fünf Messfahrten aufbereitet:
\20150803\090007\
\20150803\091227\
\20150803\092532\
\20150803\093139\
\20150825\172937\

5.1. OsmParser

Zuerst wird die OSM Datei durch den OsmParser aufgearbeitet. Ihm müssen und können folgende Informationen mit Parametern übergeben werden:

-io *Dateipfad*
Mit diesem Parameter erhält der OsmParser den Dateipfad zur ursprünglichen OSM Datei.

-oo *Dateipfad*
Mit diesem Parameter erhält der OsmParser den Dateipfad zur neuen OSM Datei, die er erstellt.

-g *Dateipfad*
Mit diesem Parameter erhält der OsmParser den Dateipfad zur GPS-Log-Datei.

-e *lat lon lat lon*
Mit diesem Parameter erhält der OsmParser zwei GPS Koordinaten in Breitengrad (*lat*) und Längengrad (*lon*).

Die ersten zwei Parameter (*-io* und *-oo*) sind obligatorisch. Falls der OsmParser einen der beiden anderen Parameter erhält, verkleinert er das Kartengebiet in der OSM Ausgabedatei. Der Parameter *-g* kann mehrfach angewendet werden.

Listing 5.1: Aufruf OsmParser mit Parameter

```
1 java OSMParser -oi C:\DVD\OSM_Dateien\duesseldorf-
    regbez-latest.osm.xml -oo C:\DVD\OSM_Dateien\
    OsmParser_Output.osm.xml -g C:\DVD\Daten_Messfahrt
    \20150803\090007\gpsd.log -g C:\DVD\Daten_Messfahrt
    \20150803\091227\gpsd.log -g C:\DVD\Daten_Messfahrt
    \20150803\092532\gpsd.log -g C:\DVD\Daten_Messfahrt
    \20150803\093139\gpsd.log -g C:\DVD\Daten_Messfahrt
    \20150825\172937\gpsd.log
```

Listing 5.1 zeigt ein Beispiel für einen OsmParaser Aufruf für die oben aufgeführten Messfahrten.

Da die Eingabe OSM Datei sehr groß ist (ca. 2,8 GB), kann diese Ausführung mehrere Minuten dauern.
Die neu erstellte OSM Datei trägt den Namen *OsmParser_Output.osm.xml* und ist ca. 10 MB groß.

Hinweis:
Es ist davon auszugehen, dass in der Praxis die Messwerte von mehreren Messfahrten verarbeitet und aggregiert werden. Dies ist aber nur möglich, wenn die OSM Datei, die vom JXMapMatcherVer3 bei Aufarbeiten jeder einzelnen Messfahrt genutzt wird stets die gleiche ist. Denn der OsmParser erstellt in seiner Ausgabe OSM Datei aus jedem wayPart einen neuen way. Diese neuen ways bekommen eindeutige und neue ids. Dies hat zur Folge, dass die Ergebnisse des JXMapMatcherVer3 nicht zusammengefasst werden können, da nunmehr ein way und somit auch eine edge unterschiedliche ids haben.
Eine Lösung besteht darin, dass der OsmParser ohne GPS-Log-Datei(en) ausgeführt wird. Das Kartengebiet der Ausgabe OSM Datei wird dann nicht verkleinert. Mit der Beispiel OSM Datei *duesseldorf-regbez-latest.osm.xml* werden die Stadt Düsseldorf sowie sehr viele ihrer Nachbarstädte abgedeckt. Hierbei ergibt sich allerdings der Nachteil, dass die Ausgabe OSM Datei des OsmParsers, die vom JXMapMatcherVer3 eingelesen wird, ca. 600 MB groß ist und somit die Ladezeit im JXMapMatcherVer3 und dem Simulator stark vergrößert.
Die alternative Lösung besteht darin, dass der Anwender im Vorfeld ein Verkehrsgebiet genau auswählt, welches er vermessen und später simulieren möchte. Zwei gegenüberliegende Eckpunkte des ausgewählten Gebiets werden dem OsmParser mit dem Parameter *-e* mitgegeben. Somit enthält die Ausgabe OSM Datei nur Kartenmaterial für das ausgewählte Gebiet. Mit dieser neuen Ausgabe OSM Datei ist es nun möglich Messfahrten mit dem JXMapMatcherVer3 zu verarbeiten. Eine Messfahrt muss auch nicht in diesem ausgesuchten Gebiet anfangen und/oder enden. Der JXMapMatcherVer3 erkennt automatisch, dass nur ein Teil der GPS Punkte im geladenen OSM Graph liegt und verwirft dann gegebenenfalls die ersten und/oder letzten GPS Punkte aus der GPS-Log-Datei, die nicht vom ausgewählten Gebiet abgedeckt werden.

5.2. Simulator Graph

Im nächsten Schritt wird mit Hilfe der neuen OSM Datei und des Simulators die XML Datei mit dem Simulator Graphen erstellt. Für den Befehl aus Listing 5.2 wird die Simulatorkomponente VSimRTT benötigt. Die genaue Einrichtung und Funktionsweise des Simulators und seiner Komponenten kann der Masterarbeit von Raphael Bialon entnommen werden.

Listing 5.2: Beispiel Simulator Graph Parameter

```
1 java -jar scenario-convert-0.14.0-SNAPSHOT.jar --
    osm2sumo -d OsmParser_Output.db -i OsmParser_Output
    .osm.xml -n
```

Im Listing 5.2 wird die OSM Datei des OsmParsers mit dem Parameter *-i* angegeben. Der Parameter *-d* bestimmt die Dateinamen der Ausgaben. In diesem Beispiel erhält die XML Datei, die den Simulator Graphen speichert, den Dateinamen *OsmParser_Output.net.xml*. Die anderen Ausgabedateien des Simulators werden nicht benötigt.

5.3. JXMapMatchVer3

Im weiteren Schritt werden die Messwerte jeder der fünf Messfahrten mit Hilfe des JXMapMatchVer3 und der oben erstellten Graph Dateien (Ausgabe des OsmParsers und Simulators) aufbereitet. Im Folgenden wird das genaue Vorgehen exemplarisch an den Daten der ersten Messfahrt erklärt.
Zuerst wird der JXMapMatchVer3 gestartet:

Listing 5.3: Start JXMapMatchVer3

```
1 java -jar C:\DVD\Programme\JXMapMatchVer3\
    eclipse_jar_export\JXMapMatchVer3.jar
```

Auf der rechten Seite der Benutzeroberfläche des JXMapMatchVer3 befinden sich mehrere Controls. Mit den obersten zwei Buttons werden die OsmParser Ausgabedatei (*OsmParser_Output.osm.xml*) und die GPS-Log-Datei (*gpsd.log*) geöffnet. Die OSM Graph Datei (*.osm.xml) und netconvert Graph Datei (*.net.xml) bilden ein Paar und werden im gleichen Verzeichnis erwartet. Des Weiteren muss der Name dieser beiden Dateien bis auf die Dateiendungen (*.osm.xml und *.net.xml) identisch sein.
Wenn der JXMapMatchVer3 Zugriff auf das Internet hat, wird im großen Hauptbereich der Benutzeroberfläche eine Landkarte angezeigt. Die Quelle für dieses Bildmaterial ist OpenStreetMap. Auf dieser Landkarte werden der geöffnete OSM Graph in lila und die gemessenen GPS Punkte in kleinen blauen Punkten (Quadraten) dargestellt. Diese Anzeige kann der Anwender mit den zwei CheckBoxen *Routing Graph* und *GPS Trace* deaktivieren. Mit den Controls unten links oder mit dem Mausrad kann in die Karte herangezoomt und mit gedrückter linker Maustaste kann die Karte verschoben werden.

Ein Klick auf den Button *N Route* aktiviert den *max. container size*. Mit diesem Number-Spinner kann das *N* des N Route Algorithmus, und somit wie viele Routen gleichzeitig nach der optimalen Route suchen sollen, eingestellt werden. Wie in Kapitel 3.2.2 (N Route Algorithmus - die optimale OSM Route) angeführt, sollte dieser Wert weder zu klein noch zu groß sein. Der Button *Route* startet den N Route Algorithmus, welcher die optimale Route finden soll. Während dieses Vorgangs werden auf der Karte alle möglichen n Routen in gelber Farbe angezeigt. Die Spitzen (aktuellste wayParts) der Routen haben die Farbe schwarz und die Spitze der aktuell optimalen Route hat die Farbe rot.

Nach dem der N Route Algorithmus seine Suche beendet hat, wird die optimale identifizierte Route in blau auf der Karte angezeigt. Nun kann der Anwender wie in Kapitel 3.2.2 (N Route Algorithmus - die optimale OSM Route) beschrieben, diese Route mit der linken und rechten Maustaste manuell korrigieren.
Der Button *Match* bildet die gemessenen GPS Punkte auf der gegebenenfalls korrigierten Route ab. (Algorithmus, siehe Kapitel 3.2.4 - Abbildung von GPS Punkten auf die Route).
Nachdem Abbilden der GPS Punkte ist der *Export* Button aktiviert. Mit ihm werden der Pfad und der Dateiname (*output.csv*) der primären Ausgabedatei festgelegt.

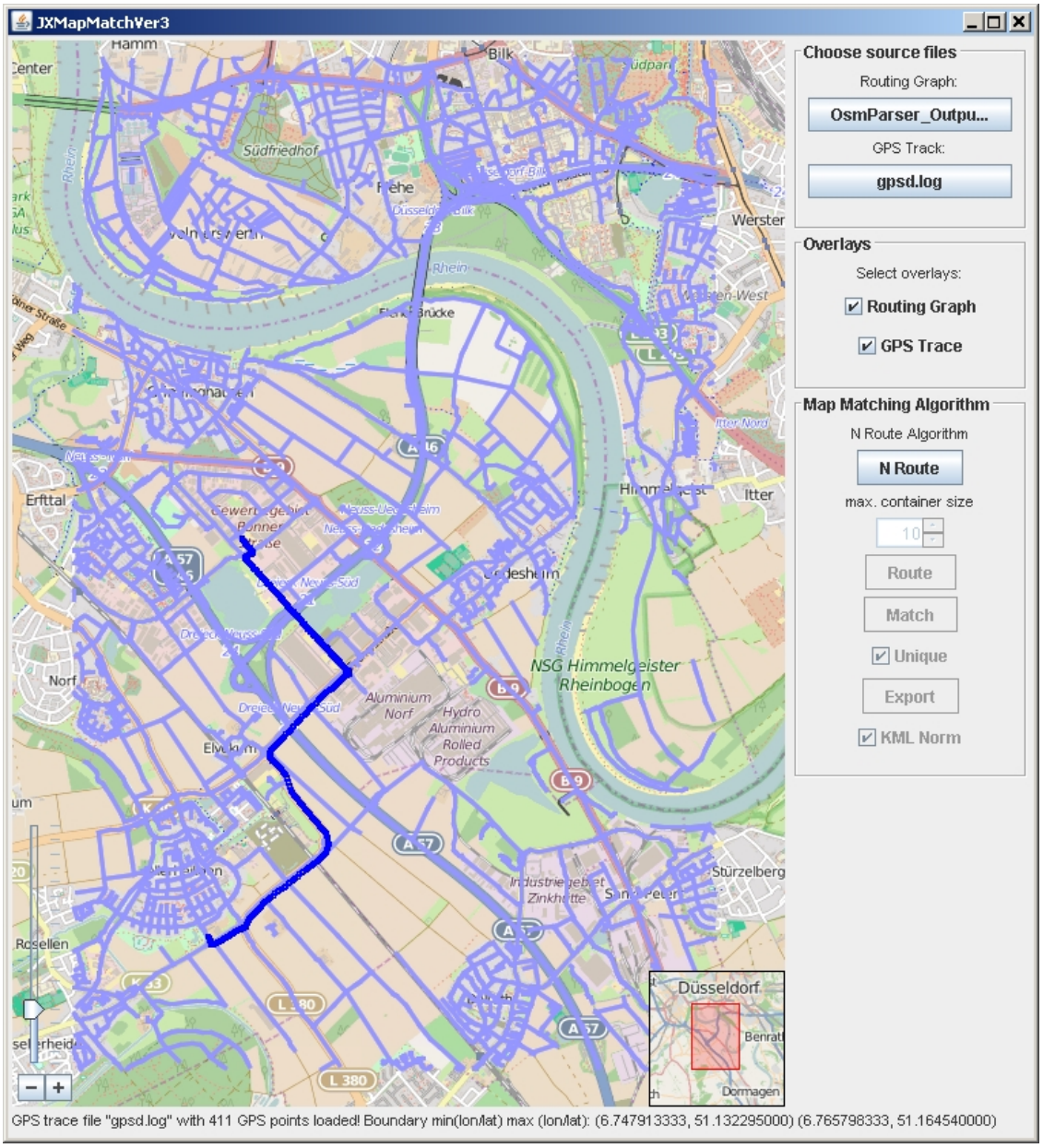

Abbildung 5.1.: Benutzeroberfläche des JXMapMatchVer3

Im vorliegenden Beispiel existiert in jedem Messfahrtordner der Unterordner *JXMapMatchVer3_Export*, in dem die Export-Dateien (*output.csv*) abgespeichert werden.

Hinweis:
Zwischen dem N Route Algorithmus und dem Matchen der GPS Punkte kann die Route

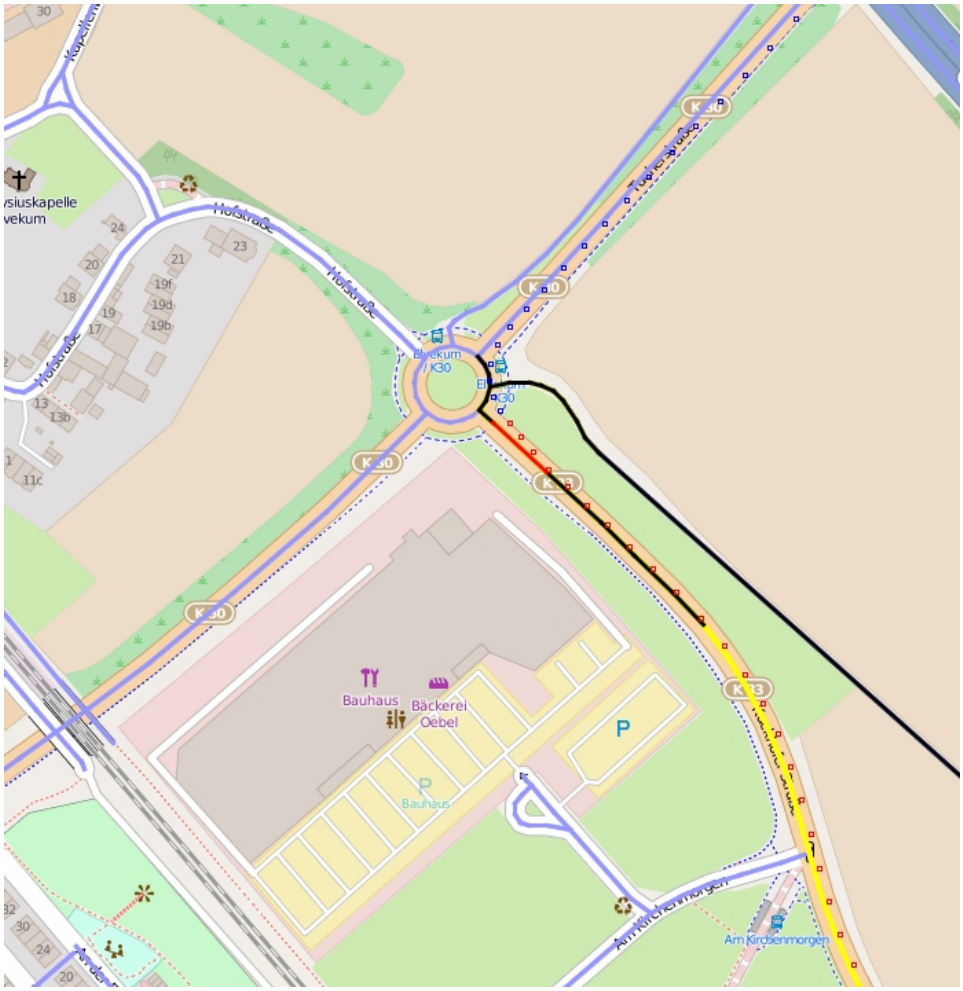

Abbildung 5.2.: N Route Algorithmus in JXMapMatchVer3

korrigiert werden. Es ist zu empfehlen, dass der Anwender ganz nah an die Route heranzoomt und dann die ganze Route auf der Karte durchläuft. Dabei sollte geprüft werden, ob die abgebildete Route auch in jeder Situation dem tatsächlich gefahrenen Weg entspricht. Bei dieser Überprüfung und gegebenenfalls Korrektur ist darauf zu achten, dass nicht versehentlich eine Maustaste gedrückt und somit ein wayPart entfernt oder an eines der beiden Enden der Route hinzufügt wird. Dies kann schnell passieren, da die Karte mit gehaltener linken Maustaste verschoben und durch einen linken Mausklick auf der Karte ein neuer wayPart hinzugefügt wird. Als Konsequenz eines solchen Fehlers kann es vorkommen, dass der erste GPS Punkt nicht auf den richtigen wayPart gemacht wird, da der erste GPS Punkt immer auf den ersten wayPart gematcht wird.

Wie in Kapitel 3.2.3 (JXMapMatchVer3) ausgeführt, werden ergänzend zur primären Ausgabedatei auch weitere Dateien im gleichen Verzeichnis gespeichert. Im Folgenden wird erläutert, welche Informationen die Ausgabe KML Dateien des JXMapMatchVer3 beinhalten und mit Google Earth darstellen.

- ***.osm.kml:** Diese KML Datei stellt den originalen OSM Graphen dar.

- ***.unmatched.kml:** Mit dieser Datei werden die gemessenen GPS Positionen der Messfahrt dargestellt.

- ***.route.kml:** Hierbei handelt es sich um die abgefahrene und auf OSM abgebildete Route als Kantenzug. Wenn eine Kante in weißer Farbe dargestellt wird, dann wurde auf sie kein GPS Punkt des GPS-Trace abgebildet. Dies kann passieren, wenn kein GPS-FIX vorhanden ist, die Kante kurz ist und/oder das Fahrzeug schnell gefahren ist.

- ***.unmatched.matched.kml:** Diese Darstellung ist eine der wichtigsten, denn sie zeigt, wie die echten gemessenen GPS Positionen auf die OSM ways vom JXMapMatchVer3 abgebildet wurden. Es ist zu empfehlen die drei Dateien **.unmatched.matched.kml*, **.route.kml* und **.unmatched.matched.kml* mit Google Earth gleichzeitig zu betrachten, um die Qualität des MapMatching von JXMapMatchVer3 zu kontrollieren. Abbildung 5.3 zeigt beispielhaft die gemessenen GPS Positionen als Pfad (rot - *.unmatched.kml), die OSM Route als Kantenzug (schwarz/weiß - *.route.kml) und das MapMatching der gemessenen GPS Positionen (gelb - .unmatched.matched.kml) in Google Earth.

  Erst diese Visualisierung macht die Qualität des JXMapMatchVer3 und seines Vorgängers (JXMapMatchVer2) vergleichbar. Denn somit ist sehr einfach und zugleich auch schnell gezeigt, wann und wo das Mapmatchen gut oder schlecht funktioniert. Ohne diese Visualisierung können nur einzelne zufällige und/oder kritische GPS Koordinaten manuell untersucht werden. Mit dieser Kontrollmöglichkeit können mit sehr akzeptablem Aufwand jeder einzelne GPS Punkt und seine Abbildung auf den OSM Graphen geprüft werden.

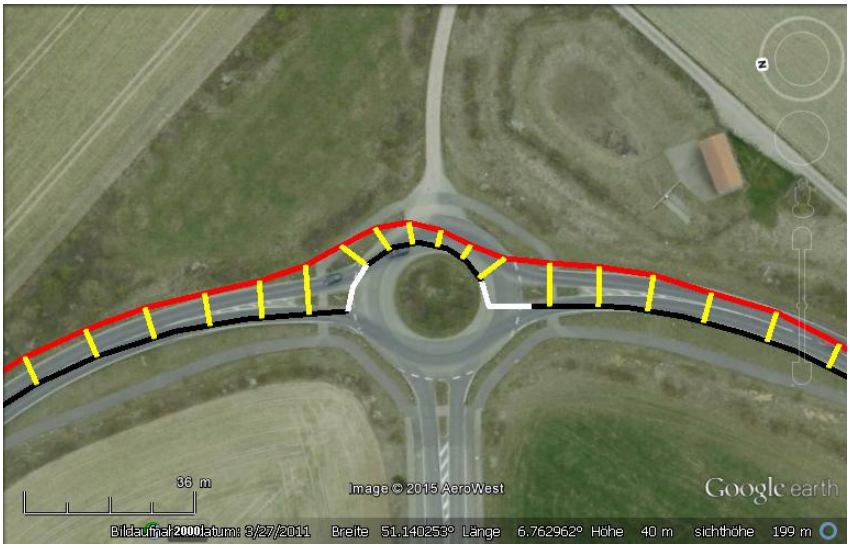

Abbildung 5.3.: GPS Trace (rot), OSM Route (schwarz/weiß), GPS MapMatching (gelb)

- ***.DatasetsDownUnMatched.kml** und ***.DatasetsUpUnMatched.kml:** Diese KML Dateien zeigen nicht das MapMatching der GPS Positionen, sondern das der gematchten Messwerte. Für jede Verbindungsrichtung (Up- / Downstream) ist eine KML Datei vorhanden.
  Da das Messprogramm versucht bis zu fünf Messungen pro Sekunde durchzuführen, existieren viel mehr Messwerte als GPS Positionen in den Log-Dateien. Somit werden auch ways mit Messwerten ausgeleuchtet, die von GPS Punkten nicht erfasst werden.
  Abbildung 5.4 zeigt das MapMatching der Downstream-Messwerte.

- ***.matched.kml:** Diese KML Datei zeigt den Pfad der entsteht, wenn nur die auf den OSM Graphen gematchten GPS Punkte verbunden werden.

- ***.DatasetsDown.kml** und ***.DatasetsUp.kml:** Diese zwei KML Dateien sind das Pendant zu der vorherigen KML Datei, mit dem Unterschied, dass hier nicht die GPS Punkte, sondern die gematchten Messwerte verbunden werden. Auch hier gibt es pro Verbindungsrichtung eine Datei.

- ***.reordered.kml:** Die GPS Daten weisen teilweise eine Genauigkeit von einigen Metern auf. Wenn das Messfahrzeug zum Beispiel an einer roten Ampel steht oder

Abbildung 5.4.: GPS Trace (rot), OSM Route (schwarz/weiß), GPS MapMatching (gelb), Downstream-Messwerte MapMatching (blau)

sehr langsam fährt, kann diese Ungenauigkeit dazu führen, dass GPS Positionen zur Fahrtrichtung nach hinten springen. Um dieses Fehlverhalten auszugleichen wurde eine *reorder-Funktion* entwickelt und implementiert. Die *.reordered.kml Datei zeigt das Ausmaß der Korrektur dieser reorder-Funktion.
In Kapitel 3.3 (Reorder) wird erläutert, wie die reorder-Funktion gematchte GPS Punkte auf der OSM Route verschiebt. Diese KML Datei zeigt für die betroffenen GPS Punkte die Verschiebung auf der OSM Route mit einem gelben Strich an.
Zu beachten ist, dass im optimalen Fall diese Datei keine Informationen enthält, was bedeutet, dass während der Messfahrt kein Problem auftrat und somit ein reorder nicht erforderlich war.

- ***.RouteDistribution.kml:** Es wurden mehrere Ansätze des MapMatchings in Betracht gezogen. Einige Ideen wurden im Vorfeld bei Besprechungen und / oder theoretischen Überlegungen verworfen. Mit dieser Datei wird das Ergebnis eines alternativen MapMatching Algorithmus dargestellt. Im Ansatz geht es bei diesem Versuch des MapMatchings um die Durchführung einer lineare Streckenabbildung (Interpolation) über die ganze Route. Da diese Visualisierung aber zeigt, dass nur in sehr seltenen Fällen ein gutes MapMatching-Ergebnis erzielt wird, wurde eine weitere Entwicklung und Implementierung des Algorithmus nicht weiter verfolgt.

- ***.Color*.kml:** Es werden sechs weitere KML Dateien erstellt. Sie stellen die drei gemessenen Werte pro Verbindungsrichtung farbig dar. Ihre Werte werden für eine maximale Ausnutzung des Farbraums zwischen den Farben grün bis gelb visualisiert. Dabei werden die gemessenen Werte für jede Messfahrt und Verbindungsrichtung normiert und auf die Farbskala interpoliert. Für jede Messfahrt und Verbindungsrichtung werden für die Datenrate der höchste Wert grün und der kleinste Wert gelb dargestellt. Bei der Latenz und der Verlustrate erhalten der kleinste Wert die Farbe grün und der größte die Farbe gelb. Die genauen nummerischen gemessenen Werte sind in den KML Dateien als Beschreibung hinterlegt, so dass der Anwender sie in Google Earth sich anzeigen lassen kann. Abbildung 5.6 zeigt die vollständige Farbskalar an. Auf Abbildung 5.6 ist ein Beispiel der farbigen Visualisierung der Datenrate im downstream zu sehen.

Abbildung 5.5.: Vollständige Farbskala der KML Messwerte Dateien

Abbildung 5.6.: Werte von Datenrate downstream in Farbe

5.4. DataUnion

Als Nächst wird DataUnion genutzt, um alle fünf *output.csv* Dateien des JXMapMatch-Ver3 zu vereinen oder, um die Ergebnisse zu aggregieren. Im Listing 5.4 vereinigt der erste java Aufruf alle Messwerte der fünf Messfahrten und speichert diese in der Datei *output_all.csv* ab.

Listing 5.4: Beispiel OutputTool Parameter

```
1 java DataUnion -o C:\DVD\Daten_Messfahrt\
    DataUnion_Ausgabe\output_all.csv -g all -i C:\DVD\
    Daten_Messfahrt\20150803\090007\
    JXMapMatchVer3_Export\output.csv -i C:\DVD\
    Daten_Messfahrt\20150803\091227\
    JXMapMatchVer3_Export\output.csv -i C:\DVD\
    Daten_Messfahrt\20150803\092532\
    JXMapMatchVer3_Export\output.csv -i C:\DVD\
    Daten_Messfahrt\20150803\093139\
    JXMapMatchVer3_Export\output.csv -i C:\DVD\
    Daten_Messfahrt\20150825\172937\
    JXMapMatchVer3_Export\output.csv
2
3 java DataUnion -o C:\DVD\Daten_Messfahrt\
    DataUnion_Ausgabe\output_ww.csv -g med- -t datarate
     -r ww -i C:\DVD\Daten_Messfahrt\20150803\090007\
    JXMapMatchVer3_Export\output.csv -i C:\DVD\
    Daten_Messfahrt\20150803\091227\
    JXMapMatchVer3_Export\output.csv -i C:\DVD\
    Daten_Messfahrt\20150803\092532\
    JXMapMatchVer3_Export\output.csv -i C:\DVD\
    Daten_Messfahrt\20150803\093139\
    JXMapMatchVer3_Export\output.csv -i C:\DVD\
    Daten_Messfahrt\20150825\172937\
    JXMapMatchVer3_Export\output.csv
```

Der zweite java Aufruf speichert seine Ergebnisse in der Datei *output_ww.csv*. Mit ihm werden die Daten mit folgenden Eigenschaften gefiltert und aggregiert:

- Der Repräsentant wird per Weg-Gewichtung berechnet.
- Wenn es mehrere Gruppen für eine edge gibt, wird die Gruppe (Repräsentant) mit dem kleineren Median ausgesucht.
- Als Auswahlmesswerttyp wird *datarate* herangezogen.

Nun befinden sich die abgebildeten, gematchten und aufgearbeiteten Daten in den zwei Dateien *output_all.csv* und *output_ww.csv*. In einem echten Szenario wäre nur einer der beiden Java Aufrufe nötig. In diesem Beispiel sollten aber beide Möglichkeiten demonstriert werden.
Diese Daten müssen sodann dem Simulator zur Verfügung gestellt werden. Da die Daten im *Character-separated values* (*csv*) Format gespeichert sind, lassen sich diese einfach lesen und/oder in ein anderes Format oder System übertragen.
Eine genau Auflistung und Erklärung aller Möglichkeiten die Daten zusammenzufassen und/oder zu aggregieren kann Kapitel 7 (Daten-Aggregation) entnommen werden.

5.5. OsmJoin

Das Listing 5.5 zeigt den java Aufruf von OsmJoin. Der erste Parameter (*-o*) gibt den Pfad für die Ausgabe OSM Datei an und die weiteren Parameter (*-i*) geben die Pfade zu den OSM Dateien, die der JXMapMatchVer3 aus den jeweiligen gematchten Routen erstellt hat.

Listing 5.5: Beispiel OsmJoin Parameter

```
1 java OsmJoin -o C:\DVD\OSM_Dateien\OsmJoin_Output.osm.
    xml -i C:\DVD\Daten_Messfahrt\20150803\090007\
    JXMapMatchVer3_Export\output.csv.route.osm.xml -i C
    :\DVD\Daten_Messfahrt\20150803\091227\
    JXMapMatchVer3_Export\output.csv.route.osm.xml -i C
    :\DVD\Daten_Messfahrt\20150803\092532\
    JXMapMatchVer3_Export\output.csv.route.osm.xml -i C
    :\DVD\Daten_Messfahrt\20150803\093139\
    JXMapMatchVer3_Export\output.csv.route.osm.xml -i C
    :\DVD\Daten_Messfahrt\20150825\172937\
    JXMapMatchVer3_Export\output.csv.route.osm.xml
```

Somit ist die finale und vom OsmJoin ausgegebene OSM Datei nur ca. 32 KB groß. Auf diese Weise kann dem Simulator eine sehr kleine und damit auch performante OSM Datei angeboten werden, die alle nötigen Informationen beinhaltet, um alle Messwerte zu nutzen.

Die Abbildung 5.7 zeigt den neuen minimalen OSM Graph, der per JXMapMatchVer3 in angepasster Farbe (schwarz) dargestellt wird.

Abbildung 5.7.: OsmJoin Ausgabe: Minimaler OSM Graph

Kapitel 6.

Analyse und Bewertung des MapMatchings

In diesem Kapitel soll das *MapMatching* (*matchen*) und das *Abbilden* der GPS Punkte des GPS Traces auf den OSM Graphen genauer analysiert und bewertet werden.

6.1. MapMatching, matchen, GPS abbilden

Das MapMatching lässt sich leider nicht nur an reinen Zahlen analysieren und bewerten. Entscheidend ist, dass die tatsächlich optimale und beste Route im OSM Graphen gefunden wurde, die den aufgezeichneten GPS Trace möglichst genau darstellt. Dabei ist nicht die Entfernung von jedem einzelnem GPS Punkt zum abgebildeten OSM way entscheidend, sondern die Entfernung aller GPS Punkte zur ihren OSM ways.

6.1.1. Maximale Entfernung der GPS Punkte

Zuerst werden aber einige unterschiedliche GPS Punkte betrachtet, die eine große Entfernung zu ihrem OSM way aufweisen. Diese Informationen werden aus der Datei *out-*

put_GPS.csv entnommen. Hierbei handelt es sich um eine Ausgabedatei des JXMapMatchVer3. Sie beinhaltet ähnliche Daten wie die *output.csv* Datei. Jedoch sind in ihr nicht alle Messwerte enthalten, sondern nur GPS Punkte und der zu diesem Zeitpunkt aktuellste Messwert. Als Auswahlkriterium werden die Daten der Spalte *unMatched_distance* herangezogen. Diese Spalte gibt die Entfernung zwischen dem gemessenen GPS Punkt und dem gematchten Punkt auf dem OSM way an.

Die größte Entfernung mit 19,3675874329045 Metern hat der GPS Punkt mit dem Zeitstempel 1440516809000000000 aus der Messfahrt 20150825 172937. In Abbildung 6.1 werden dieser Punkt und sein MapMatching mit dem blauen Strich angezeigt. Hierbei handelt es sich um eine Autobahnauffahrt. Obwohl die Entfernung fast 20 Meter beträgt, war das MapMatching hier erfolgreich.

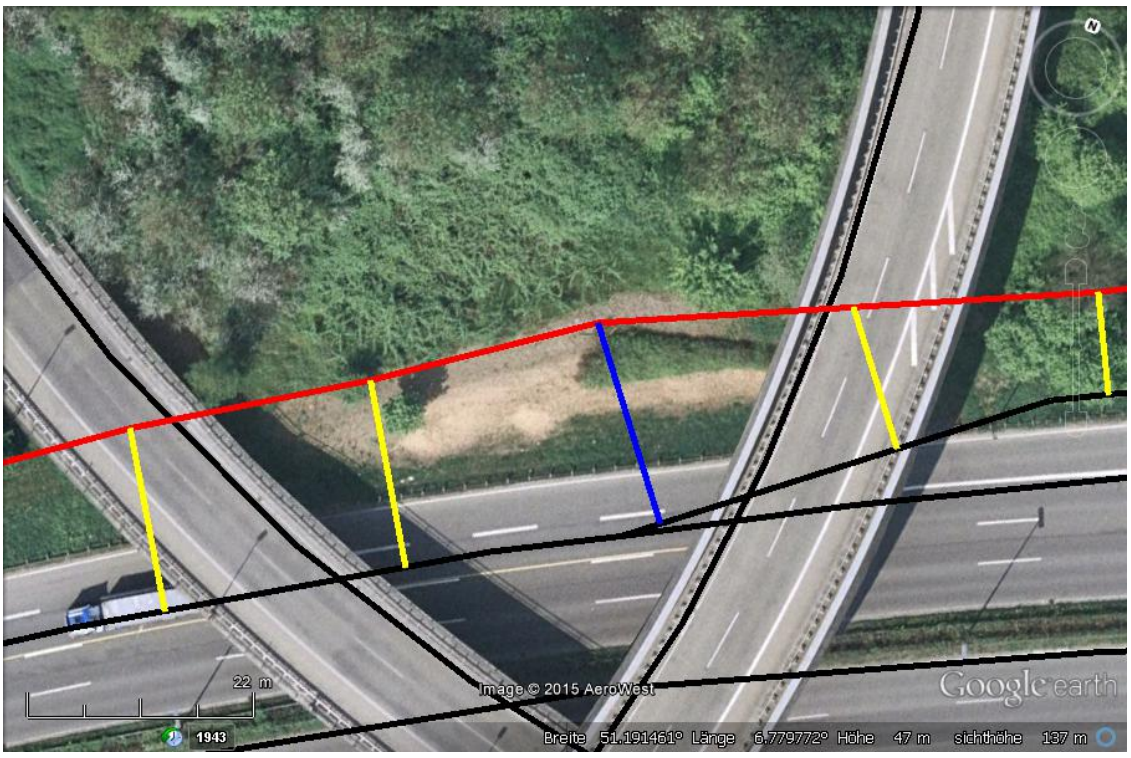

Abbildung 6.1.: MapMatching von GPS Punkt 1440516809000000000

Das nächste Beispiel zeigt in Abbildung 6.2 zwei GPS Punkte auf der Forumstraße in Neuss. Hierbei handelt es sich um die zwei Punkte mit der größten MapMatching Entfernung, die sich nicht auf der Autobahn befinden. Die GPS Punkte haben den Zeit-

stempel 1438587329000000000 (blau) und 1438587330000000000 (grün), die Entfernung 14,65242489 Meter und 14,54546567 und wurden auf der Messfahrt *20150803 093139* aufgezeichnet. Auch hier ist schnell ersichtlich, dass das MapMatching erfolgreich war.

Abbildung 6.2.: MapMatching von GPS Punkt 1438587329000000000

Diese zwei hier dargestellten sowie viele weitere untersuchte Beispiele zeigen, dass die Entfernung vom GPS Punkt zu keiner eindeutigen Aussage über die Qualität des MapMatching führt.

6.1.2. Besondere Beispiele fürs MapMatching

Weiter sollen zwei spezielle Fälle vorgestellt werden. In den dazugehörigen Abbildungen sind der OSM Graph hellblau, der GPS Trace der Messfahrt rot, die gefundene Route schwarz und das MapMatching der GPS Punkte gelb dargestellt.

Abbildung 6.3 zeigt einen kleinen Ausschnitt einer Test-Messfahrt von Düsseldorf Benrath nach Solingen Mitte. Die gefundene optimale Route wurde vom N Route Algorithmus mit N = 10 berechnet. Der Ausschnitt zeigt eine Fahrt, bei der mit Absicht ein Umweg über einen Parkplatz gefahren wurde. Sogar in diesem Fall hat der N Route Algorithmus die optimale Route eigenständig ermittelt.

Abbildung 6.3.: MapMatching mit Umweg

Das nächste Beispiel zeigt eine Situation, bei der der N Route Algorithmus nicht den tatsächlich gefahrenen Weg ermittelt hat. Für diese Situationen hat der Anwender im JXMapMatchVer3 die Option zwischen dem N Route Algorithmus und dem Matchen der GPS Punkte, die Route manuell zu korrigieren. Abbildung 6.4 zeigt die vom N Route Algorithmus berechnete und gefundene Route; Abbildung 6.5 zeigt die vom Anwender korrigierte Route.

Die Tatsache, dass hier der JXMapMatchVer3 den tatsächlichen gefahrenen Weg im OSM Graphen nicht gefunden hat, erklärt sich wie folgt:
Die Ursache ist auf die fünf GPS Punkte, die unterschiedlich gematcht wurden, zurück-

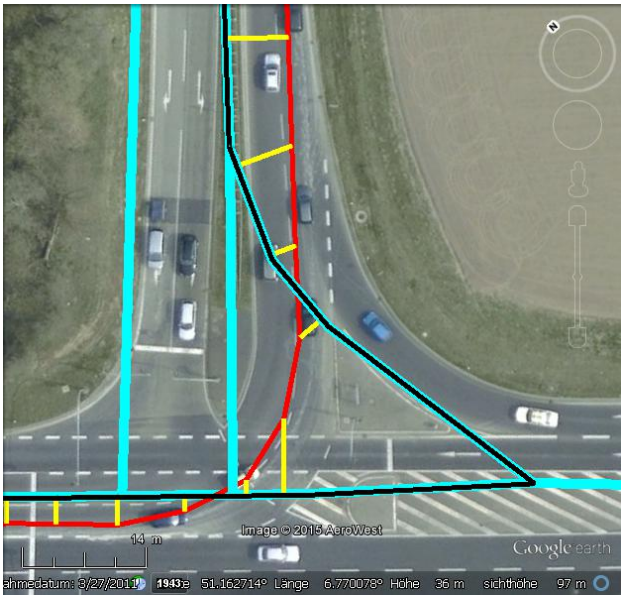

Abbildung 6.4.: MapMatching ohne Korrektur

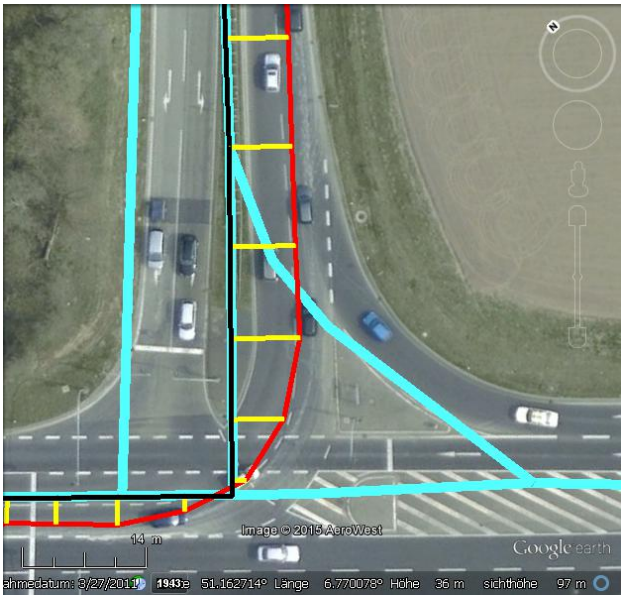

Abbildung 6.5.: MapMatching mit Korrektur

zuführen. Der JXMapMatchVer3 sucht eine Route, die zu allen GPS Punkten den minimalen Abstand hat. Die Teststrecke verläuft von unten links nach oben. Die Tabelle 6.1 zeigt die Entfernung der fünf GPS Punkte und ihre Entfernungen zur Route, die unterschiedlich auf die zwei Routen gematcht wurden.

Tabelle 6.1.: Entfernung GPS Punkte zur (korrigierten) Route

GPS Punkt (Zeitstempel)	Entfernung zur Route ohne Korrektur (m)	Entfernung zur Route mit Korrektur (m)
1435819228000000000	1.60238932140246	1.49272691230408
1435819229000000000	8.36332407911526	5.81015310613499
1435819230000000000	3.25414000381409	7.75923433896539
1435819231000000000	3.05403042461811	7.38942223528964
1435819231000000000	3.05403042461811	7.14974111599966
Summe	19.3279142535680	29.6012777086938

Der Tabelle ist zu entnehmen, dass die ersten zwei GPS Punkte näher an der korrigierten Route liegen. Jedoch sind die restlichen GPS Punkte deutlich näher an der *falschen* Route. Somit ist die Summe der Entfernungen der GPS Punkte zu der falschen / nicht korrigierten Route kürzer. Dies zeigt, dass der N Route Algorithmus nicht die erstbeste Route, sondern die *mathematisch bessere* Route bevorzugt.
In diesem konkreten Beispiel hat der JXMapMatchVer3 die falsche Route ausgewählt. Dies ist aber mathematisch betrachtet unter Berücksichtigung aller Entfernungen der GPS Punkte die bessere Route. Diese Eigenschaft hat in diesem Einzelfall dafür gesorgt, dass nicht die tatsächliche, gefahrene Route gewählt wurde. Dem gegenüber stehen sehr viele andere Situationen, in denen genau diese Eigenschaft der Routenbestimmung zu einer besseren Routenwahl führen.

6.1.3. Projektion: MapMachting in Abbiegungen

In diesem Kapitel soll noch einmal das MapMatching von GPS Punkten und Messwerten in Kurven, Abbiegungen und Kreuzungen genauer beleuchtet werden. Den Abbildungen 6.6 und 6.7 kann entnommen werden, dass die GPS Punkte (gelb) und somit auch die Messwerte (blau) leider linear verzerrt auf die OSM Route gematcht werden.

Die Ursache liegt darin, dass die gefahrene Route und die OSM Route teilweise um mehrere Meter versetzt sind. Die Gründe hierfür wurden bereits in Kapitel 3.2 (Aufgaben, Probleme, Lösungen und Grundkonzeption) erörtert. In der aktuellen Lösung werden die Messwerte mit Hilfe der Zeitstempel linear zwischen den GPS Punkten auf die gefahrene

Abbildung 6.6.: Innen-MapMatching in Abbiegungen

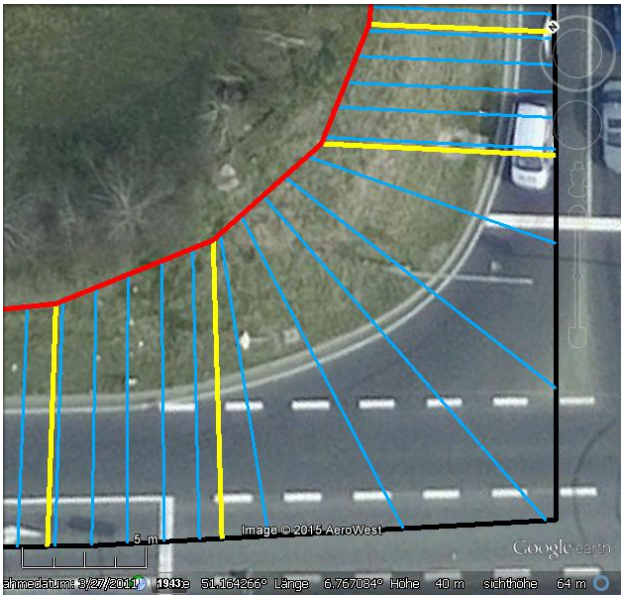

Abbildung 6.7.: Außen-MapMatching in Abbiegungen

Route verteilt. Anschließend werden sie linear zwischen die gematchten GPS Punkte auf die OSM Route abgebildet. Hierdurch kann es passieren, dass mehrere Messwerte von einem längeren Routenabschnitt auf einen sehr kurzen OSM Routenabschnitt abgebildet werden. Abbildung 6.6 kann sogar entnommen werden, dass mehrere Messwerte auf

einen Punkt abgebildet werden können. Dies führt zum Verlust von Messwerten in der Simulation, denn für einen Punkt auf einem OSM way und somit auf einem Punkt einer edge kann es nur einen Messwert geben. Es wäre durchaus möglich Messwerte, die auf einen Punkt abgebildet werden, zu aggregieren, jedoch würde dies die Messwerte glätten und/oder verfälschen.
Wenn die OSM Route außen in der Abbiegung liegt (siehe Abbildung 6.7), dann werden die GPS Punkte und Messwerte über einen teilweise viel größeren Routenabschnitt verteilt beziehungsweise projiziert.

Um in solchen Situationen eine *bessere* Lösung des MapMatching der Messwerte zu erzielen, müssten zuvor die GPS Punkte *besser* auf die OSM Route abgebildet werden.
Ein möglicher Lösungsansatz wäre es, nicht nur die Messwerte linear zwischen zwei GPS Punkten zu interpolieren, sondern in diesen *besonderen* Situationen alle GPS Punkte und Messwerte linear über die ganze Strecke der besagten Situationen zu interpolieren. Hierfür müssten diese *besonderen* Situationen automatisch und zuverlässig erkannt werden. Ein Lösungsansatz für dieses Problem könnte darin bestehen, die Kurven und Abbiegungen mit Hilfe des Winkels von zwei benachbarten ways zu untersuchen. Dieser Winkel könnte mit der Fahrtrichtung *verglichen* und *bewertet* werden, um dann Kurven, Abbiegungen und Kreuzungen mit solchen ungünstigen Situationen zu erkennen. Dieser Lösungsansatz könnte in den beschriebenen Situationen ein noch besseres MapMachting erlauben. Diese Idee wurde auch zum Teil untersucht. Ein *lineares Strecken-MapMatching* (/interpolieren) über die ganze Route zeigt die KML Datei **.RouteDistribution.kml* (siehe Kapitel 3.2.3 - JXMapMatchVer3). Versuche haben gezeigt, dass dieses *Strecken MapMatching*, nicht über die ganze Route, sondern nur in bestimmten Situationen (zum Beispiel: Abbiegungen) ein besseres Ergebnis liefert.
Abbildung 6.8 zeigt, wie Messwerte nach dem aktuellen Lösungsansatz auf eine Abbiegung ungünstig gematcht werden. Abbildung 6.9 zeigt, wie das MapMatching der neuen theoretischen Lösung nur auf diese eine Situation aussehen würde. Abbildung 6.9 zeigt, wie das neue MapMatching auf die ganze Route und nicht nur auf kritische Situationen aussieht.

Zusammenfassend kann festgehalten werden, dass eine Erkennung von kritischen Situationen und Erfassung ihrer Größe von großer Relevanz sind.

Abbildung 6.8.: MapMatching in Abbiegungen

Abbildung 6.9.: Neues MapMatching in Abbiegungen

Eine solche automatische Erkennung wurde für zu komplex beurteilt, um sie als ergänzende *Teilaufgabe* im Rahmen dieser Masterarbeit anzugehen. Abbildung 6.11 zeigt, wie schnell sich die Winkel der ways und die Fahrtrichtung unterscheiden oder zusammenfallen. Dies veranschaulicht nochmals, dass die Umsetzung dieses theoretischen Lösungs-

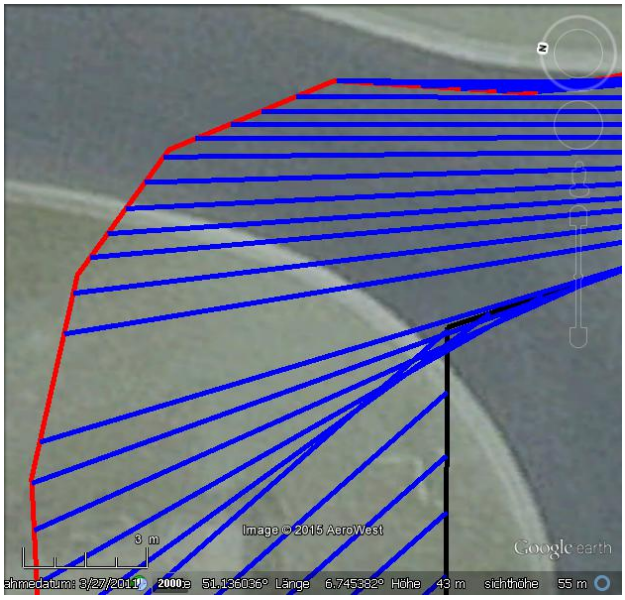

Abbildung 6.10.: Neues MapMatching über ganze Route

ansatzes sehr komplex und umfangreich ist. Selbst eine Aufwandsabschätzung war nicht ohne Weiteres möglich. Da die vorhandene Lösung brauchbare bis gut und sehr gute Ergebnisse liefert, wurde dieser Lösungsansatz unter Berücksichtigung eines Kosten-Nutzen-Aspektes nicht weiter verfolgt.

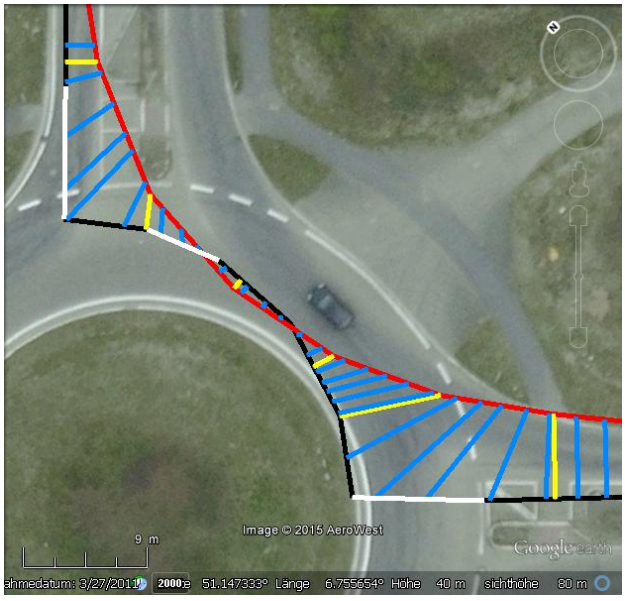

Abbildung 6.11.: MapMatching in Abbiegungen im Kreisverkehr

Eine mögliche Teil-Identifikation der oben beschriebenen kritischen Situationen erlaubt das MapMatching von GPS Positionen. Wie auf Abbildung 6.6 zu sehen ist, werden GPS Positionen in Außen-Abbiegungen gegebenenfalls auf einen Punkt der OSM Route abgebildet. Hierdurch werden auch alle Messwerte, die zwischen diesen GPS Punkten liegen, ebenfalls auf diesen einen Punkt abgebildet und gehen somit in der Praxis bei der Simulation verloren. Untersuchungen haben gezeigt, dass GPS Positionen, die auf den gleichen Punkt abgebildet werden, bei dem Abbilden der Messwerte auf die OSM Route übersprungen werden können. Die Messwerte werden dann linear zwischen zwei eindeutig gematchte GPS Positionen interpoliert. Diese Funktion wurde in JXMapMatchVer3 integriert und kann aber wieder vom Anwender deaktiviert werden. Die Deaktivierung kann durch die CheckBox *Unique GPS* unter dem *Match* Button durchgeführt werden.

Kapitel 7.

Daten-Aggregation

In diesem Kapitel werden die Messdaten (datarate, delay und loss_rate), ihre Auswahl vom Simulator und die Notwendigkeit ihrer Aggregation erläutert.
Im Idealfall wird während einer Messfahrt alle 200 ms ein Messdatensatz pro Verbindungsrichtung gemessen. Ein Messwert besteht aus den Messdaten datarate, delay und loss_rate, seiner GPS Position beziehungsweise der Längenposition auf einer way und einem Zeitstempel. Diese Messdaten wurden mit Hilfe des JXMapMatchVer3 auf edges abgebildet, die den Straßengraphen im Simulator nachbilden. In der OSM Definition gibt es kein maximale Entfernung zwischen zwei nodes, die einen wayPart bilden. Somit kann eine edge einen geraden Straßenabschnitt von beliebiger Länge darstellen. Daher werden auf die allermeisten edges einige bis viele Messdaten abgebildet. Da das Messfahrzeug während der Messprozedur fährt, werden die Messdaten in der Regel auf unterschiedliche Längenpositionen der edges abgebildet. Die Menge der Messdaten, die entsteht, wenn das Fahrzeug über den Straßenabschnitt einer edge fährt, werden in einer **Gruppe** zusammengefasst. Wenn das Fahrzeug über diesen Straßenabschnitt mehrfach fährt, entsteht für jede weitere Überfahrt eine neue Gruppe. Die Identifikation jeder Gruppe ist eine Nummer. Der dazugehörige Spaltenname in den csv-Dateien lautet **matchedLinkNr**.

Während der Simulation befindet sich ein simuliertes Fahrzeug zu jedem Zeitpunkt auf einem Punkt einer edge und somit auf einer Längenposition einer edge. Der Simulator

benötigt die Messwerte, um ortsbezogen die gemessenen Netzwerkcharakteristiken für das Fahrzeug zu simulieren.

Die Übernahme von Messwerten, die sich auf der gleichen edge wie das Fahrzeug befinden und den kleinsten Abstand zum Fahrzeug haben, stellt eine Möglichkeit für die Auswahl der Messwerte dar. Daraus erwächst der Nachteil, dass ein Fahrzeug sehr häufig Messwerte aus verschiedenen Gruppen erhält. Hier ein Beispiel, wie sich dieser Nachteil auswirken kann:
Eine edge ist 10 Meter lang und hat 2 Gruppen. Die erste Gruppe hat Messwerte, die auf die geraden Meterzahlen gematcht wurden; die Messwerte der zweiten Gruppe wurden auf die ungeraden Meter gematcht. Beim Durchfahren dieser edge wechseln die übernommenen Netzwerkcharakteristiken zehn mal die Gruppe, da nach jedem Meter ein Messwert der anderen Gruppe näher am Fahrzeug ist.

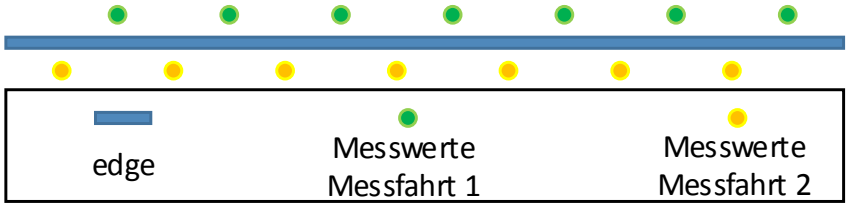

Abbildung 7.1.: Oszillieren von Messwertgruppen auf einem way

Dieses schnelle Oszillieren zwischen den Gruppen auf einer edge wird in Abbildung 7.1 skizziert und hat zur Folge, dass die simulierten Netzwerkcharakteristiken bei der Simulation unrealistisch nachgestellt werden.

Ein realistisches und somit besseres Zugriffsmuster auf die Messwerte wurde von Raphael Bialon vorgeschlagen und im Simulator implementiert:
Ein Fahrzeug sucht die Gruppe, die demselben way zugeordnet ist und die kleinste zeitliche Differenz zur aktuellen Simulationszeit aufweist. Im Anschluss wird ein Messwert in der Gruppe ermittelt, der von der Längenposition des Fahrzeugs die kleinste Entfernung hat.
Allerdings birgt auch dieses Vorgehen Nachteile:

- Erstens kann es vorkommen, dass ein Fahrzeug mitten auf einer edge die Gruppe wechselt, da die Simulationszeit vorwärts läuft.

- Zweitens hat der Anwender keine Kontrolle darüber, welche der zur Verfügung gestellten Gruppen und somit Messwerte vom Simulator angewendet werden. Er kann lediglich die Anfangssimulationszeit angeben, jedoch keinen Einfluss auf die Art der verwendeten Messwerte - ob zum Beispiel Messwerte mit einer hohen Datenrate oder großer Latenz benutzt werden - nehmen.

Um dem Anwender mehr Einflussnahme über die simulierten Netzwerkcharakteristiken zu geben, wurde die Möglichkeit der Datenaggregation erwogen. Hierbei kann der Anwender vor der Simulation entscheiden, welche Eigenschaften die Messwerte haben sollen, auf die der Simulator zurückgreift. Gleichzeitig beruhen die simulierten Netzwerkeigenschaften auf echten gemessenen Daten und sind weder berechnet, approximiert noch geglättet. Die oben genannten Nachteile werden vermieden, wenn dem Simulator für jede edge nur eine Gruppe angeboten wird. Zur Verwirklichung dieser Lösung wurde das Programm DataAggregation herangezogen.
Wie bereits in Kapitel 3 (Funktionsweise und Algorithmen) beschrieben, findet die Auswahl der Gruppe mit den gewünschten Eigenschaften der Messdaten in zwei Schritten statt. Im ersten Schritt wird ein Repräsentant pro Gruppe berechnet, und im zweiten Schritt wird für jede edge mit Hilfe des Repräsentanten eine Gruppe ausgewählt.

7.1. Repräsentanten ermitteln

Bei dieser Aufgabe soll ein Wert gefunden werden, der die ganze Gruppe von Messwerten vertritt. Für die Ermittlung dieses Wertes wurden acht Berechnungsmöglichkeiten als sinnvoll betrachtet und implementiert. Die Auswahl folgt auf den Parameter *-r* und wird dem Programm beim Starten übergeben:

- **min:** Es wird der kleinste Wert aus der Gruppe ausgewählt.
- **max:** Es wird der größte Wert aus der Gruppe ausgewählt.
- **avg:** Es wird das arithmetische Mittel berechnet und übernommen.
- **med:** Es wird der Median berechnet und übernommen.

- **minstde:** Vom arithmetischen Mittel wird die Standardabweichung subtrahiert. Das Ergebnis wird als Repräsentant verwendet. (Falls dieser Wert kleiner als der min-Wert ist, dann wird der min-Wert übernommen.)

- **maxstde:** Zum arithmetischen Mittel wird die Standardabweichung addiert. Das Ergebnis wird als Repräsentant verwendet. (Falls dieser Wert größer als der max-Wert ist, dann wird der max-Wert übernommen.)

- **avgstde:** Es wird das arithmetische Mittel aller Werte, die innerhalb der minstde- und maxnstde-Grenzen liegen, berechnet und übernommen. Dies ist ein arithmetisches Mittel, bei dem die *Ausreiser-Werte* nicht berücksichtigt werden.

- **ww:** Es wird ein Wert nach Weg-Gewichtung (**w**ay **w**eighted) berechnet. Diese Berechnungsart des Repräsentanten soll die Messwertauswahl, die der Simulator durchführt, nachahmen. Dies wird an einem kleinen Beispiel erklärt:
  Eine edge hat die Länge L = 10 und an den Längenpositionen l_1 = 2, l_2 = 4 und l_3 = 7 die Werte w_1 = 11, w_2 = 12 und w_3 = 13. Der erste Wert (11) gilt für den edge-Längenabschnitt von 0 bis 3. Der zweite Wert (12) gilt für den edge-Längenabschnitt von 3 bis 5,5 und der dritte Wert (13) für den edge-Längenabschnitt von 5,5 bis 10. Somit hat der erste Wert eine Gewichtung von 3, der zweite Werte eine Gewichtung von 2,5 und der dritte Wert eine Gewichtung von 4,5. Für den Repräsentanten (r) ergibt sich der folgende Wert:

$$r = w_1 * \frac{l_1 + \frac{l_2 - l_1}{2}}{L} + \sum_{i=2}^{n-1} \left(w_i * \frac{(l_i + \frac{l_{i+1} - l_i}{2}) - (l_{i-1} + \frac{l_i - l_{i-1}}{2})}{L} \right) + w_n * \frac{L - (l_{n-1} + \frac{l_n - l_{n-1}}{2})}{L}$$

$$r = 11 * \frac{3}{10} + 12 * \frac{2{,}5}{10} + 13 * \frac{4{,}5}{10}$$

$$r = \frac{33}{10} + \frac{30}{10} + \frac{58{,}5}{10}$$

$$r = \frac{33+30+58{,}5}{10}$$

$$r = \frac{121{,}5}{10} = 12{,}15$$

Eine eindeutige und immer passende Empfehlung, wie die Repräsentantenauswahl einer Gruppe bestimmt und berechnet werden soll, kann nicht ausgesprochen werden. Denn für jede Möglichkeit gibt es gute und schlechte Einsatzszenarien.
Der Simulator sucht auf einer edge den Messwert aus, der am nächsten an der Simulationsposition liegt. Da alle Positionen auf einer egde die gleiche Wahrscheinlichkeit für die Simulationsposition haben (Gleichverteilung), bildet die Weg-Gewichtung (ww) die Auswahl des Simulators eines Messwertes aus einer Gruppe am besten nach. Wenn es keine (anderen) Kriterien oder Bedingungen für die Simulation gibt, empfiehlt es sich die Weg-Gewichtung aus den eben geschilderten Eigenschaften zu nutzen.

7.2. Gruppenauswahl

Eine edge kann mehrere Gruppen haben und jede Gruppe hat einen Wert als Repräsentanten. Als Nächstes wird entschieden, welcher Repräsentant und somit welche Gruppe ausgesucht und für die Ausgabedatei übernommen werden soll. Auch diese Auswahl ist parametrisierbar (*-g*) und hat mehrere Möglichkeiten:

- **all:** Es findet keine Filterung oder Auswahl statt und es werden alle Gruppen übernommen (vergleiche Kapitel 3.2.8.2 - Zusammenfassen von Messwerten).

- **med-:** Es wird der Median ausgewählt. Wenn es eine gerade Anzahl von Repräsentanten/Gruppen gibt, so wird die Gruppe mit dem kleineren der beiden mittleren Repräsentanten gewählt.

- **med+:** Es wird der Median ausgewählt. Wenn es eine gerade Anzahl von Repräsentanten/Gruppen gibt, so wird die Gruppe mit dem größeren der beiden mittleren Repräsentanten gewählt.

- **0 - 1:** Hier wird eine Zahl (p) zwischen 0 und 1 erwartet. Die Grenzen 0 und 1 sind auch erlaubt. Aus dieser Zahl wird ein Wert, der zwischen den kleinsten und größten Repräsentanten liegt, berechnet (m). Für diese Berechnung wird eine lineare

Abbildung durchgeführt. Sodann werden der Repräsentant und seine Gruppe ausgesucht, die dem berechneten Wert (m) am nächsten sind.
Zum besseren Verständnis folgt ein kleines Beispiel:
Gegeben sind sieben Repräsentanten: 11, 12, 12.5, 12.8, 13, 13.5 und 15.2 . Als Parameter-Wert wurde p = 0.33 eingegeben. Zuerst wird m berechnet:

$$m = min + p * (max - min)$$

$$m = 11 + 0.33 * (15.2 - 11)$$
$$m = 11 + 0.33 * 4.2$$
$$m = 11 + 1.386$$
$$m = 12.386$$

In diesem Beispiel wird die Gruppe mit dem Repräsentant von 12.5 ausgesucht, da dieser dem Repräsentant m = 12.386 am nächsten kommt.

Dieses Auswahlverfahren hat den Vorteil, dass der kleinste Repräsentant mit p = 0 und der größten mit p = 1 ermitteln werden kann. Des Weiteren ermöglicht dieses Verfahren sehr flexibel jeden anderen Repräsentant zu finden.

netconvert erstellt für den Simulator Graphen für jede Fahrtrichtung einer Straße (way) eine eigene edge. DataUnion berücksichtigt für die Gruppenauswahl per *default* nur Gruppen, deren Messwerte auf der edge gemessen wurden, als das Messfahrzeug in die gleiche Richtung gefahren ist, wie die edge verläuft. Nur wenn für eine edge keine Messwerte für die gleiche Richtung vorhanden sind, werden auch Gruppen mit Messwerten berücksichtigt, die auf der Gegenfahrbahn gemessen wurden. Diese Eigenschaft heißt *PreferReal* und kann mit dem Parameter *-pr no* deaktivert werden. Bei Deaktivierung werden für die Gruppenauswahl alle Gruppen berücksichtigt, unabhängig davon, ob die Messwerte in der Hin- oder auf der Rückfahrtrichtung einer edge gemessen wurden.

7.3. Analyse des Repräsentanten

Ergänzend wurden Möglichkeiten der automatischen Auswahl der Berechnungsart des Repräsentanten untersucht. Als potenzielles Kriterium für das Auswahlverfahren wurden die Messwerte selbst und Ihre Anzahl in einer Gruppe betrachtet.
Die erste Option, die Messwerte als Kriterium heranzuziehen, wurde verworfen, da für die Entscheidungsfindung bereits die Antwort benötigt wird. Die Antwort auf die Frage, welche Berechnungsart für den Repräsentanten genutzt werden soll, verlangt bereits die Lösung; Aufgabe und Lösung sind zirkulär: Wie kann die Berechnungsart des Repräsentanten automatisch aus den Messwerten selbst ermittelt werden? Um die Berechnungsart zu bestimmen, wird ein Repräsentant benötigt.
Die Automatisierung des Auswahlprozesses der Berechnungsart des Repräsentanten war nicht zielführend. Zwecks Absicherung wurden dennoch einige Auswertungen durchgeführt. Die Abbildungen 7.2 und 7.3 zeigen zwei beispielhafte aus mehreren Untersuchungen für den Messwerttyp *datarate*.

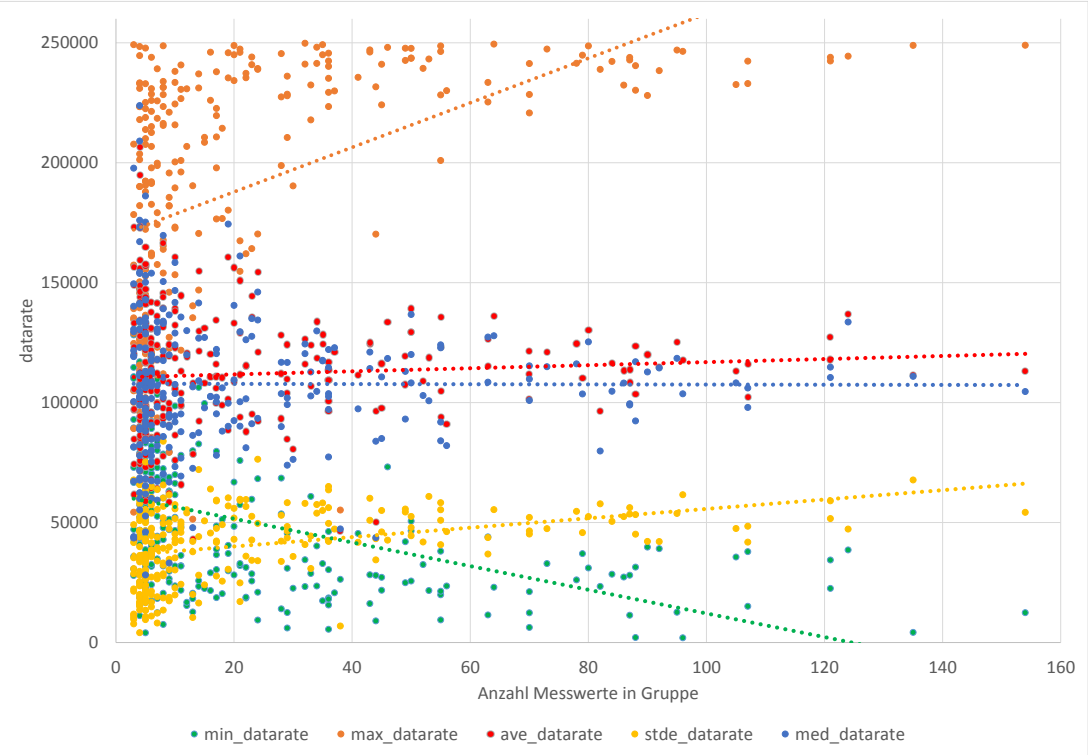

Abbildung 7.2.: datarate Repräsentanten zu Gruppengröße

In Abbildung 7.2 sind verschiedene Repräsentanten (Minimum, Maximum, arithmetisches Mittel und Median) sowie die Standardabweichung mit den dazugehörigen Trendlinien dargestellt. Die X-Achse gibt die Größe der Gruppen an. Dem Diagramm ist zu entnehmen, dass mit der Größe der Gruppe die Wahrscheinlicher für ein kleineres Minimum und ein größeres Maximum ansteigt. Dies kann logisch abgeleitet werden, denn je mehr Messwerte in einer Gruppe vorhanden sind, desto höher ist die Wahrscheinlichkeit, dass besonders hohe oder niedrige *Ausreißer-Messwerte* vorkommen.

Die Abbildung 7.3 baut auf denselben Messwerten auf. Jedoch wurden dies mal für jede Gruppe das arithmetische Mittel, der Median und die Standardabweichung im Verhältnis zwischen Minimum und Maximum normiert und sodann auf der Y-Achse prozentual dargestellt. Die X-Achse gibt erneut die Größe der Gruppe an.
Für den Wert x aus einer Gruppe wird der normierte und prozentuale Wert np durch

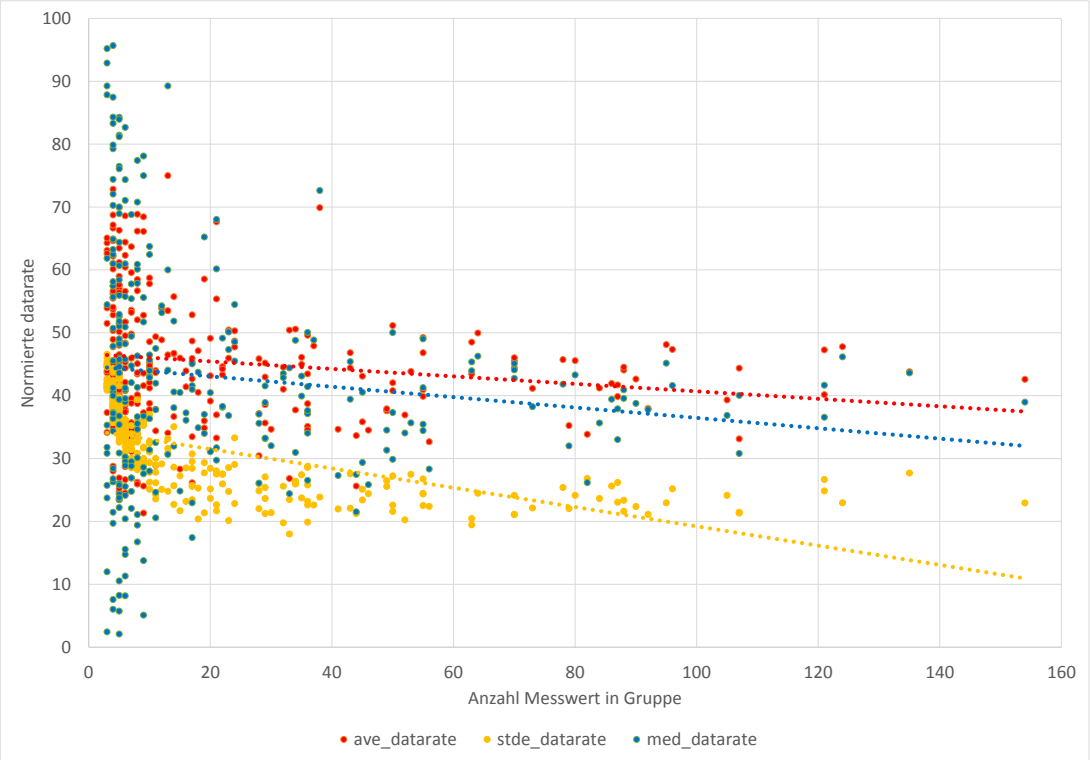

Abbildung 7.3.: datarate Repräsentanten zu Gruppengröße - normiert und prozentual

folgende Formel berechnet:

$$np = \frac{(x-min)}{max-min} * 100$$

Es ist dieser Information zu entnehmen, dass bei kleinen Gruppen die Standardabweichung größer ist als bei großen Gruppen. Dies ist auch nachvollziehbar, denn wenn eine Gruppe nur wenige Messwerte hat, steigt die Standardabweichung schnell an, wenn es in dieser Gruppe einen oder zwei *Ausreißer-Messwert* gibt. Dies bestätigt auch die zum Teil sehr kleinen und großen Mediane und arithmetischen Mittel der kleinen Gruppen. Das Verfahren erlaubt aber keine Auskunft darüber, welcher Repräsentantenberechnungsart der Vorzug zu geben ist.

Kapitel 8.

Zusammenfassung

Im Rahmen dieser Masterarbeit wurden vier Programme entworfen, implementiert und erweitert. Das Ziel dieser vier Programme und ihrer Algorithmen ist es während einer Autofahrt gemessene Mobilfunkcharakteristiken so weit aufzuarbeiten, dass sie anschließend als Basis für die Mobilfunksimulation eines V2X-Applikationsimulators genutzt werden können. Die gemessenen Mobilfunkcharakteristiken (im Folgenden Messwerte genannt) werden zusammen mit Modemverbindungsinformationen und einer GPS Positionsaufzeichnung während einer Messfahrt protokolliert und als Datenbasis dieser Masterarbeit zur Verfügung gestellt. Damit die Simulationslösung diese Daten einlesen und verarbeiten kann, müssen diese mehrfach zeit- und positionsbasiert abgebildet und interpoliert werden, so dass nach der Anwendung aller vier Programme jeder gemessene Messwert auf genau eine Position des Straßengraphen des Simulators abgebildet wird. Des Weiteren hat der Anwender die Möglichkeit die Messwerte von mehreren Messfahrten zu vereinen. Innerhalb dieser vereinten Messwertmenge können die Messwerte auch unter Berücksichtigung des Straßengraphen des Simulators aggregiert werden. Dies ermöglicht dem Anwender Einfluss auf die Mobilfunkcharakteristik einer Simulation zu nehmen. Gleichzeitig wird trotz des Aggregierens auf echte in der Natur beobachtete Messwerte zurückgegriffen, ohne dass dabei die Messwerte berechnet, manipuliert oder geglättet werden, was die Aussagekraft und Qualität der Messwerte und somit auch der Simulation nicht beeinträchtigt.

Während der Bearbeitungszeit dieser Abschlussarbeit haben sich folgende Aufgaben und Herausforderungen als wichtig für die finale Lösung herausgestellt:

Der Simulator erstellt aus einem OpenStreetMap (OSM) Straßengraphen einen eigenen optimierten Graphen, der für die simulierten Fahrzeugbewegungen herangezogen wird. Hierbei ist es erforderlich, dass der OSM Graph und der Simulator Graph bijektiv abbildbar sind, damit die Messwerte nicht nur auf den OSM Graphen, sondern auch auf den Simulator Graphen, eindeutig positioniert werden. Diese Aufgabe übernimmt das entworfene und implementierte Programm OsmParser. Es bereitet die Daten einer OSM Datei unter anderem so weit auf, dass die späteren internen Optimierungen des Simulators für den eigenen Graphen eine bijektive Abbildung nicht verhindern.

Bevor die Messwerte auf den Graphen des Simulators positioniert werden, müssen sie auf den OSM Graphen abgebildet werden. Hierfür wird zuerst der Kantenpfad im OSM Graphen (im Folgenden Route genannt) berechnet, der den tatsächlich gefahrenen Weg darstellt. Dieses *Offline MapMatching* wird mit Hilfe der aufgezeichneten GPS Positionen und dem Programm JXMapMatchVer3 durchgeführt. JXMapMatchVer3 ist eine Fortsetzung des JXMapMatchVer2, der am Lehrstuhl für Rechnernetze und Kommunikationssysteme der Universität Düsseldorf entwickelt wurde. Es wurden wichtige Datenstrukturen und Algorithmen erweitert, ausgetauscht und hinzugefügt. Der implementierte Offline MapMatching Algorithmus hat die Bezeichnung N Route MapMatching und ist eine angepasste Version der Lösung aus dem Paper *Efficient map-matching of large GPS data sets- Tests on a speed monitoring experiment in Zurich* (2004) von Fabrice Marchal und Kay W. Axhausen.

Das Abbilden der Messwerte auf die vom N Route MapMatching gefundene Route wird durch mehrere zeit- und positionsbasierte Interpolationen und Abbildungsschritte umgesetzt. Zwischen den einzelnen Abbildungsschritten wurden Optimierungen eingebaut, so dass die GPS Punkte und Messwerte nicht nur aus mathematischer Sicht auf die Route positioniert werden, sondern auch die echte Bewegung des Messfahrzeugs realistischer berücksichtig wird. Hierfür kann der Anwender die Route gegebenenfalls manuell korrigieren. JXMapMatchVer3 erkennt automatisch kritische Abschnitte beim Abbilden der GPS Punkte und Messwerte und nutzt eine eingebaute

Optimierung (Stichwort: *reorder* und *Unique*).

Die Aggregationsmöglichkeit der Messwerte ist optional und wird in zwei Schritten durchgeführt: 1. Berechnung des Repräsentanten und 2. Gruppenauswahl.
Beide Schritte sind unabhängig und parametrisierbar, so dass der Anwender viele Aggregationsmöglichkeiten hat.
Messwerte, die auf einen vom OSM definierten Straßenabschnitt abgebildet werden, werden in eine Gruppe zusammengefasst. Für diese Gruppe wird ein Repräsentant nach einer von acht implementierten Auswahl- und Rechenarten ermittelt. Anschließend wird mit Hilfe des Repräsentanten eine Gruppe durch eine von drei Auswahlverfahren bestimmt.

Die entworfenen und umgesetzten Programme, Lösungen und Algorithmen ermöglichen dem Anwender konfigurierbar reale und beobachtete Messwerte aufzubereiten. Somit leistet diese Masterarbeit mit ihrem Ergebnis nicht nur einen *Proof of Concept* einer theoretischen Idee. Das Resultat dieser Arbeit kann und wurde bereits während ihrer Erstellungszeit praktisch eingesetzt, um die Qualität und Aussagekraft von Simulationen von V2X-Appliaktionen mit der Mobilfunkkommunikation zu steigern.

8.1. Ausblick

Während der Erstellung dieser Masterarbeit sind weitere Aufgaben und Optimierungsmöglichkeiten aufgekommen, die aufgrund von zeitlichen Vorgaben nicht im Detail untersucht und implementiert werden konnten. Sie sollen aber dennoch in diesem Kapitel erwähnt werden, um in weiteren Arbeiten und Anpassungen der bisher umgesetzten Programme berücksichtigt zu werden.

In der aktuellen Version des JXMapMatchVer3 werden die Messwerte mit Hilfe ihrer Zeitstempel linear zwischen die GPS Punkte interpoliert. Diese Verteilung der Messwerte stellt eine gleichförmige Bewegung des Messfahrzeugs nach. Aus mathematischer Sicht würde das MapMatching genauere Ergebnisse liefern, wenn zuerst eine rundere Bewegung durch die GPS Positionen gebildet würde. Für die Berechnung eines durchgängig

nicht eckig gefahrenen Weges könnten Näherungspolynome und kubische Splines herangezogen werden. Nachdem der tatsächliche und runde gefahrene Weg nachgestellt wurde, könnte anschließend die Geschwindigkeit des Fahrzeuges zwischen den GPS Punkten neu berechnet und ebenfalls durch Näherungspolynome und kubische Splines geglättet werden. Zum Schluss werden die Messwerte auf die OSM Route interpoliert.
Da davon auszugehen ist, dass eine Messung mit einer vorsichtigen und defensiven Fahrweise durchgeführt wird, würden sich die MapMatching Ergebnisse mathematisch verbessern, aber nur minimal andere Messwerte als Folge haben. Deshalb sollte der Aufwand der durchzuführenden Arbeit im Verhältnis zum daraus resultierenden Mehrwert genau abgewogen werden.

In Kapitel 6.1.3 wird auf die Problematik des MapMatchings in Abbiegungen eingegangen. Es wird festgestellt, dass das Erkennen von den beschriebenen kritischen Situationen, ihrer Position und Größe sehr wichtig für eine gute Lösung sind. Ein erster implementierter Lösungsansatz für Innen-MapMatching Abbiegungen wird am Ende von Kapitel 6.1.3 vorgestellt (*CheckBox Unique*). Ein Ansatz für kritische Situationen für Innen- und Außen-MapMatching in Abbiegungen könnte wie folgt lauten:
Es sollte das Verhältnis der Entfernungen von zwei aufeinander folgenden nicht gematchten und gemachten GPS Punkten berechnet werden. In Außen-Abbiegungen (siehe Abbildung 6.7) ist die Entfernung von den gematchten GPS Punkten deutlich größer als die Entfernung zwischen den nicht gematchten GPS Punkten. Umgekehrt ist die Entfernung in Innen-Abbiegungen (siehe Abbildung 6.6) bei den nicht gematchten GPS Punkten deutlich kleiner. Mit dieser Schlussfolgerung könnten weitere kritische Situationen automatisch erkannt werden.

Anhang A.

Spaltenerklärung

Der JXMapMatcherVer3 und der DataUnion erstellen neue csv Dateien, die Informationen von Messwerten in Tabellenform angeben. Es folgt eine Erklärung der Spalten.

Primäre Ausgabedatei des JXMapMatchVer3 (*.csv):

- **type:** Messart des Messwertes - Gleiche Fahrtrichtung wie die edge (Real) oder entgegen der Fahrtrichtung (BackDirektion) (Siehe Kapitel 3.2.3)
- **down_up:** Verbindungsrichtung , Messwert vom down- oder upstream
- **timestamp:** Zeitstempel des Messwerts
- **matched_latitude:** Gematchter Breitengrad (GPS)
- **matched_longitude:** Gematchter Längengrad (GPS)
- **unmatched_latitude:** Ungematchter Breitengrad (GPS)
- **unmatched_longitude:** Ungematchter Längengrad (GPS)
- **startNode_id:** id der Start node der edge
- **endNode_id:** id der End node der edge

- **edge_id_str:** id der edge
- **length_in_edge:** Längenposition des Messwertes auf der edge in Metern
- **length_of_edge:** Länge der edge in Metern
- **matched_distribution_in_WayPart:** Längenposition des Messwertes auf der edge, normiert von 0 bis 1
- **datarate:** Datenrate des Messwerts
- **delay:** Latenz des Messwerts
- **loss_rate:** Paketverlustrate des Messwerts
- **matchedLinkNr:** Gruppen id
- **WCDMA_Ch:** Modeminformationen, Channel bei 3G
- **WCDMA_SC:** Modeminformationen, Primary Scramblingcode bei 3G
- **GSM_CellId:** Modeminformationen, CellID bei 2G
- **GSM_LAC:** Modeminformationen, Location Area Code bei 2G

Primäre Ausgabedatei des DataUnion (*.csv):
Diese Datei hat die gleichen Spalten wie die primäre Ausgabedatei des JXMapMatch-Ver3. Jedoch ist folgende Spalte zusätzlich enthalten:

- **matchedLinkNrGlobal:** Gruppen id, eindeutige Gruppen id über alle primären Ausgabedateien des JXMapMatchVer3

Sekundäre Ausgabedatei des DataUnion (*.Sum.csv):
Diese Datei enthält pro Zeile Informationen über die Messwert Gruppen. Viele Spalten sind für die Datenrate, Latenz und Paketverlustrate vorhanden. Dies wird mit einem * abgekürzt. Folgende Spalten sind neu:

- **count:** Anzahl der Messwerte in einer Gruppe
- **count_*:** Anzahl der gültigen Messwerte für die Datenrate/Latenz/Paketverlustrate
- **min_*:** Kleinster Wert für die Datenrate/Latenz/Paketverlustrate
- **stde_*:** Standardabweichung für die Datenrate/Latenz/Paketverlustrate
- Die restlichen Spalten geben den Repräsentanten an (sieh Kapitel 7.1)

Anhang B.

Datenträger

Literaturverzeichnis

[fk14] FABRICE MARCHAL AND KAY W. AXHAUSEN: *Efficient map-matching of large GPS data sets-Tests on a speed monitoring experiment in Zurich*. 2004

[go14] NORBERT GOEBEL AND MARKUS KOEGEL AND MARTIN MAUVE AND KALMAN GRAFFI: *Trace-based Simulation of C2X-Communication using Cellular Networks*. 2014. Obergurgl, Austria 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS 2014) - Special Session on VANETs and ITS3

[bi15] RAPHAEL BIALON: *Coupled simulation of road traffic, V2X-applications and V2X-communication via mobile cellular networks*. 2015

[go08] NORBERT GOEBEL: *Ein Demonstrator für ein Peer-To-Peer-basiertes Verkehrsinformationssystem*. 2008

[li93] LIPPE, Peter von der: *Deskriptive Statistik* 1993. ISBN 978-3437402685

[Ja14] ULLENBOOM, Christian: *Java ist auch eine Insel: Programmieren mit dem Standardwerk für Entwickler, aktuell zu Java 8. (Galileo Computing)* 2014. ISBN 978-3836228732

[ku08] KUROSE, James F.; ROSS, Keith W.: *Computernetzwerke - der Top-Down-Ansatz (4. Aufl.).* Pearson Studium, 2008. ISBN 978–3–8273–7330–4

[sp15] *spiegel - Freightliner Inspiration: Daimler testet selbstfahrenden Lkw im Verkehr*. http://www.spiegel.de/auto/aktuell/freightliner-daimler-testet-

selbstfahrenden-lkw-in-nevada-a-1032293.html, 2015.

[ze15] *zeit - Daimler darf selbstfahrende Lkw auf der Straße testen*. http://www.zeit.de/mobilitaet/2015-05/daimler-selbstfahrende-lkw-usa, 2015.

[au14] *auto-motor-und-sport - Lkw von Morgen fährt selbst*. http://www.auto-motor-und-sport.de/news/mercedes-future-truck-2025-autonom-in-die-zukunft-8441794.html, 2014.

[ne14] *newsplay - Selbstfahrender LKW auf der IAA vorgestellt*. http://www.newsplay.de/video/Newsplay/Wirtschaft/video-Selbstfahrender-LKW-auf-der-IAA-vorgestellt-future-truck-Mercedes-Benz-Verkehr-802705.html, 2014.

[he15] *heise - Apple beantragt Zugang zu Auto-Testanlage in Kalifornien*. http://www.heise.de/newsticker/meldung/Apple-beantragt-Zugang-zu-Auto-Testanlage-in-Kalifornien-2779956.html, 2015.

[wd15] *wdr - Wuppertal bekommt erste deutsche Teststrecke für selbstfahrende Autos*. http://www1.wdr.de/themen/infokompakt/nachrichten/nrwkompakt/teststrecke-wuppertal-100.html, 2015.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 14.09.2015 Adrian Skuballa